

全球资深软件开发专家亲自执笔，AndEngine创始人Nicolas Gramlich作序鼎力推荐  
理论知识系统而全面，讲解了Android游戏开发的方方面面；实战性强，以经典游戏案例为驱动，全面展示了Android游戏开发的流程、方法和技巧，是系统学习Android游戏开发的经典教程

Learning Android Game Programming  
A Hands-On Guide to Building Your First Android Game

# Android游戏开发 实践指南

(美) Rick Rogers 著  
爱飞翔 译



机械工业出版社  
China Machine Press



## 开发优秀的Android手机游戏!

这是一本适合有一定Android开发基础的读者快速而系统地学习Android游戏开发的经典教程, 全书通过一个完整的经典游戏案例讲解了Android游戏开发所必须掌握的理论知识, 以及其流程、方法和技巧, 旨在帮助读者开发出优秀的游戏产品。书中对各种主要类型的手机游戏进行了剖析, 并给出了翔实的案例代码, 读者能从中迅速获得实战经验。

值得强调的是, 本书以AndEngine开源引擎为基础, 将游戏制作中的诸多话题巧妙地串联起来, 使读者在阅读过程中系统地学习场景、图层、图形绘制、精灵、动画、文本绘制、用户输入、地图、粒子系统、物理效果、人工智能、碰撞检测、计分等开发手机游戏所必备的技能。同时, 作者投入大量篇幅用于讲述制作关卡、美工与声音所需的各种工具及其用法, 读者可以借此了解到策划、美工与声音制作的具体流程, 从而对手机游戏各项元素的整合有一个全局的把握。

此外, 本书演示了《少女大战吸血鬼》(Virgins Versus Vampires, 简称V3)的实现过程, 你可以从Android Market下载免费的《Virgins Versus Vampires》游戏, 本书将会教你如何开发这个游戏。

## 通过阅读本书, 可学到以下知识:

- 使用免费的Android代码编写工具、图片编辑工具以及声音制作工具
- 实现Android游戏的核心部分: 游戏主逻辑循环
- 运用场景切换效果与实体修改器将游戏变得更加生动活泼
- 制作位图与矢量图, 制作精灵与动画效果
- 将触摸、多点触摸、键盘、语音识别、加速度计、定位器及罗盘等用户输入手段集成到游戏中
- 使用瓦片地图构建虚拟的游戏世界
- 创建、保存并复用各种功能强大的粒子效果
- 搜寻、获取、修改与使用背景音乐及音效
- 用Box2D实现逼真的物理效果模拟
- 使用AI技术将游戏变得更加灵活、更加有趣
- 构建根据游戏间元素的互相碰撞而计分的程序框架



客服热线: (010) 88378991, 88361066  
购书热线: (010) 68326294, 88379649, 68995259  
投稿热线: (010) 88379604  
读者信箱: hzsj@hzbook.com

PEARSON

www.pearson.com

华章网站 <http://www.hzbook.com>

网上购书: [www.china-pub.com](http://www.china-pub.com)

上架指导: 计算机/程序设计/移动开发

ISBN 978-7-111-39154-8



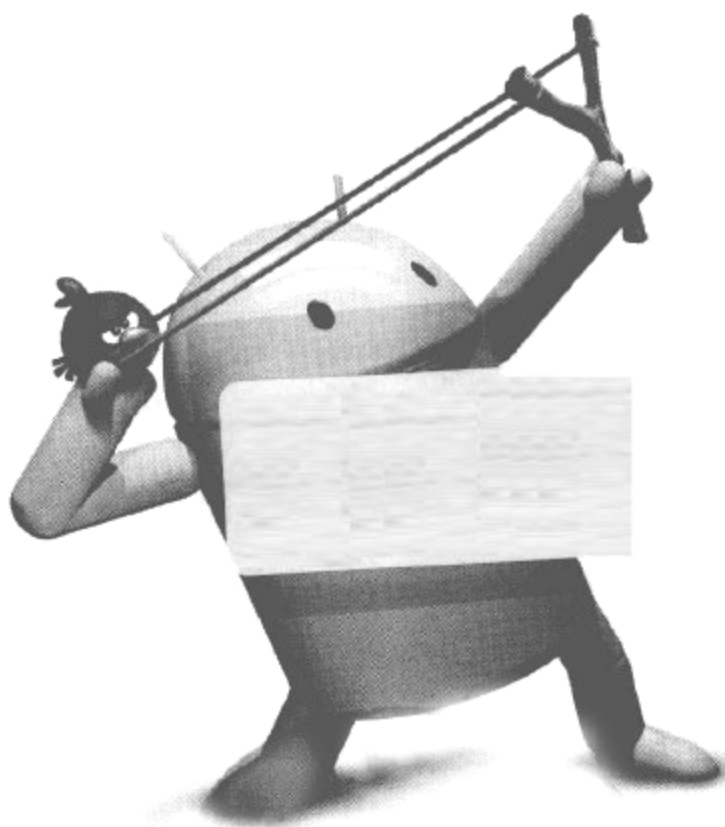
9 787111 391548

定价: 79.00元

**Learning Android Game Programming**  
A Hands-On Guide to Building Your First Android Game

# Android游戏开发 实践指南

(美) Rick Rogers 著  
爱飞翔 译



机械工业出版社  
China Machine Press

本书是一本经典的 Android 游戏开发教程,由资深软件开发专家亲自执笔,AndEngine 引擎创建者作序推荐。

书中以开源引擎 AndEngine 为基础,不仅以专题的形式巧妙地将 Android 游戏开发必须掌握的各项关键技术(场景、图层、图形绘制、精灵、动画、物理效果、粒子系统、碰撞检测、关卡设计、美工、声音……)串联起来,系统地讲解了 Android 游戏开发者应该学习的理论知识,而且用一个完整的案例贯穿全书,将游戏开发的各项要素整合到一起,对 Android 游戏开发的方法和流程做了一个全景展示,可操作性极强。

全书共分 17 章:第 1 章介绍手机游戏的概况和类型;第 2 章~第 15 章各章都会阐述一个与游戏开发相关的话题,其中包括游戏的要素与游戏开发的工具,游戏逻辑循环的概念和用 AndEngine 开始开发游戏的方法,场景、图层、场景切换与实体修改器,精灵和动画精灵的绘制方法,文本和用户输入,瓦片地图,粒子系统,声音,物理效果,人工智能,计分与碰撞以及多媒体扩展包;第 16 章介绍如何通过完善现有功能使游戏更有趣;第 17 章介绍游戏的测试与发行。每章最后都有练习题,可帮助读者边学边练,迅速提高技能,书最后提供了习题答案。

Authorized translation from the English language edition, entitled *Learning Android Game Programming: A Hands-On Guide to Building Your First Android Game*, 9780321769626 by Rick Rogers, published by Pearson Education, Inc., Publishing as Addison-Wesley, Copyright © 2012.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD., and CHINA MACHINE PRESS Copyright © 2012.

本书封底贴有 Pearson Education (培生教育出版集团)激光防伪标签,无标签者不得销售。

封底无防伪标均为盗版

版权所有,侵权必究

本书法律顾问 北京市展达律师事务所

本书版权登记号:图字:01-2012-1277

图书在版编目(CIP)数据

Android 游戏开发实践指南/(美)罗格斯(Rogers, R.)著;爱飞翔译.——北京:机械工业出版社,2012.8

书名原文:Learning Android Game Programming: A Hands-On Guide to Building Your First Android Game

ISBN 978-7-111-39154-8

I. A… II. ①罗… ②爱… III. 移动电话机—游戏程序—程序设计—指南 IV. ① TN929.53-62 ② TP311.5-62

中国版本图书馆 CIP 数据核字:(2012)第 160070 号

机械工业出版社(北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑:关敏

北京市荣盛彩色印刷有限公司印刷

2012 年 9 月第 1 版第 1 次印刷

186mm×240mm·27.25 印张

标准书号:ISBN 978-7-111-39154-8

定价:79.00 元

凡购本书,如有缺页、倒页、脱页,由本社发行部调换

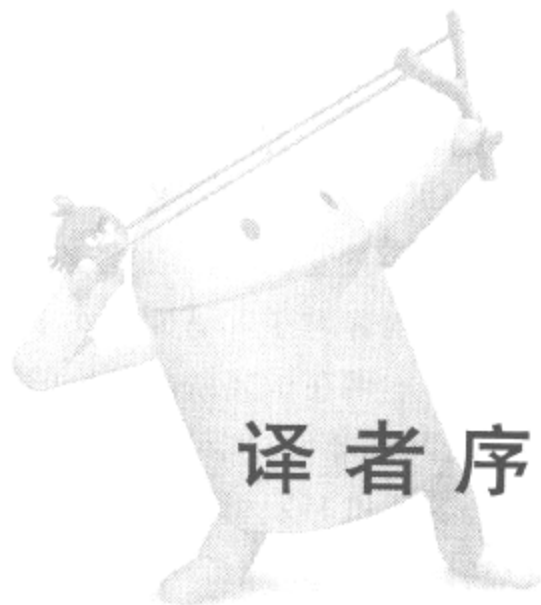
客服热线:(010) 88378991; 88361066

购书热线:(010) 68326294; 88379649; 68995259

投稿热线:(010) 88379604

读者信箱:hzjsj@hzbook.com





## 译者序

随着智能手机的兴起，各种手机平台上的软件开发逐渐形成一项庞大的产业。智能手机的高度可扩展性决定且催生了一系列丰富多彩的软件产品。Android 系统是继 Symbian 与 iOS 之后又一个拥有庞大手机终端的平台，在其上开发软件不仅有利于开发者展示自身的兴趣、实现自身的价值，同时也可以为开发者带来相当可观的收入。与其他流行开发平台相比，Android 平台所使用的开发语言与 Java 语言高度类似，同时该平台也提供了一整套详尽的 API 文档与相关教程，这是它能够吸引大量初学者的原因之一。面向对象的高层语言搭配底层不断优化的 Dalvik 虚拟机，使得 Android 开发环境在程序管理与执行效率之间取得了恰当的平衡。不论是在 Linux、Windows 还是 Mac OS X 系统上，只需花不到一个小时，就可以搭建好用于制作 Android 软件的一整套开发环境。方便且友好的模拟器与调试机制，更是极大地简化了开发过程中许多繁杂的步骤。

与应用程序开发相比，游戏开发显得更加灵活多变，它不仅关注代码技术细节，同时还牵涉周边的策划、美工、音乐等主题。游戏开发对于打造团队凝聚力来说，是个很好的磨练过程，同时也能对开发者自身的职业规划产生积极的影响。将一款制作精良的游戏发布到各种具有大量潜在用户的 Android 软件商店之中，不仅可以提振制作团队与开发者的名声，也可以帮团队和开发者获得长期且稳定的各种广告收入与下载收入。本书正是这样一本面向广大手机游戏开发者的专著。作者是游戏行业内具有多年经验的开发者，在书中带领读者由浅入深、逐步熟悉手机游戏的全套制作流程。书中以 AndEngine 开源引擎为基础，将游戏制作中的诸多话题巧妙串联起来，使读者在阅读过程中系统地学习场景、图层、图形绘制、精灵、动画、文本绘制、用户输入、地图、粒子系统、物理效果、人工智能、碰撞检测、计分等制作手机游戏所必备的技能。同时，作者投入大量篇幅用于讲述制作关卡、美工与声音所需的各种工具及用法，读者可以借此了解策划、美工与声音制作的具体流程，从而对手机游戏各项元素的整合有一个全局

的把握。

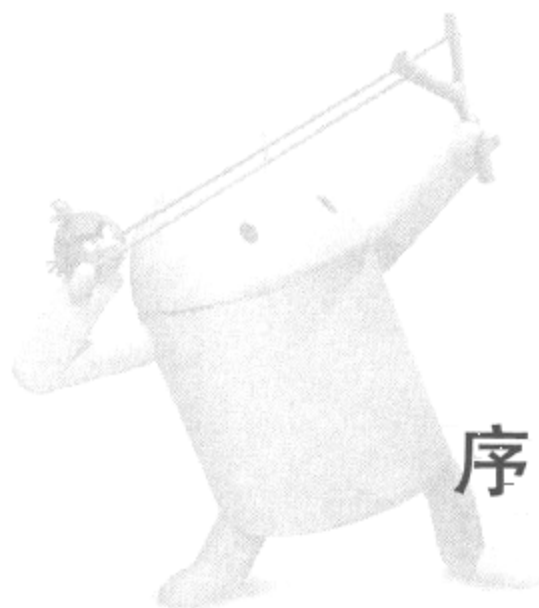
与同类图书或教程相比，本书的优点之一就是适用面广。不论是个人游戏开发者，还是游戏开发团队，或专业的游戏制作公司，都可以在本书中找到自己需要的内容，而且这些内容对于非 Android 平台的游戏开发来说，很多也是能够通用的。尤其值得赞赏的是，本书在讲述游戏制作的过程中，一直紧扣 AndEngine 制作引擎，使读者在学会游戏制作的同时，也可掌握游戏引擎的使用方法。如果读者想要深入研究 AndEngine 这款流行的手机开发引擎的话，可以把本书当做一本绝佳的入门教程，不仅如此，读者还可以通过学习 AndEngine 设计与架构，来对比研究其他流行的同类型手机游戏引擎，甚至制作自己专用的引擎，从而为积累行业经验打下坚实的基础。这可以说是一举两得，读者可以在学会游戏制作的同时，又掌握游戏引擎技术。此外，书中大量讲述了资源授权问题以及游戏发行与推广等相关知识，使读者在认真阅读本书并充分研究每章所附习题之后，能够在短时间内制作出符合商业标准的手机游戏产品来。总之，本书是一本内涵丰富且可读性与可操作性强的优秀手机游戏制作教程。

在本书的翻译过程中，我们忠实于原著的同时尽量保持语言表达的流畅。对于大量的游戏制作领域术语，都给出了英文原文及适量的注解，对于尚没有统一中文名的术语，则尽可能地选择大家接受度高的翻译方法。

本书由爱飞翔翻译，王鹏、舒亚林及张军也参与了部分翻译工作。翻译过程中得到了岳阳先生与网友 166MMX 的帮助，在此表示感谢。由于译者水平有限，错误和不当之处在所难免，敬请广大读者批评指正。

爱飞翔





## 序

2010 年年初，在 Android 平台上还没有强大而且免费的 2D 游戏引擎，然而时至今日，开发者已经可以从众多的引擎中选择最合适的，来展现其个人创意。

每天有 50 万台 Android 设备被激活，每一台都能在几分钟之内配置好。毫不夸张地说，每一天都是非常关键的，市场形势正在使成功的商业模式由大公司转移到独立开发者，他们一个晚上就能做出来“愤怒的小鸟<sup>①</sup>二代”。

我创建 AndEngine 的目标是提供一个免费且易用的游戏开发框架，它可以让经验不多的开发者迅速投入到快速增长的市场中来，同时又不会限制专业级游戏开发者的创意。

现在市面上已经有两百多个游戏是由 AndEngine 开发的，其代码也已执行了超过 100 万次。它可以让开发者成功地制作出满足广大客户需求的产品，也能给他们带来稳定的收入。2011 年中期，我受雇于 Zynga 公司<sup>②</sup>，自此，AndEngine 的专业水准日臻成熟。

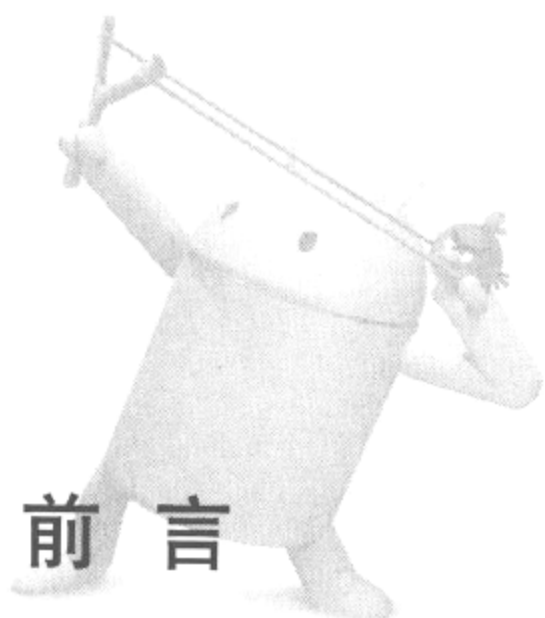
越来越多的开发者需要了解 Android 平台游戏开发的知识，这意味着，他们强烈要求阅读一本可靠的指导教程。Rick Rogers 写的这本书很棒，它以通俗的语言讲解了常见的游戏开发问题。书中以 AndEngine 作为技术基础来制作游戏，Rick 通过一个完整的范例游戏指导读者进行开发，整个开发过程既涵盖了所有初学者应该掌握的知识点，也为专业级游戏开发者提供了经验与心得。敬请品读此书！

Nicolas Gramlich, AndEngine 创始人

① 《愤怒的小鸟》(Angry Birds) 是芬兰公司 Rovio Mobile 推出的一款益智游戏。在游戏中玩家控制一架弹弓发射小鸟来打击建筑物和小猪，并以摧毁关中所有的小猪为最终目的。——译者注

② 一个社交游戏公司，于 2007 年 6 月成立。Zynga 开发的游戏多半是网页游戏，并发布于 Facebook 以及 MySpace 一类的社交网站。公司的总部在美国旧金山。——译者注





## 本书要点

这是一本讲解如何编写 Android 手机游戏的书。只要读者略有开发 Android 应用的经验，就可以通过阅读本书，将这种经验同 AndEngine 开源游戏引擎结合起来，制作出 2D 手机游戏。无论要写什么类型的游戏，本书都会提供范例，并逐步讲解它们。本书旨在让你熟悉 AndEngine 并且尽快发布游戏，其中许多例子都是为了支持一个范例游戏：《少女大战吸血鬼》（Virgins Versus Vampires, V3）的开发。

本书开篇的第 1 章会介绍手机游戏的概况、流程度、游戏的类型以及游戏策划的范例。接下来的数章，每章将会针对一个游戏开发相关的话题展开论述。

- 第 2 章讲述了用来开发游戏的工具，包括进行代码开发、美工和声音制作所用的工具。
- 第 3 章介绍了游戏逻辑循环的概念，并且演示了如何用 AndEngine 开始开发游戏。
- 第 4 章深入探讨图形绘制，详解了 AndEngine 所提供的场景切换与实体修改器机制，运用它们可使游戏的显示效果丰富多彩。
- 第 5 章再进一步深入游戏中的位图和矢量图形的绘制，演示了如何绘制精灵。
- 第 6 章介绍了构建动画精灵的简单方法，以及如何让物体动起来。
- 第 7 章给出使用 AndEngine 以各种方式在游戏中显示文本的范例。
- 第 8 章研究了 Android 游戏可用的用户输入选项，包括触摸、多点触摸、键盘、语音识别、加速度计、定位器与方向检测器。
- 第 9 章讲述了 AndEngine 如何载入和使用瓦片地图，以及如何用瓦片集去构造无限广阔的虚拟世界。
- 第 10 章演示了内建于 AndEngine 的粒子系统，并展示了如何用 XML 文件定义

与保存粒子效果。

- 第 11 章展示了如何用 AndEngine 来查找、获取、修改、使用背景音乐与音效。
- 第 12 章研究了物理引擎 Box2D。它与 AndEngine 一起，使开发基于物理交互对象的游戏更加容易。
- 第 13 章研究了可以让游戏更加智能、更加好玩的人工智能技术。
- 第 14 章搭建了一个基于游戏元素间碰撞的计分框架。
- 第 15 章探究了 AndEngine 可以利用扩展而完成的功能，例如创建 Android 活动桌布，播放 MOD 音乐文件，创建“增强现实游戏”，以及在多人游戏的玩家中进行通信。
- 第 16 章通过完善现有功能或增加新功能使游戏更具可玩性，并至此结束本书的范例游戏开发。
- 第 17 章讲述了为确保游戏顺利发行而需要做的事情，然后告诉读者如何发行与推销游戏。
- 附录提供了每章末尾习题的答案。

本书最好按顺序阅读，但如果读者觉得跳读更合适的话，那样也可以。每一个主题基本上都作为独立的概念来讲述，但是如果需要引用其他章的概念来讲述它，那么那些概念也会被提到。

对于读者来说，本书的目的很简单：玩得开心。本书的写作精神是，游戏应该是有趣的，而且开发游戏本身也应该有趣。愿你的游戏能够在 Android Market 的“热门下载”列表中登上榜首。

## 本书的目标读者

如果读者迫不及待地想要为 Android 设备开发 2D 游戏，并且至少具备一些用 Android SDK 和 Java 进行 Android 应用开发的经历，那么本书就很适合你了。本书介绍了手机游戏的基本主题，并演示了如何用 AndEngine 游戏引擎去实现它们。要学习这些范例的话，不一定必须是个 Android 开发高手，但需要熟悉 Android 的基本概念 [例如活动 (Activity)、服务 (Service)、意图 (Intent) 等]，同时需要能熟练地阅读与编写 Java 代码、使用 Android SDK。

## 本书的范例代码

书中代码可在本书网站获取：

<http://www.informit.com/title/9780321769626>。

也可以在配套的-github 站点获取:

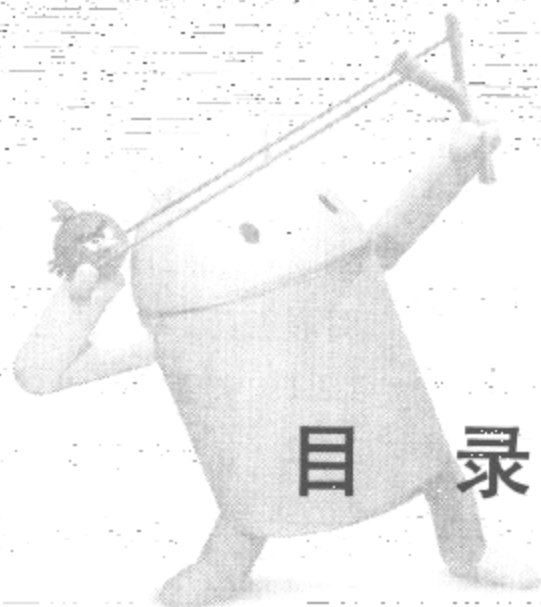
<https://github.com/portmobile/LAGP-Example-Code>。

## 致谢

衷心感谢以下人士协助笔者完成此书。

- Nicolas Gramlich, 他为了打造世界级的 Android 游戏引擎创立了 AndEngine, 并将努力工作的成果作为开源项目分享给大家 (也包括每位读者在内)。Nicolas 慷慨地允许我使用 AndEngine 作为本书的基础, 并且自愿审阅了本书的草稿。他持续地完善与扩充 AndEngine, 并将这些改进提供给本书使用。
- Trina MacDonald, 本书的组稿编辑, 正是她提议写一本 Android 游戏开发相关的书。她是位出色的项目经理, 没有她不知疲倦的辛苦工作, 本书不可能与大家见面。
- James Becwar、Stephan Branczyk、Jason Wei, 本书的技术编辑。不可能有比他们更好的技术审校者了。他们三位令我在写书时能够本着实事求是的态度, 确保技术信息的准确性, 也保证了所有代码都能正确运行。
- Songlin Qiu 是位出色的拓展编辑。本书行文清晰流畅之处, 都要归功于她在审稿时提供的许多宝贵建议。如果读者发现文字有难懂之处, 恐怕要怪我未采纳其中某些建议。
- 本书付诸刊印, 要感谢生产经理 Julie Nahil 与文字编辑 Jill Hobbs 的艰苦努力。我能不过多地分散精力, 始终专注于本书的写作并最终将其完成, 很大程度上要归功于他们。
- Olivia Basegio 身为助理编辑, 做了许多工作。她将本书草稿交给合适的人员去审阅, 并将其初步裁切。她也编排了本书的插图与许可信息, 并在我忘记事情时提醒我。没有 Olivia 的帮助, 我们只能看到一些零碎的资料, 而不可能有一本完整的书呈现在眼前。
- 这里无法详尽地列出在遇到困难时鼓励我的每位朋友与家人的名字。我要特别感谢女儿 Allison Jackson 和 Katie Kehr, 她们一如既往的乐观使我相信最终能完成此书, 她们用自己的生活态度为我树立了榜样。
- Susie Jackson, 我的妻子, 为我做的每件事情提供灵感, 也包括写这本书。她棒了! 能够和她结婚我感到很幸运。她在生活中带给我的自信与积极态度, 让我能够坐在办公室里努力创作。再次感谢你, Susie。





# 目 录

译者序  
序  
前言

## 第1章 手机游戏 / 1

- 1.1 手机游戏市场 / 2
- 1.2 电脑游戏的世界 / 3
  - 1.2.1 游戏类型 / 3
  - 1.2.2 适合于手机的游戏 / 5
  - 1.2.3 典型的游戏组件 / 6
  - 1.2.4 《少女大战吸血鬼》 / 8
  - 1.2.5 V3 的设计 / 9
- 1.3 AndEngine 范例 / 11
- 1.4 总结 / 12
- 1.5 习题 / 12

## 第2章 游戏要素与工具 / 14

- 2.1 软件开发工具 / 15
  - 2.1.1 Android SDK / 15

- 2.1.2 AndEngine 游戏引擎库 / 16
- 2.1.3 AndEngine 游戏概念 / 17
- 2.1.4 Box2D 物理引擎 / 19
- 2.2 图形工具 / 19
  - 2.2.1 矢量图工具: Inkscape / 20
  - 2.2.2 位图工具: GIMP / 21
  - 2.2.3 动画捕捉: AnimGet / 22
  - 2.2.4 瓦片地图创建工具: Tiled / 23
  - 2.2.5 TrueType 字体创建与编辑工具: FontStruct / 23
- 2.3 声音工具 / 24
  - 2.3.1 音效工具: Audacity / 24
  - 2.3.2 背景音乐工具: MuseScore / 25
- 2.4 初试身手: 制作启动画面 / 26
  - 2.4.1 创建游戏项目 / 27
  - 2.4.2 加入 AndEngine 库 / 27
  - 2.4.3 加入启动画面代码 / 28
  - 2.4.4 用模拟器运行游戏 / 30
  - 2.4.5 用 Android 设备运行游戏 / 31
- 2.5 总结 / 31
- 2.6 习题 / 32

### 第3章 游戏循环与菜单 / 33

- 3.1 游戏循环概述 / 34
- 3.2 AndEngine 的游戏循环 / 35
  - 3.2.1 初始化 Engine 对象 / 35
  - 3.2.2 其他 Engine 类 / 36
- 3.3 为 V3 增加菜单屏幕 / 37
  - 3.3.1 AndEngine 的菜单 / 37
  - 3.3.2 构建 V3 的开始菜单 / 40
  - 3.3.3 创建菜单 / 40
  - 3.3.4 MainMenuActivity 类 / 45
  - 3.3.5 常数与字段 / 46

- 3.3.6 onLoadResources() 方法 / 46
- 3.3.7 onLoadScene() 方法 / 46
- 3.3.8 createStaticMenuScene() 方法与 createPopUpScene() 方法 / 46
- 3.3.9 onKeyDown() 方法与 onMenuItemClicked() 方法 / 47
- 3.3.10 从启动画面切换到菜单 / 47
- 3.4 内存使用 / 50
- 3.5 “退出”选项 / 50
- 3.6 总结 / 50
- 3.7 习题 / 51

## 第4章 场景、图层、场景切换与实体修改器 / 52

- 4.1 AndEngine 的场景 / 53
  - 4.1.1 实体 / 组件模型 / 53
  - 4.1.2 Entity 类 / 54
  - 4.1.3 构造器 / 54
  - 4.1.4 Entity 类的位置相关方法 / 55
  - 4.1.5 Entity 类的缩放相关方法 / 55
  - 4.1.6 Entity 类的颜色相关方法 / 56
  - 4.1.7 Entity 类的旋转相关方法 / 57
  - 4.1.8 管理子对象 / 57
  - 4.1.9 管理 Modifier / 58
  - 4.1.10 其他有用的 Entity 类方法 / 58
  - 4.1.11 Layer 类 / 59
  - 4.1.12 Scene 类 / 59
  - 4.1.13 背景管理 / 60
  - 4.1.14 子 Scene 对象管理 / 60
  - 4.1.15 Layer 对象管理 / 61
  - 4.1.16 上级 Scene 对象管理 / 61
  - 4.1.17 触摸区域管理 / 61
  - 4.1.18 特殊 Scene 类 / 61
  - 4.1.19 用于 Entity 的 Modifier 类 / 62
  - 4.1.20 EntityModifier 类的通用方法 / 63



- 4.1.21 位置相关的 EntityModifier 类 / 63
- 4.1.22 缩放相关的 EntityModifier 类 / 66
- 4.1.23 颜色相关的 EntityModifier 类 / 67
- 4.1.24 旋转相关的 EntityModifier 类 / 67
- 4.1.25 透明度相关的 EntityModifier 类 / 68
- 4.1.26 延迟相关的 EntityModifier 类 / 69
- 4.1.27 Modifier 的组合 / 69
- 4.1.28 EaseFunction / 71
- 4.2 创建游戏第 1 关的场景 / 79
- 4.3 总结 / 84
- 4.4 习题 / 85

## 第 5 章 绘制与精灵 / 86

- 5.1 快速回顾 Entity 类 / 87
- 5.2 绘制线条与矩形 / 88
  - 5.2.1 线条 / 88
  - 5.2.2 矩形 / 88
- 5.3 精灵 / 88
  - 5.3.1 贴图 / 89
  - 5.3.2 效率问题 / 99
  - 5.3.3 复合精灵 / 100
- 5.4 总结 / 104
- 5.5 习题 / 105

## 第 6 章 动画 / 106

- 6.1 动画所需素材 / 107
- 6.2 动画的瓦片贴图 / 108
- 6.3 AndEngine 的动画 / 108
- 6.4 动画范例 / 110
- 6.5 将动画加入 Level1Activity 类 / 114
- 6.6 动画制作的问题 / 122
- 6.7 高级话题：从 3D 模型中制作 2D 动画 / 123

6.8 总结 / 123

6.9 习题 / 123

## 第7章 文本 / 125

7.1 字型与字体 / 126

7.2 载入字型 / 127

7.2.1 Font 类 / 127

7.2.2 StrokeFont 类 / 127

7.2.3 FontFactory 类 / 128

7.2.4 FontManager 类 / 128

7.2.5 Typeface 类 / 128

7.3 AndEngine 中的文本 / 129

7.3.1 AndEngine 中的文本 API / 129

7.3.2 桌面通知 / 132

7.4 定制字型 / 133

7.5 将定制字型加入 V3 / 135

7.6 总结 / 142

7.7 习题 / 142

## 第8章 用户输入 / 144

8.1 Android 与 AndEngine 的输入方式 / 145

8.1.1 字母键盘与袖珍键盘 / 146

8.1.2 触摸 / 146

8.1.3 自定义手势 / 152

8.1.4 屏幕游戏手柄 / 152

8.1.5 加速计 / 153

8.1.6 位置和方向 / 153

8.1.7 语音 / 158

8.2 将用户输入加入 V3 / 161

8.3 总结 / 166

8.4 习题 / 166

## 第9章 瓦片地图 / 168

- 9.1 为何使用瓦片地图 / 169
- 9.2 瓦片地图的类型 / 169
  - 9.2.1 正交瓦片地图 / 171
  - 9.2.2 等距投影瓦片地图 / 171
- 9.3 瓦片地图的结构 / 172
- 9.4 AndEngine 中的瓦片地图 / 172
  - 9.4.1 TMX 与 TSX 文件 / 172
  - 9.4.2 TMXLoader 类 / 172
  - 9.4.3 TMXTiledMap 类 / 173
  - 9.4.4 TMXLayer 类 / 174
  - 9.4.5 TMXTile 类 / 174
- 9.5 瓦片编辑器: Tiled / 175
- 9.6 TMX 文件 / 176
- 9.7 正交瓦片地图游戏:《打吸血鬼》/ 177
  - 9.7.1 WAV 的瓦片地图 / 177
  - 9.7.2 创建 WAV 的瓦片集 / 178
  - 9.7.3 创建 WAV 的瓦片地图 / 179
  - 9.7.4 《打吸血鬼》游戏的代码 / 181
- 9.8 等距投影瓦片地图 / 191
- 9.9 总结 / 191
- 9.10 习题 / 192

## 第10章 粒子系统 / 193

- 10.1 粒子发射器是什么 / 194
- 10.2 粒子系统如何运作 / 195
- 10.3 AndEngine 的粒子系统 / 195
  - 10.3.1 ParticleSystem 类 / 196
  - 10.3.2 ParticleEmitter 类 / 197
  - 10.3.3 ParticleInitializer 类 / 198
  - 10.3.4 ParticleModifier 类 / 199
  - 10.3.5 有用的 ParticleSystem 类方法 / 200

- 10.4 创建粒子系统 / 201
  - 10.4.1 以传统方式创建粒子系统 / 201
  - 10.4.2 以 XML 文件创建粒子系统 / 202
- 10.5 将粒子发射器加入 V3 游戏中 / 206
  - 10.5.1 以传统方式制作 V3 的爆炸效果 / 206
  - 10.5.2 以 XML 文件方式制作 V3 的爆炸效果 / 210
- 10.6 总结 / 211
- 10.7 习题 / 211

## 第 11 章 声音 / 213

- 11.1 如何在游戏中使用声音 / 214
  - 11.1.1 音乐 / 214
  - 11.1.2 音效 / 214
- 11.2 音乐与音效的来源 / 215
- 11.3 音乐与音效制作工具 / 216
- 11.4 音频解码器 / 216
- 11.5 使用 AndEngine 播放声音 / 217
  - 11.5.1 Music 类 / 218
  - 11.5.2 Sound 类 / 218
  - 11.5.3 MusicFactory 类 / 219
  - 11.5.4 SoundFactory 类 / 219
- 11.6 将声音加入 V3 游戏 / 220
  - 11.6.1 创建音效 / 220
  - 11.6.2 创建背景音乐 / 223
  - 11.6.3 修改 V3 游戏的代码 / 225
- 11.7 总结 / 235
- 11.8 习题 / 236

## 第 12 章 物理效果 / 237

- 12.1 Box2D 物理引擎 / 238
  - 12.1.1 Box2D 概念 / 238



- 12.1.2 设定 Box2D / 240
- 12.2 构建物理学游戏的关卡 / 241
- 12.3 AndEngine 与 Box2D / 242
  - 12.3.1 下载 AndEnginePhysicsBox2DExtension 并将其加入游戏项目 / 242
  - 12.3.2 Box2D 的 API / 244
  - 12.3.3 简单的物理效果范例 / 247
  - 12.3.4 关卡加载 / 252
- 12.4 《愤怒的村民》: V3 中的物理学小游戏 / 255
- 12.5 实现 IV 游戏 / 255
  - 12.5.1 创建关卡 / 256
  - 12.5.2 编写 IVActivity.java / 260
- 12.6 总结 / 270
- 12.7 习题 / 270

## 第 13 章 人工智能 / 272

- 13.1 游戏 AI 相关话题 / 273
  - 13.1.1 简单的脚本 / 273
  - 13.1.2 决策树、Minimax 树与状态机 / 273
  - 13.1.3 专家系统或基于规则的决策系统 / 276
  - 13.1.4 神经网络 / 277
  - 13.1.5 遗传算法 / 278
  - 13.1.6 路径查找 / 279
  - 13.1.7 动态困难度调节 / 280
  - 13.1.8 程序化的音乐生成 / 280
- 13.2 实现 V3 游戏的 AI / 281
- 13.3 总结 / 290
- 13.4 习题 / 290

## 第 14 章 计分与碰撞 / 291

- 14.1 计分系统设计 / 292
  - 14.1.1 更新小游戏取得的分数 / 293
  - 14.1.2 记录 5 个最高分 / 293

- 14.1.3 在小游戏场景中显示分数 / 294
- 14.1.4 分数页面的显示 / 295
- 14.2 AndEngine 的碰撞 / 298
  - 14.2.1 AndEngine 的 Shape 碰撞 / 298
  - 14.2.2 Box2D 的碰撞 / 299
- 14.3 开始计算玩家的得分 / 300
- 14.4 《墓地》(第 1 关) 场景 / 300
  - 14.4.1 常量和字段 / 300
  - 14.4.2 onLoadEngine 方法与 onLoadResources 方法 / 303
  - 14.4.3 onLoadScene 方法 / 304
  - 14.4.4 mStartVamp 任务 / 306
- 14.5 《打吸血鬼》 / 307
  - 14.5.1 常量和字段 / 307
  - 14.5.2 onEoadScene 方法 / 308
  - 14.5.3 openCoffin 和 closeCoffin 方法 / 309
- 14.6 《愤怒的村民》 / 310
  - 14.6.1 常量和字段 / 310
  - 14.6.2 onLoadScene 方法 / 311
  - 14.6.3 onEoadComplete 方法 / 312
  - 14.6.4 addStake 方法 / 313
- 14.7 总结 / 314
- 14.8 习题 / 314

## 第 15 章 多媒体扩展包 / 315

- 15.1 下载多媒体扩展包 / 316
- 15.2 动态壁纸 / 317
  - 15.2.1 Android 动态壁纸 / 317
  - 15.2.2 创建 V3 的 Android 动态壁纸 / 318
- 15.3 MOD 格式音乐 / 322
  - 15.3.1 搜寻 MOD 格式的音乐 / 322
  - 15.3.2 XMP MOD 播放器 / 323
- 15.4 多人游戏 / 324

- 15.5 AndEngine 的多点触摸 / 326
- 15.6 增强现实游戏 / 328
- 15.7 总结 / 332
- 15.8 习题 / 332

## 第 16 章 游戏集成 / 334

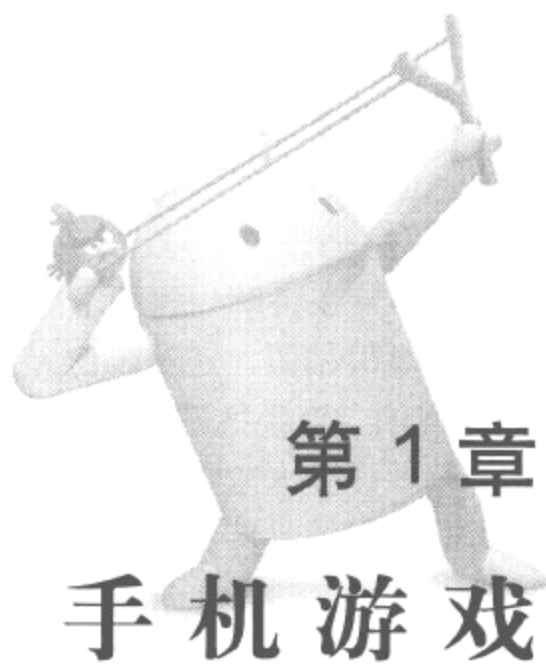
- 16.1 困难度调节 / 336
  - 16.1.1 困难度参数的保存 / 336
  - 16.1.2 困难度参数的设定 / 337
- 16.2 游戏结束画面的代码 / 337
- 16.3 第 1 关：主游戏 / 340
- 16.4 《打吸血鬼》 / 345
- 16.5 《愤怒的村民》 / 347
- 16.6 选项菜单 / 350
- 16.7 总结 / 350
- 16.8 习题 / 350

## 第 17 章 测试与发行 / 352

- 17.1 应用程序商业模式 / 353
- 17.2 测试与发行准备工作 / 354
  - 17.2.1 在实际设备上测试游戏 / 354
  - 17.2.2 考虑加入终端用户许可协议 / 355
  - 17.2.3 向 Manifest 文件加入图标与标签 / 357
  - 17.2.4 关闭记录与调试功能 / 357
  - 17.2.5 在游戏中增加版本号 / 357
  - 17.2.6 获取密钥 / 358
  - 17.2.7 编译与签名最终的 .apk 文件 / 359
  - 17.2.8 测试最终的 .apk 文件 / 359
- 17.3 发行游戏 / 360
  - 17.3.1 Android Market / 360
  - 17.3.2 Amazon App Store / 362
- 17.4 推广游戏 / 363

- 17.4.1 App Store 推广 / 364
- 17.4.2 游戏评论网站推广 / 366
- 17.4.3 手机广告 / 366
- 17.4.4 口碑营销 / 367
- 17.4.5 社交网络推广 / 367
- 17.5 总结 / 367

## 附录 习题解答 / 369



# 第1章 手机游戏

- 1.1 手机游戏市场
- 1.2 电脑游戏的世界
- 1.3 AndEngine 范例
- 1.4 总结
- 1.5 习题

也许没有什么东西能像游戏精神这样无所不在——几乎每个人都喜欢玩某种游戏。而且，正如俗话说“每个人都至少能构思出一部好的小说”，毫不夸张地说，每个人也都至少有一个好的游戏点子。你可能有一个手机游戏的创意，否则就不会翻开这本书了。本书旨在讲解如何编写游戏，使之运行在 Android 手机上。不管要开发的游戏是和范例游戏很相似，还是与之差别很大，本书都会带领读者用流行的 AndEngine 游戏引擎<sup>①</sup>来制作 2D 手机游戏并将其发布到 Android Market<sup>②</sup>上。

对于许多人来说，软件开发本身就是一个游戏——一个无休止的解谜游戏，我们在其中试图找寻实现应用程序创意的最佳方法，同时，更多的谜题将会在调试最初所写代码时出现。当把应用程序本身看做游戏时，我们可以在多个层面享受到整个开发过程的乐趣。快过来一起玩这个软件开发的酒吧，它会实现一直在你脑海中盘旋的创意！

## 1.1 手机游戏市场

在今天的智能手机市场里，游戏可谓是杀手级应用。根据一份数据<sup>③</sup>，在所有年满 13 岁的美国手机用户中，超过 23% 的人在手机上玩游戏——而且这个比例还在增加，尤其是对于超过 6000 万的智能手机用户来说。根据另一份数据<sup>④</sup>，65% 的智能手机用户曾经在手机上玩过至少一款手机游戏。计算一下可知，这意味着现在大约有 4000 万人在智能手机上玩游戏。

制作手机游戏可以是非常盈利的。尽管很难预料哪种游戏会一炮走红，然而快速浏览一下 Android Market 就会发现，成千上万的用户都曾下载了某种类型的游戏。就算每次下载只获利数美元，累计起来也相当可观。人们玩某款游戏一段时间之后，就大多会厌倦，很有可能再去下载新的游戏。

作为众多游戏迷中的一员，笔者将游戏看做最钟爱的手机应用之一。不管是在等人的时候打发时间，还是在公交车上，或者就是想放松几分钟，我都会在手机上玩会儿游戏，这是一件非常愉快的事情。

我认为每款游戏都应该有趣，但这并不意味着游戏不能同时具有教学功能。游戏经常用做教学或广告的手段——为什么不呢？如果学生或者潜在的顾客能够开心地玩一款带给他们益处的游戏，那会是件好事情。

① AndEngine 引擎的网址是 <http://www.andengine.org>。

② Android Market 已更名为 Google Play，网址是 <https://play.google.com/store>，以下不再标出。——译者注

③ comScore ([http://www.comscore.com/Press\\_Events/Press\\_Releases/2010/12/comScore\\_Reports\\_October\\_2010\\_U.S.\\_Mobile\\_Subscriber\\_Market\\_Share](http://www.comscore.com/Press_Events/Press_Releases/2010/12/comScore_Reports_October_2010_U.S._Mobile_Subscriber_Market_Share))。

④ nielsenwire ([http://blog.nielsen.com/nielsenwire/online\\_mobile/the-state-of-mobile-apps/](http://blog.nielsen.com/nielsenwire/online_mobile/the-state-of-mobile-apps/))。



## 1.2 电脑游戏的世界

差不多从有了计算机开始，人们就在上面玩游戏了，而且各式各样游戏都已制作出来。在 Jane McGonigal 的书《Reality Is Broken》中，作者说，大部分游戏都有以下四个属性。

- 目标：游戏清晰地定义了玩家需要达成的目标。重要的是，目标虽具有挑战性，却是可以达成的。理想情况下，玩家总是能充分发挥出他们的能力。目标让玩家在玩游戏时有一种使命感。
- 规则：游戏具有所有玩家必须遵守的规则。规则经常使得目标的达成变得困难，从而鼓励玩家产生创造性的玩法。
- 反馈：游戏必须告诉玩家他们玩得怎么样。实际上，有趣且有创意的反馈系统是让游戏有趣的关键。
- 自愿参与：除非真的愿意去玩，否则它就不算是个游戏。游戏的这个特性意味着玩家要接受游戏所定的目标、规则与反馈系统。

在制作游戏之前，需要思考存在哪些类型的游戏，也要思考哪些类型适合在手机上玩，哪些不适合，还要看看所有电脑游戏的共同要素。

### 1.2.1 游戏类型

游戏开发者不是一开始就打算归类游戏的，而且也没有标准的分类表。不过，久而久之，游戏就由不同的人按不同标准划分归类了。本节所列的分类并不是绝对权威的，而且它们确实在很多地方有重叠。精确的分类并不重要——问题的关键在于可以开发的游戏类型是很多的。

#### 技巧或动作游戏

动作游戏的玩家通常需要使用一些实时技能（例如在恰当的时机跳过障碍物，朝移动的目标射击）以通关。举例来说，其子类型包含以下几种。

- 迷宫游戏。
- 平台动作游戏，此类游戏需要玩家在平台间来回移动以达到某地或阻止敌人。
- 塔防游戏：玩家用某物（例如防守塔）来抵御持续袭来的一群怪物。
- 射击游戏：具有固定的、滑动的或者滚动的游戏场景。
- 对战格斗游戏：两个对手互相打斗。
- 多人格斗游戏：玩家与一群对手打斗（通常有相关的武打动作）。
- 第一人称射击游戏（First-Person Shooter, FPS）：玩家以射击者的视角进行游戏。
- 第三人称射击游戏：同 FPS，但玩家以第三人的视角进行游戏。

#### 策略游戏

策略游戏少有实时的事件，更多的是构思与实现某个战略计划以克服障碍。其子类

型包含以下几种。

- ❑ 回合制游戏：包括传统棋盘游戏。
- ❑ 限时策略游戏：每次移动发生在固定的时间间隔。
- ❑ 大型多人在线角色扮演游戏（MMORP）：经典的《龙与地下城》（Dungeons & Dragons）式游戏的扩充。玩家在其中扮演某种角色并与其他人竞争。

### 冒险或叙事游戏

冒险与叙事游戏是基于丰富的故事情节而构建的。它具有精心构思的角色以及描述玩家游戏目标的故事情节。

- ❑ 简单的 2D 故事游戏经常包括迷宫及与其他游戏实体的互动。
- ❑ 复杂的 3D 故事游戏随着游戏的进行与故事情节的展开，能够展示出不同的视角。有一些已经改编成好莱坞电影。

### 模拟游戏

通常，模拟类游戏描绘某种真实境况，例如一辆玩家可以驾驶的交通工具。此类游戏再现了真实场景的物理效果，除了具有游戏的功能外，还可作为良好的指导手册来用。其子类型包含以下几种。

- ❑ 体育模拟游戏。
- ❑ 飞行模拟或太空模拟游戏。
- ❑ 驾驶模拟或赛车模拟游戏。
- ❑ 航海模拟或潜水模拟游戏。
- ❑ 生命模拟游戏（与策略游戏有重合）。

### 解谜游戏

许多解谜游戏是文字谜题（例如填字游戏）的直译，然而该类型也包括配对与隐藏物品游戏。复杂的游戏经常将小的解谜游戏当做大游戏的一部分。举例来说，解谜游戏基于如下概念。

- ❑ 基于文字的（例如填字游戏）。
- ❑ 基于数字 / 数学的（例如数独）。
- ❑ 视觉匹配。
- ❑ 隐藏物体（例如扫雷）。
- ❑ 从一组零件中构建。

### 增强现实游戏

为了娱乐而玩游戏当然是好的，然而有时玩游戏却有更大动机。正如 Jane McGonigal 在《Reality Is Broken》中说的那样，有些游戏是为了使生活变得更加舒适而以某种方式

对现实情境进行增强。举例来说，下列游戏都是增强现实游戏（ARG）。

- 《Jetset》：模拟机场的安全检查（帮你在等候安检时消磨时间）。
- 《Chore Wars》：该游戏将家务活变成创意比赛。
- 《World Without Oil》：通过模拟石油极度匮乏的世界，来倡导节能理念。

### 1.2.2 适合于手机的游戏

在这么多的游戏类型当中，我们需要关注的是适合在例如手机和平板电脑等移动设备上玩的游戏。我们也要关注适合小团队开发的游戏。

因为移动设备游戏市场颇具潜力，所以大量的研究和构想都致力于如何制作出一款好的手机游戏。电脑游戏的一般原则仍然适用于手机游戏，同时，良好的手机游戏又有如下特质。

- 不浪费玩家的时间。
- 在游戏过程中提供帮助。
- 游戏目标易于理解。
- 清楚地显示游戏状态。
- 用户持续玩手机游戏的时间很短。
- 游戏的暂停与继续很方便，在必要场合（如来电时），手机能够自行暂停与继续游戏。
- 玩家可以在短时间内取得一定的进展。
- 移动设备有影响游戏性的物理限制。
  - 较小的设备屏幕以及多变的屏幕大小、分辨率、像素密度。
  - 多种用户输入方式（例如单手操作、双手操作、触摸、数字键盘、多点触摸、字母键盘、方向键盘、轨迹球。
  - 有限的运算能力。
  - 有限的电池续航能力（此因素影响耗能的绘图与计算操作）。

就算你有足够的资源去开发一款类似 Halo<sup>①</sup>那样华丽的 3D 第一人称射击游戏，玩家也不太可能像玩 XBox 版本那样，坐下来用智能手机玩上数个小时。更有可能的情况是，玩家玩一会儿，就暂停，几天之后再接着玩。

说到资源问题，制作一款商业游戏需要多少资金？一款单机类型的普通电视游戏<sup>②</sup>需要花费 1000 万美元。多机类型的开发需要花费 2 到 3 倍的资金（据估计，一款复杂的

① 中文名“光环”，是由 Bungie 开发、微软发行的科幻第一人称射击游戏。故事发生在“Halo 宇宙”中。这个“宇宙”是由 Bungie 为这个系列所创造的。——译者注

② 电视游戏（Console Game）是一种用来娱乐的交互式多媒体。通常是指使用电视屏幕为显示器，在“电子游戏机”（Game Console）上执行的游戏，与电脑游戏（PC Game）同属电子游戏的一种。——译者注

游戏需要花费 1 亿美元)。仅是软件开发包 (SDK) 与授权许可协议就要花费一大笔钱。如果仔细想一下专业的 3D 电视游戏, 很容易发现需要开销的地方——3D 美工、动作捕捉、动画、游戏性、用户测试以及软件开发。这些既耗时, 又很昂贵。

本书是教读者和你的一两个伙伴如何开发 Android 平台手机游戏的。Android SDK 是免费的, 在写作本书时, 仅需 25 美元即可在 Android Market 上注册账户, 并向所有 Android 设备用户发售游戏。本书只关注 2D (二维) 游戏, 因为相对于 3D 游戏来说, 它的美工及编程更加简单。就像稍后将要讲到的那样, 所有类型的 2D 游戏, 其结构和组件都是基本相同的。我们需要用一个范例游戏来说明。

### 1.2.3 典型的游戏组件

在介绍具体例子之前, 先看看通用的游戏组件, 游戏需要用代码实现这些组件。以下是范例游戏将会包含的组件。

#### 启动画面 (闪屏)

为了使游戏运行更加流畅, 在某关开始前通常要载入所需图像。在可能持续数秒的载入过程中, 不希望用户一直面对黑屏, 所以需要有启动画面, 以便使用户知道游戏在正常运转。启动画面是可选的, 但为了说明如何制作, 我们的范例游戏还是包括它的。

#### 菜单

游戏运行起来之后, 需要有地方让用户打开各种选项 (例如开关声音, 在游戏中查看帮助)。通常用图形化的菜单屏幕展示各种选项。在用户打开某选项时, 当场执行或者切换到另一个屏幕执行 (例如帮助)。

#### 音乐

对大多数人来说, 音乐对情绪有强烈影响。背景音乐对于营造游戏的气氛, 以及游戏各部分之间的切换, 都非常重要。

#### 音效

音效可以使游戏更加有趣。两个物体碰撞时, 玩家期望听到某种声音。不论是“叮当”声, “砰”的一声, 还是“啪”的一声。本书的范例游戏也为每个范例角色包含了音效。每个恶魔在登场时都会伴随着特有的音效。

#### 时间

大多数游戏都包含时间因素。有的是时钟计时 (完成谜题所得分数与解谜所费时间相关), 有的是与计算机 (或电脑控制的对手) 实时做出的移动展开对抗赛。



在《少女大战吸血鬼》(V3)游戏中,时间因素体现在玩家要在恶魔碰到少女前消灭它们。

### 生命

游戏若耍好玩,必须具有挑战性,所以玩家必须时常遭遇失败。杀掉玩家(以虚拟的方式)是表现失败后果的一种便捷方式。有些游戏在每次进入时会给玩家多条生命,而另外一些(比如V3),仅给玩家一次机会。

### 障碍物

障碍物在不同的游戏中以不同的方式使用。在很多游戏中,玩家要试着到达目标,然而沿途会有障碍物阻拦。在塔防类游戏(以及V3)当中,敌人试图到达目的地,玩家要在沿途放置障碍物。

### 关卡

具有挑战性的游戏是有趣的,然而将不同难度区隔开来是很重要的,这样可以让玩家先从低难度开始玩,随着游戏技巧与经验的提升,逐渐提高难度。设置关卡来达成此目的,已被证明是行之有效的手段。在最初的几关里面,玩家可以学到如何玩此游戏,当新的关卡出现时,玩家为了能通关,必须努力提升自己的游戏技巧。设置关卡也是增加游戏丰富程度的好办法。

### 敌人

游戏中的敌人有时称做实体(尽管在AndEngine中,该词另有所指)。这些角色是玩家要打赢游戏所必须对付的坏人(或其他玩家)。它们与障碍物的区别是,敌人会主动阻挠玩家,而障碍物是被动的。稍后本章会列出V3中出现的敌人,以及对其行为的描述。

### 玩家

当然,在任何游戏中,玩家都是最重要的组件。游戏的全部意义就在于让玩家持续参与其中,并且着迷于此。这样他才会一直玩下去。游戏必须让其觉得有一定难度,但是又不能过于困难,否则玩家会因为沮丧而放弃。为了使玩家保持兴趣,游戏内容必须丰富多变。在玩家获得成就时,也应给其奖励。

### 场景

如果将游戏比做电影,那么展示给玩家的场景就如同电影场景一般。每个场景的背景都不大会改变(尽管玩家的视角有时会变)。同游戏有互动效果的敌人与障碍物会以动画的方式来实现。

### 1.2.4 《少女大战吸血鬼》

塔防类游戏在手机游戏中非常流行。此类游戏易于理解（目标即是阻止坏人），且便于随时中断（暂停/继续），适合在小屏幕上完成，也不需要大量的运算。在产品方面，塔防类游戏所需美工相对简单一些，而且能够有效利用编写电脑游戏时所用的主要元素。当然，也希望游戏有趣才行，所以我们试着在此类游戏中增加一些幽默与具有挑战性的成分。

游戏中需要一个“塔”去守护少女。可以不假思索地说，在漫长的历史中，没有任何东西能够像纯洁那样被守护，所以我们把它当做坏人们的目标。在那个时代，吸血鬼通常被当做坏蛋，所以我们把它加入到游戏中来。也许我们还可以用某种方式完成“善良的吸血鬼”这一桥段，善良与邪恶的矛盾总是能增添游戏的趣味。

图 1.1 展示了正在运行中的游戏画面，该游戏称为《少女大战吸血鬼》。

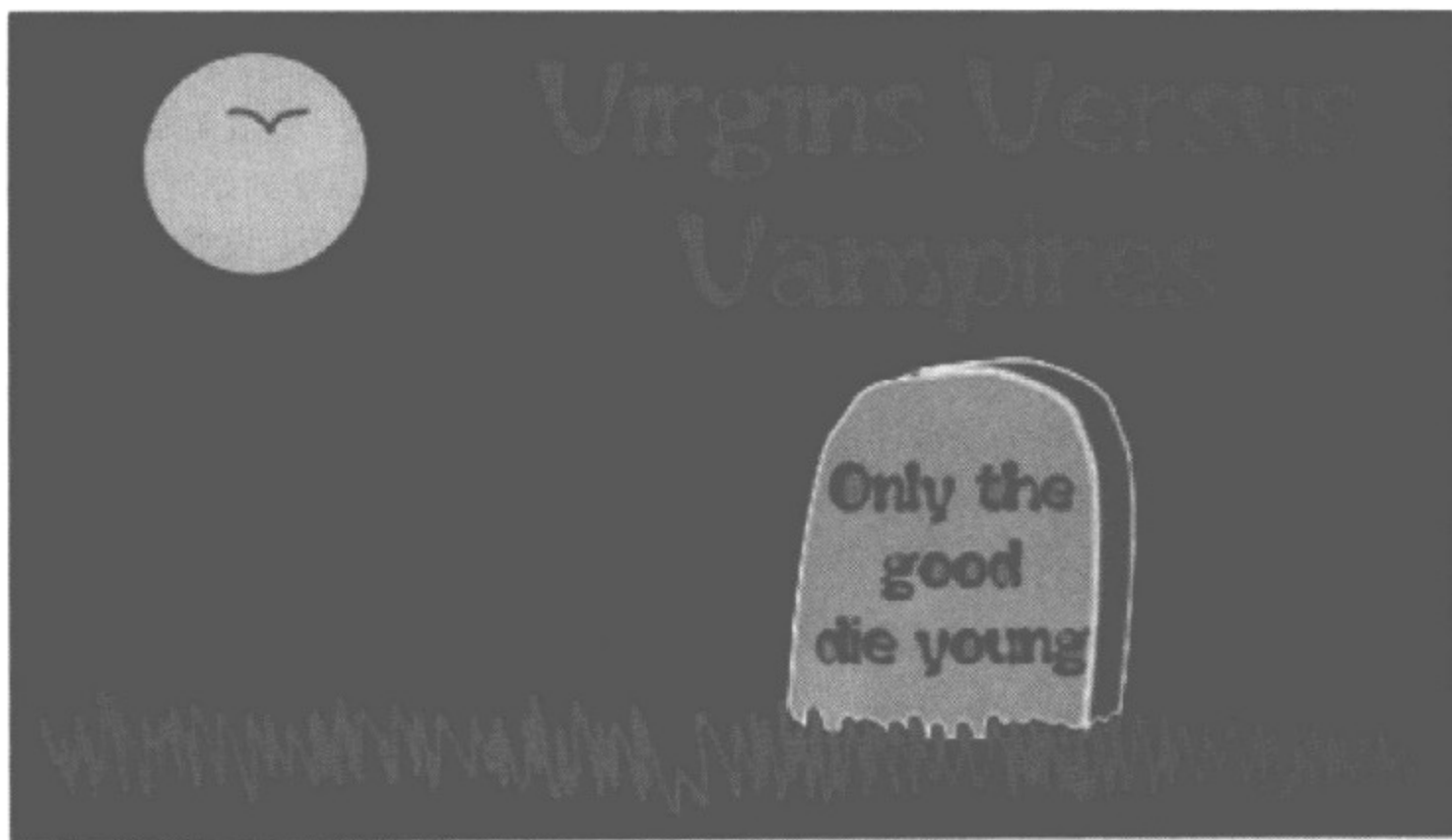


图 1.1 《少女大战吸血鬼》游戏截屏

在 Android Market 上可以免费下载 V3 游戏。现在请花几分钟下载下来并且玩上一会儿，至少打完第 1 关，大概感受一下游戏的风格。

游戏中需要有多种障碍物阻碍吸血鬼的行程，下面这些道具将会派上用场。

#### 子弹

玩家先将其拖至场地中，松开手之后，子弹即发射出去。



- 杀伤力：杀掉任何碰到的敌人。
- 生命期：从子弹发射到飞出屏幕为止。
- 得分：因为很容易干掉一整排敌人，所以每消灭一个吸血鬼的得分不是很高。

### 战斧

战斧也是先拖至场地中，再松开，即可发射。

- 杀伤力：杀掉第一个碰到的吸血鬼。
- 生命期：从战斧发射到碰上吸血鬼为止。
- 得分：因为它只能杀掉一个吸血鬼，所以得分稍高。

### 十字架

玩家将其放置之后，便静止不动，等待吸血鬼从上面走过。

- 杀伤力：杀掉第一个碰到它的吸血鬼。
- 生命期：从十字架放下到杀掉吸血鬼为止。
- 得分：因为必须被吸血鬼偶然撞上才能生效，所以得分最高。

少女们留在屏幕左边的“B小姐女子学校”中，而坏家伙们将从右边出来。玩家的任务是朝这些强盗投掷障碍物以阻拦它们入侵城堡。游戏需要提供获取与放置障碍物的方式，也要能让玩家看到坏人们的行进状况。游戏要有多道关卡，这样玩家可以先从简单的开始，随着游戏策略与能力的提升，再选择较难的。当然，游戏要能够统计并显示分数。

## 1.2.5 V3 的设计

当构思好游戏的创意之后，需要预想一下所需的场景以及它们之间的切换流程。电影剧本写作者与创意作家都会拿铅笔和纸绘制一张故事板<sup>①</sup>，用索引卡片或者方框来代表每个场景，简短地描述其内容，并勾画其大致轮廓。可以用箭头来表示场景之间的切换，并在旁边附上切换的触发时机。

图 1.2 是 V3 游戏的故事板。这里我们特意把故事板做得非常短小，完整游戏的故事板要占据数页内容。

每一个游戏场景也都有一张索引卡为代表，其上勾画出该场景图像的草图。如果是在一张大图上绘制故事板，那么可以把草图放在流程图的旁边。图 1.3 是 V3 游戏第 1 关的索引卡。

① storyboard，又叫情节串连图。——译者注

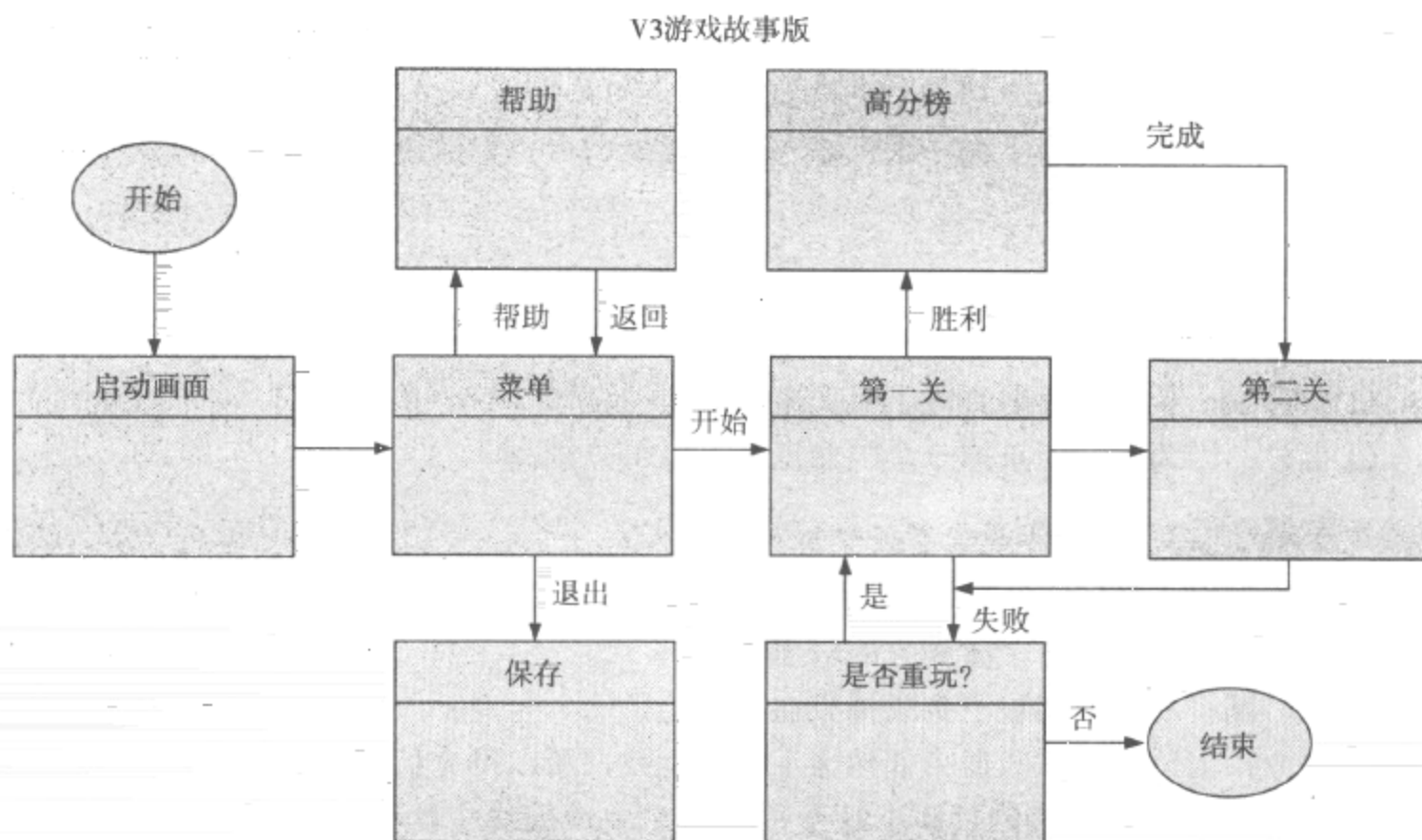


图 1.2 游戏故事板上初步确定的流程图

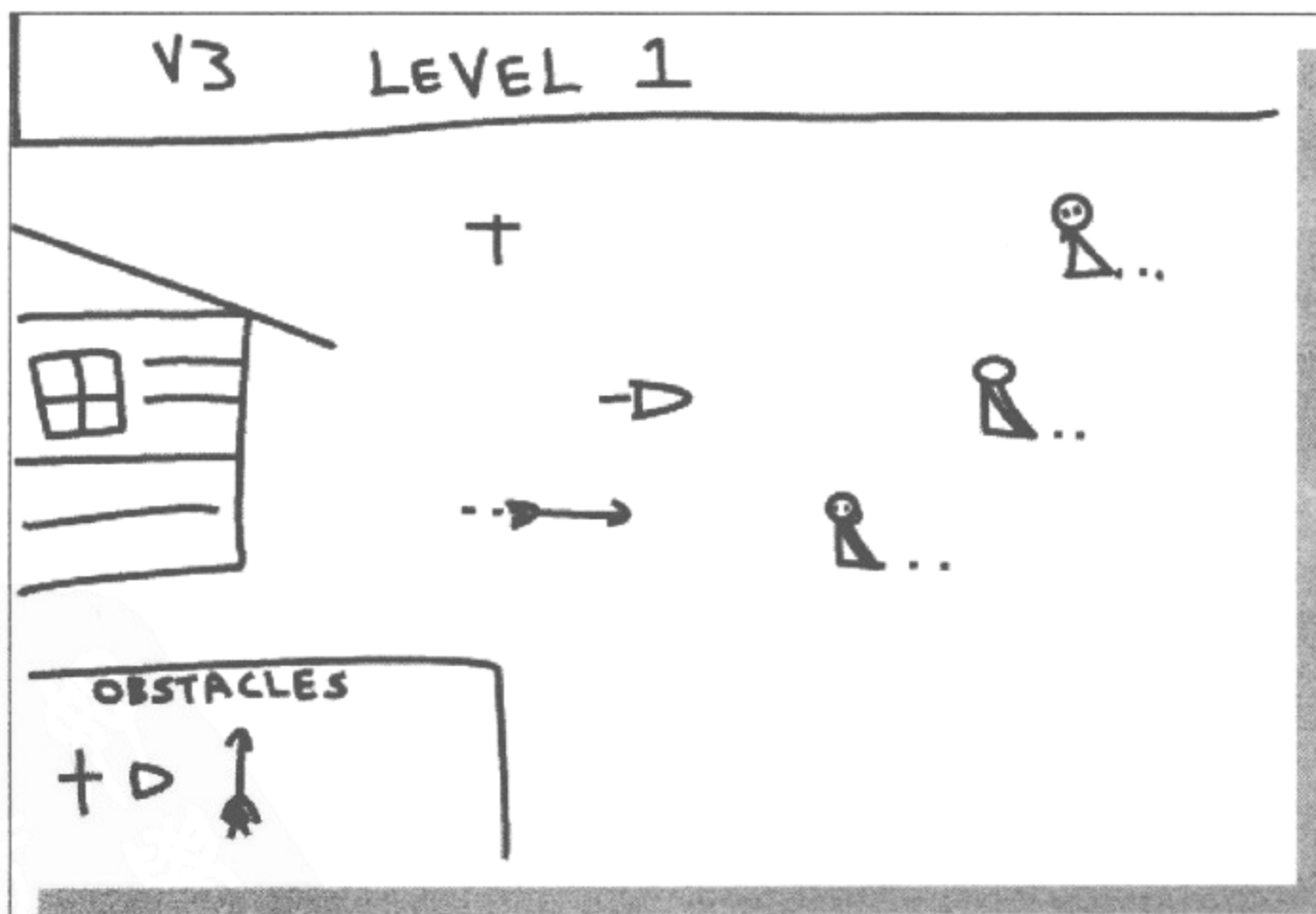


图 1.3 故事板上第 1 关的索引卡

### 1.3 AndEngine 范例

接下来, 本书将会用到 AndEngine 游戏平台, 现在来看看这个引擎都有哪些功能。Android Market 上面好多游戏都是用 AndEngine 做的, 不过不用再另外下载一个游戏了, 我们直接下载演示 AndEngine 功能的范例程序就好。

范例程序是由 AndEngine 的主开发者 Nicolas Gramlich 制作的, 可以免费下载。打开 Android Market 并从 Android device 中搜索 “AndEngine Example”, 然后就会看到如图 1.4 所示的下载界面。

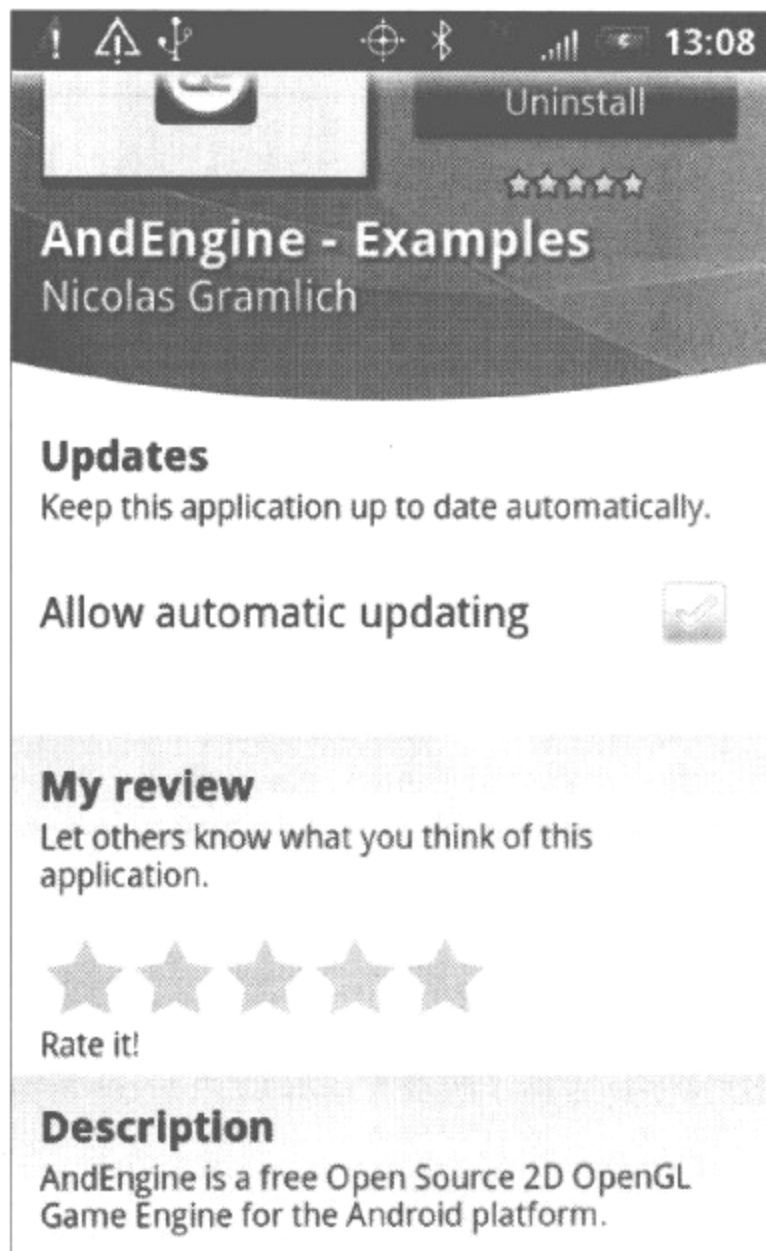


图 1.4 在 Android Market 中下载 AndEngine

Nicolas 慷慨地开放了 AndEngine Examples 的源代码 (在如下网址下载 <http://code.google.com/p/andenginexamples/>), 这是一份研究引擎功能的参考资料。如果读者愿意 (或者由于某些原因不能访问 Android Market) 的话, 可以在上述网站下载 .apk 安装文

件并通过 adb (Android Debug Bridge) 载入 Android 设备 (或模拟器)。第 12 章将详述如何构建源代码, 在这里, 只需要将范例程序安装到手机并运行就好。启动之后, 会看到如图 1.5 所示的功能菜单。



图 1.5 AndEngine 初始画面

菜单项分为不同的组, 每一组都演示了 AndEngine 开发平台的一个功能层面。请花些时间来运行各种范例程序, 看看 AndEngine 能够提供哪些游戏开发所用到的功能。

## 1.4 总结

本章通篇都是介绍性文字, 包括了游戏的基本知识以及手机游戏的基础概念。

- 讨论了各种类型的游戏, 并概括了迄今为止创造出来的一些新型游戏。当然, 读者也可以自创游戏类型, 但是在构思游戏创意时, 已有的这么多类型已经完全够用了。
- 分析了一款成功的 (有趣的) 手机游戏所具备的特点。在阅读本书剩下的章节时, 始终要牢记一点: 手机游戏的意义在于带给玩家乐趣。在接下来制作游戏的过程中, 将充分吸取从前的游戏开发者总结出来的经验与教训。
- 展示了一款塔防游戏, 以其作为范例来说明本书所介绍的工具及技术的用法。就

其本身来说，游戏概念是很简单的，但是它涵盖了手机游戏的典型元素。

- 定义了若干基本的游戏术语，以便讨论游戏的典型组件。
- 为开发 Android 游戏打下基础。现有的术语已经可以讨论不同的游戏组件了，接下来继续研究创建这些组件所需的工具以及实现它们所需的技术。

## 1.5 习题

1. 描述一下你想构建的游戏。不要过分专注于每一个小细节（实现并测试游戏时，会发现新的细节问题），但要写出游戏的主要元素。读者可以假定自己在为某游戏发行公司书写一份关于制作新游戏的提案书。

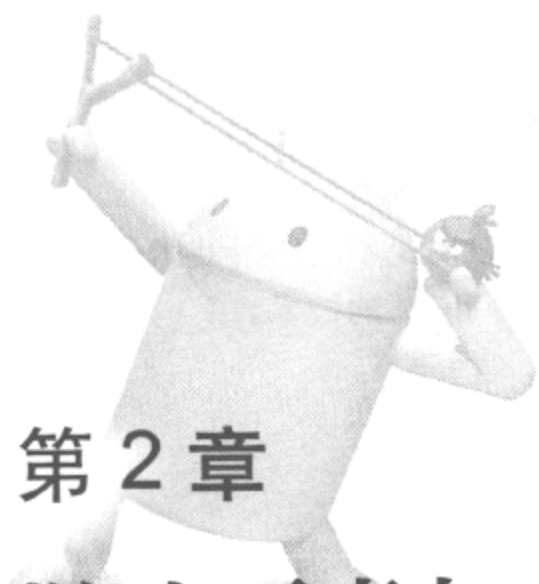
2. 邀请朋友们来评价这份游戏提案。看看他们是否认为该游戏很好玩，是否有关于改进游戏的建议或修改。对于该提案的建议可能会五花八门，这取决于大家的情绪甚至是讨论时所喝的饮料。有些建议很可行，有些则太过“创造性”了。讨论完后，总结一下共有多少条建议可以纳入游戏提案。

3. 画出该游戏的故事板。故事板没有标准格式，读者可以用自己习惯的风格。试着包含下列元素：

- 游戏所需场景。
- 场景间切换方式。
- 每个场景的特征。
- 每个场景粗略的图形布局。

4. 开列一份所需游戏配图的清单。有些游戏的图片不需要精心制作，用简单的几何图形就足够。另外一些游戏则需要全体美工一起来创建复杂的虚拟世界。





## 第 2 章

# 游戏要素与工具

- 2.1 软件开发工具
- 2.2 图形工具
- 2.3 声音工具
- 2.4 初试身手：制作启动画面
- 2.5 总结
- 2.6 习题

综观上一章所述，制作手机游戏显然需要编写相当复杂的软件，而且需要创建其他游戏组件，例如图形、动画、音效及音乐。要想把游戏提供给大家玩，就必须取得这些知识产权相关物件的商业授权。最简单的办法还是自己来创建这些东西。

还好，这些组件都有现成的制作工具，而且很多都是免费的。只要有一台能上网的工作电脑，所需的全部工具都能找得到。

本书的范例游戏《少女大战吸血鬼》是用 Java 语言写的，运行在 Android 系统的 Dalvik 虚拟机上。本游戏所利用的开源游戏引擎 AndEngine 及物理引擎 Box2D，都有 Android 平台的移植版本。本游戏及其所有知识产权相关物的创建都是由免费工具完成的，包括软件开发工具、图形处理工具、声音制作工具等。上述工具本章都会讲到，并用它们制作启动画面，以此进入真正的游戏制作过程。

Android 游戏只是普通的 Android 应用程序，这一点很重要。游戏是用 Dalvik/Java 语言编写的，可以访问所有的应用程序编程接口（Application Programming Interface, API），它们的活动类<sup>①</sup>具有 Android 应用程序活动类的所有特征（暂停、继续等）。游戏会引用 AndEngine 库，而且在 .apk 文件中也会包含一份该库。

## 2.1 软件开发工具

编写软件要有开发工具，幸运的是，无论是开发一般手机软件还是手机游戏，都有优秀的开发工具可用。更幸运的是，即使是以赢利为目的制作游戏，依然可以免费地下载并使用这些工具。

### 2.1.1 Android SDK

如果读者不熟悉 Android SDK，那就先把书放下，花些时间熟悉一下 SDK，然后再往下读。首先在这个网站（<http://developer.android.com>）阅读下载与安装的方法。

Android SDK 需要用到一个名为 Eclipse 的集成开发环境（Integrated Development Environment, IDE），还需用到 Oracle 公司 Java 开发包（JDK）中的相关工具。Android 网站上的安装说明将会引领读者完成这两个软件的安装（如果需要安装它们的话）。在本书写作之时，SDK 是以组件的形式组织起来的。本书的范例是用如下软件配置构建的。

□ Android SDK。

□ Android SDK Platform Components through 4.0。

① 原文为 activities，activity 中文为“活动”。Android 的开发包中有“android.app.Activity”类，此类定义了 Android 应用程序的基本生命期循环（创建、销毁、暂停、继续等），一般来说，开发者要通过继承该类来编写自己的应用程序入口。——译者注

- ☐ Android SDK Tools, r14。
- ☐ ADT Plugins for Eclipse 14.0.0。
- ☐ Eclipse Helios (该版本 Android SDK 所推荐的 Eclipse 版本)。
- ☐ Oracle/Sun Java Development Kit (JDK 6, 也叫 JDK 1.6)。

读者也可以用更新的版本构建, 本书的范例不太依赖于软件版本。如果有问题, 请访问网站: <https://github.com/portmobile/LAGP-Examples-Code>, 查看是否有针对新版软件的代码更新。本书写作时, 最新的 Android 版本是 4.0, 又名雪糕三明治 (Ice Cream Sandwich)。

读者还需要用 Android SDK 所附带的 AVD Manager 为需要支持的手机创建 Android 虚拟设备 (Android Virtual Device, AVD)。为了运行本书范例, 我们创建一个类似 HTC EVO 智能手机的 AVD。

- ☐ 名称: EVO。
- ☐ 目标平台: Android 2.2 (API level 8)。
- ☐ 皮肤: HVGA。
- ☐ SD 卡容量: 128MB。
- ☐ SdCard: 有。
- ☐ 重力加速仪: 有。
- ☐ LCD 密度值: 160。
- ☐ 音频输出: 有。
- ☐ 摄像头: 无 (该版本模拟器不支持, 范例游戏也不需要它)。
- ☐ 电池: 有。

请读者浏览 SDK 附带的教程, 并熟悉如下操作: 创建 Android 项目、编辑代码、构建项目、在模拟器上运行程序、用 Eclipse 调试项目、使用 LogCat 控制台以及其他 SDK 附带工具。如果要在 Android Market 上发行游戏, 那么要求其能在真机上流畅运行, 这需要读者掌握如何将 .apk 文件上传并安装到 Android 手机。

Android 开发者网站上面的文档写得非常详尽, 如果读者还需要一份更容易上手的入门教程与更多的开发范例, 那么可以进一步阅读相关的 Android 开发书籍, 例如 Lauren Darcey 与 Shane Conder 所著的《Sam's Teach Yourself Android Application Development in 24 Hours》。

### 2.1.2 AndEngine 游戏引擎库

AndEngine 游戏引擎库使得为 Android 设备开发二维游戏非常容易。Nicolas Gramlich 引领了 AndEngine 的开发, 并为其编写了许多代码, 此项目是开源的, 鼓励大家访问其开发站点并参与项目的开发。

我们当然可以使用 Java 编写例程，并利用 Android 的 API 把游戏写出来，但是，利用已经写好的游戏引擎去开发，有如下几个原因：

- 可以利用他人的成果。举个极端的例子，如果真的需要的话，我们可以自己编写进行 Android 编程所使用的 IDE，甚至自己写 Java 编译器。但是除非有特殊的功能需求，否则这么做是没有意义的。
- 使用类似 AndEngine 这样的开源引擎，可以根据自己的需要，随意扩展引擎功能。如果大家都需要这个扩展功能的话，我们可以将它提交至开源项目代码库，这样每个人都能用到改版之后的引擎了。
- 如果在开发中遇到了问题，可以询问开发者社区的同行，有可能别人也遇到了这个问题，并且知道如何解决或绕过它。
- 可以利用其他开发者优化过的代码。运行游戏需要相当多的资源，例如绘图、显示动画、计算物理效果、播放声音、响应用户输入等。通过使用游戏引擎，我们可以直接使用已经优化过的程序。

当前还有许多 Android 游戏引擎在开发中，然而本书将只专注于使用 AndEngine。

有关 AndEngine 的重要网址如下。

- AndEngine 源代码库：<http://code.google.com/p/andengine/>。
- 范例程序源代码库：<http://code.google.com/p/andengineexamples/>。
- AndEngine 社区论坛：<http://www.andengine.org/forums/>。
- AndEngine Wiki 页面：<http://wiki.andengine.org/AndEngine>。

当读者阅读本书时，以上网址可能会变更。如果有浏览器的话，请搜索“AndEngine Android”，这样就可以找到最新的页面网址了。

AndEngine 以一个 .jar 文件的形式存在。jar 文件是 Java 压缩包格式。本章稍后将会介绍如何用它搭配 Android SDK 项目来开始游戏的开发。引擎是在 GNU Lesser GPL 协议下发布的，该协议允许以任何合理的理由使用其源代码与二进制文件。（注意：我不是律师，读者或其律师可以在源代码网站上读到其引用的协议原文。）

### 2.1.3 AndEngine 游戏概念

先前所举的那个电影的例子，对于理解 AndEngine 很有帮助。游戏就像一部电影，其包含的概念都能在电影中找到对应物。

#### 摄像机

游戏的摄像机决定了玩家玩游戏时的视角，它与二维空间的摄影机非常相似。摄像机可以摇动或者缩放，以改变目前呈现的画面。镜头的摇动或者缩放可以受控于玩家的操作，也可用编程的方法实现。

## 场景

游戏像电影一样，也是由在其中发生一系列事件的场景组成的。在电影制作中，场景是按照固定的顺序组织起来的，然而在游戏中，场景则是根据游戏的发展而变换的。游戏有点像拍摄完成后即时播放的电影。

## 图层

场景是由图层组成的。图层是逐个叠加的，类似于旧时制作卡通动画时所用的赛璐珞。图层也用来表现仿 3D 效果<sup>①</sup>，当摄像机摇拍时，近处的图层移动相对快一些，远处的相对慢一些。

## 精灵

精灵在游戏中是人或者物体的视觉表现形式，类似于电影中的演员。精灵可以是运动的或静止的，但通常来说，精灵会在游戏过程中跟随场景移动。

## 实体

在 AndEngine 中，屏幕上所绘制的所有东西都是实体。精灵、瓦片、几何图形、线条，这些都是实体。所有的实体都具有属性，诸如颜色、旋转角度、缩放比例、位置等，这些属性的值都可通过修改器来改变。

## 修改器

AndEngine 的修改器功能很强大，可以改变实体的属性，而且改变的效果可以当场生效，也可以在某一个时间段内逐渐生效。本书的范例游戏会频繁地利用修改器来制作精灵与其他实体的特效。

## 贴图

一般来说，贴图是应用在物体上的 2D 位图材质，它决定了实体的外观，而且 OpenGL 图形环境的很多功能都和贴图有关。

## 纹理区域

贴图定义一张完整的位图，而纹理区域则是其中的一个小区域。稍后将会讲述 2D 图形的性能优化，其关键是将许多小块图片拼合成一张大图来使用。

## 引擎

引擎是用来展示场景的，它负责更新动画与修改器的图像，计算实际的绘图坐标，处理用户输入事件（触摸、按键、感应器），并控制整个游戏的执行。引擎与电影中的

<sup>①</sup> 原文为 2 ½D，又叫伪 3D。——译者注



制片人/导演非常像，告诉每个人应该做什么事情。

### BaseGameActivity 类

该类继承了 Android 的 Activity 类，是游戏中每个场景的基础。BaseGameActivity 类执行所有场景的共有逻辑：设置游戏引擎，适配 Activity 类的生命期需求规范，激活感应器。这个类将在第 3 章中详细讲解。

### 物理连接器

AndEngine 具备了基本的物理运算能力，然而 Box2D 物理引擎有更加丰富的功能。通过物理连接器可以将 AndEngine 的对象同 Box2D 相连接。如果游戏不含 Box2D 物理运算，则不需要使用它。

## 2.1.4 Box2D 物理引擎

AndEngine 含有 Box2D 物理引擎的 Android 平台移植版：JBox2D（是开源的），它可以非常真实地以多种方式模拟物体间的物理作用，尤其是：

- ☐ 刚体的物理作用。
- ☐ 稳定堆积。
- ☐ 重力。
- ☐ 用户自定义的物体。
- ☐ 快速的碰撞检测 / 接触检测。
- ☐ 滑动摩擦力。
- ☐ 矩形、圆形及多边形。
- ☐ 若干关节<sup>①</sup>类型：距离、旋转、棱柱、滑轮、齿轮、鼠标。
- ☐ 休眠（将静止物体从效果模拟运算中移除，直至它们被触碰为止）。

## 2.2 图形工具

任何视频游戏都需要生成相当多的图形，需要绘制背景、渲染精灵以及由精灵产生的动画。有很多电脑绘图软件，其中有一些专业软件相当复杂。如果你是一位会使用 Adobe Illustrator 或其他专业工具的美术设计人员<sup>②</sup>，那么可以跳过本段了。

相反地，如果读者和我一样，没有多少美术设计的经验与天赋，也没有太多的资金投入游戏开发，那么本节将会介绍用来制作 V3 游戏图片所用的工具（许多是免费的）。这些工具都非常优秀，而且被专业人士广泛使用。这些工具都有非常丰富的在线支

① 在 Box2D 中，关节指连接两个物体以限制其运动的限定物。——译者注

② 俗称美工，下同。——译者注



持。如果读者确定要使用这些“免费”工具的话，请为制作与维护它们的项目组做些力所能及的贡献。如果你能为项目做些编码、测试、Bug 修复、技术支援、文档撰写的事情，当然是好的；如果不能的话，请考虑提供一些资金支持，以便使项目能够继续运转。

读者可能急于从互联网上的大量图片中下载一些直接使用，这样做一定要确保已从图像拥有者那里获得了使用许可。可以下载某张图片并不意味着可以使用它。如果游戏是计划要收费的话，使用他人的图像进行“商业”用途，将是非常棘手的事情。

读者大概知道，图像处理软件可分为两大类：绘制与操作矢量图的“绘图程序”和在桌布上绘制彩色位图的“画图程序”，它们各有其用，且都将在 V3 中派上用场。

### 2.2.1 矢量图工具：Inkscape

矢量图的绘制非常方便，主要有两个原因：

- 在绘制成品图像时，其中每一个绘图组件都被看做一个对象，可以独立地移动、缩放、旋转以及编辑。
- 图形组件与整个图像都可以随意缩放，并且效果不会失真。这一点对于精灵的绘制尤其重要，因为它们的成品图通常很小。矢量图的缩放并非完全没有缺陷，因此绘图时应尽量根据所绘物体的最终尺寸来画，必要时再行缩放。

Inkscape (<http://www.inkscape.org>) 是个非常流行的矢量图绘制工具，许多 Linux 发行版都附带它，并且可在各版本的 Mac OS X 与 Windows 操作系统上运行。下载包中含有许多帮助文件与教程。如果读者已经熟知矢量图的绘制，那么会很快熟悉它的操作。如果你和我差不多，没有绘图的经验与天赋，那么可能会多花一些时间来熟悉它，但这比学 Adobe Illustrator 这样的软件要快得多。刚才我有没有提到这个工具是免费的？图 2.1 演示正在用 Inkscape 绘制一只蝙蝠。

基本的 AndEngine 游戏引擎包并不能渲染矢量图，该功能可通过扩展包来实现。笔者在制作 V3 时的做法是，先用 Inkscape 画出满意的图来，然后将其缩放至合适尺寸，再保存为矢量图格式 (.svg)，并导出为 .png (Portable Network Graphics) 格式的文件，最后如果有必要的话，用下段将会讲到的 GIMP 工具制作一个透明的背景。这样所产生的位图文件（依然是 .png 格式）就是 AndEngine 所需的图片文件格式。

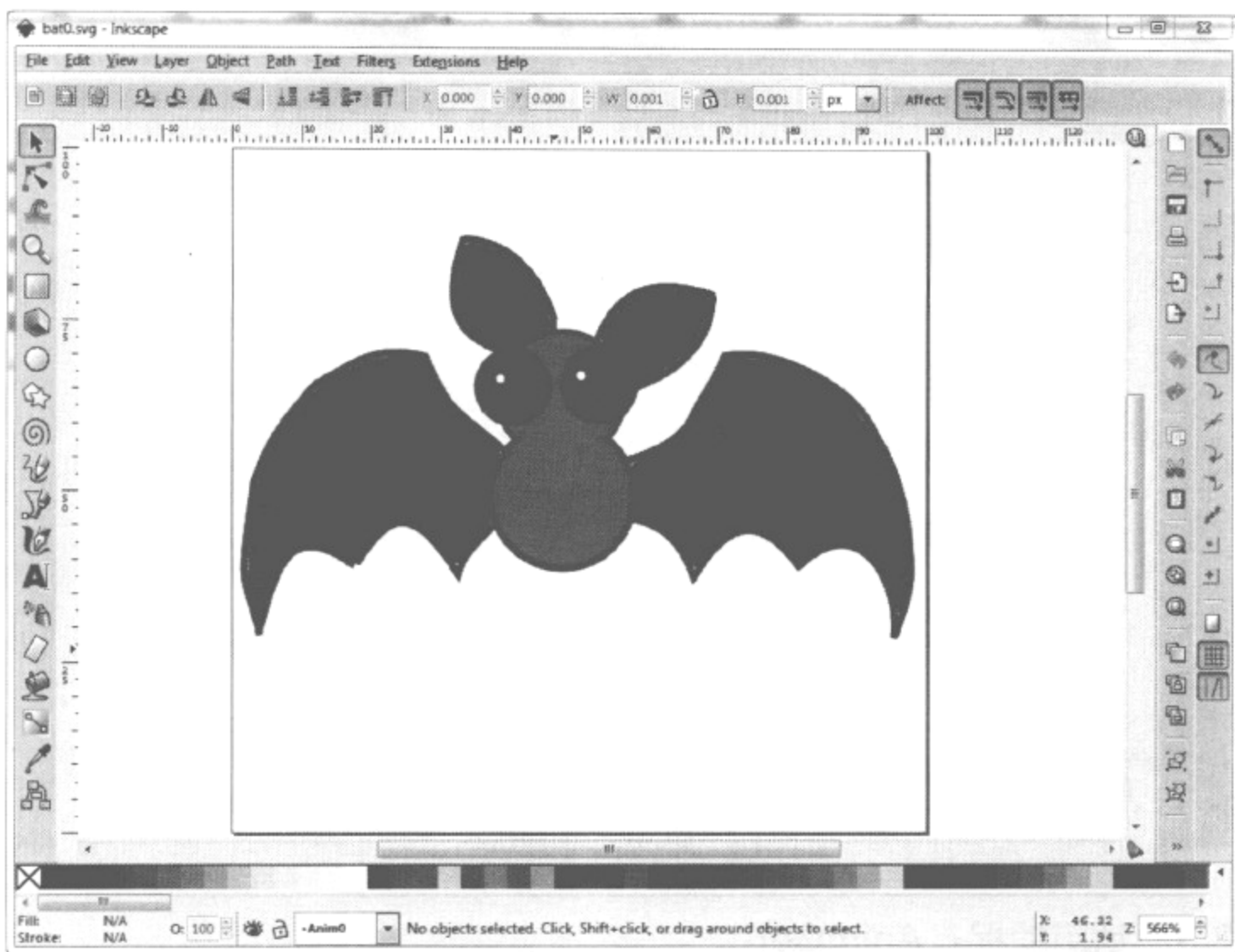


图 2.1 Inkscape 矢量图绘制编辑器

### 2.2.2 位图工具：GIMP

GIMP (GNU Image Manipulation Program ; <http://www.gimp.org>) 是世界范围内多数人使用的知名的老牌跨平台位图绘制程序。众多 Linux 发行版都含有这款免费软件。GIMP 开发组本身不支持 Windows 和 Mac OS X 操作系统，但是可以下载到用于这些操作系统的安装文件。

图 2.2 所示蝙蝠与图 2.1 相同，但它是作为位图渲染的，此位图是用 Inkscape 软件将矢量图另存为 .png 格式而得到的。正如读者所见，这么做得到的图像略显颗粒状，再加上透明背景，正好可以当做所需精灵动画的一帧。当精灵显示出来的时候，背景将会透过透明的格子区域而显示出来。

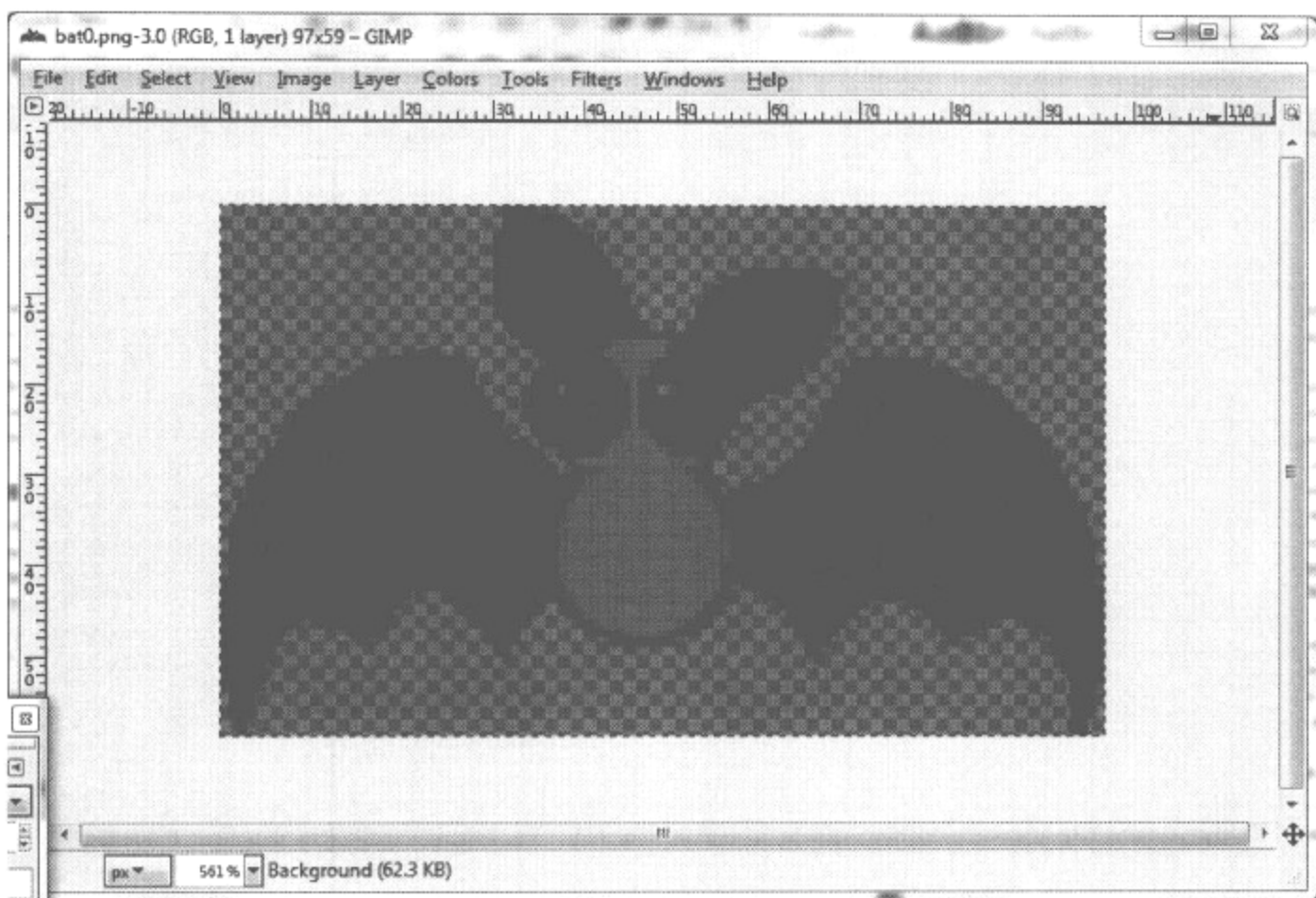


图 2.2 GIMP 位图编辑器

### 2.2.3 动画捕捉：AnimGet

制作动画是既乏味又耗时的事情，游戏所需各个角度、各个动作的动画都需要一帧一帧来做。

有个简单的办法，就是用工具制作出 3D 动画（Blender 是个广泛运用的开源工具，商业工具有 3D Max、Poser、Maya 等），然后按照每个所需的角度来渲染 2D 动画，这样将得到一份 AVI 文件或 GIF 动画文件，最后将其分解成单个的帧，以便组装成精灵图。Michael Menne 制作了一个名为 AnimGet 的工具专门用于完成分解工作，该工具很多地方都能下载到（用 Google 搜索“AnimGet”即可），它只能在 Windows 操作系统下运行。

AnimGet 的工作原理非常巧妙，它通过监控屏幕上一块特定区域的变化情况来捕捉动画，而不是用一些诸如解析动画文件这样的复杂方式去做。软件先收集该区域的初始像素，然后每隔 10 毫秒收集一次，如果侦测到有像素改变了，那么 AnimGet 抓取该区域的一个新副本，并把它存放到新的文件里。这些快照都是在内存中生成的，所以速度很快，然而用户要小心地选择捕捉区域，让其只包含需要捕捉的动画即可。当用户命令 AnimGet 停止捕捉时，所抓取的图像会被写成一个独立的文件。

### 2.2.4 瓦片地图创建工具：Tiled

在电脑游戏中，瓦片用来绘制一组规则的矩形图像，例如一小块地图。在 V3 游戏中，有吸血鬼入侵的游戏区域以及其上的障碍物，都是通过瓦片拼成的。Tiled 地图编辑器（可在 <http://www.mapeditor.org/> 下载）是一款免费软件，用 Qt 跨平台 UI 开发库编写，因此可运行在 Windows、Linux 及 Mac OS X 操作系统上。本书第 9 章的相关章节专门会讲述瓦片地图的用法，在图 2.3 中，我们先看一下正在编辑中的瓦片地图是什么样子。

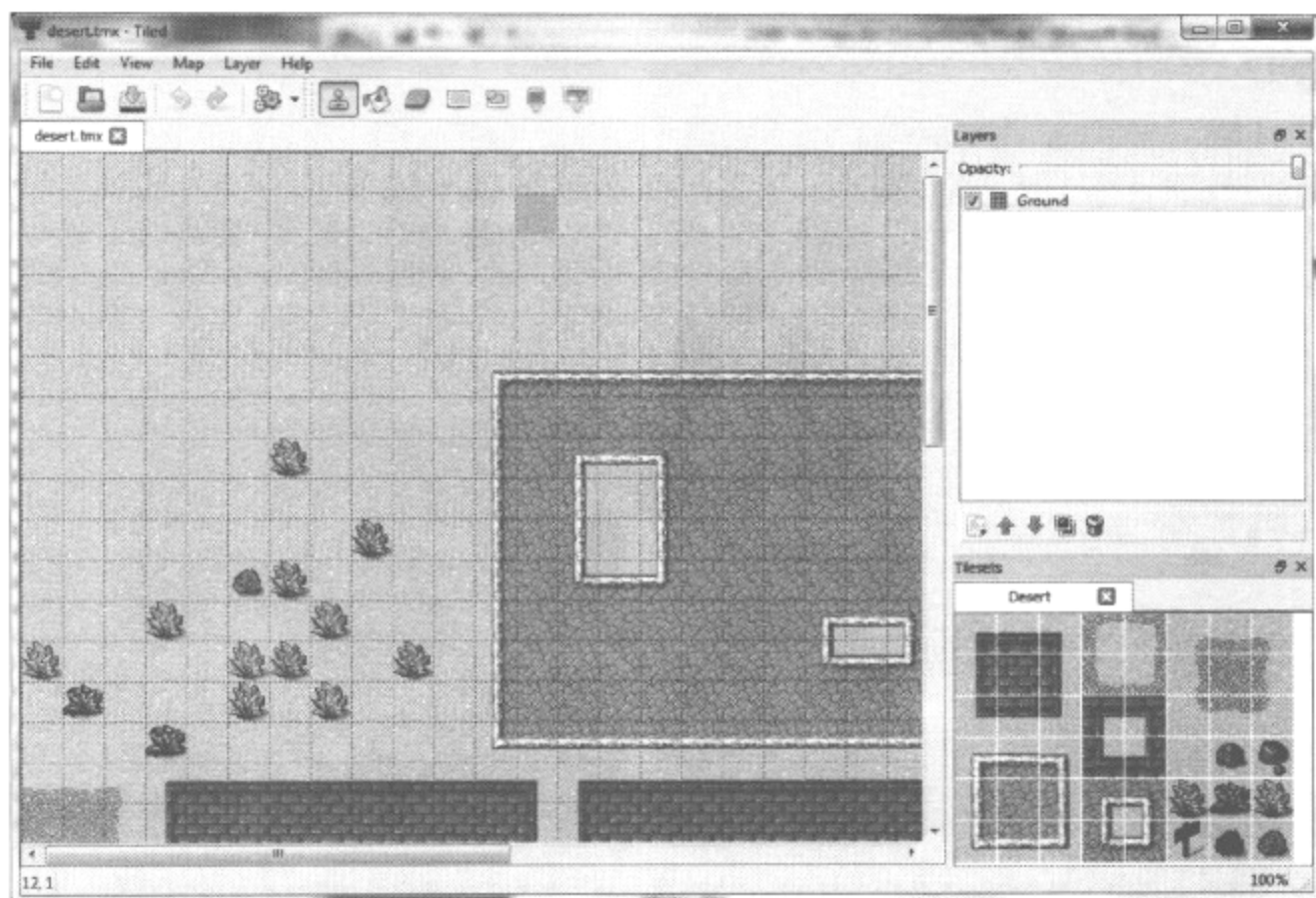


图 2.3 Tiled 瓦片地图编辑器

### 2.2.5 TrueType 字体创建与编辑工具：FontStruct

Android 系统支持使用 TrueType 字体，所以我们的游戏也将用它。可以找到许多免费或收费的字体，而且创建与编辑 TrueType 字体的工具也有很多，尤其是由 FontShop 所出品的一款基于 Web 的工具：FontStruct。用户可以透过开源社区分享所创建的字体，这些字体都是可供下载的，每款字体都附有许可协议（通常是 Creative Commons 协议<sup>①</sup>）。

① 中文名称为“创作共用协议”。创用 CC（Creative Commons，CC）是一个非赢利组织，也是该组织所提供一系列弹性著作权授权方式的名称。创用 CC 组织的主要宗旨是使得著作物能更为流通与改做，作为其他人据以创作及共享的基础，并以所提供的授权方式确保上述理念。——译者注

自己创建字体需要花很大工夫，不是很推荐这么做，除非找不到游戏所需的现成字体，或者读者非常喜欢自己制作字体。图 2.4 是 FontStruct 网站的 FontStructor 窗口页面，用户可在这里创建与编辑字体。

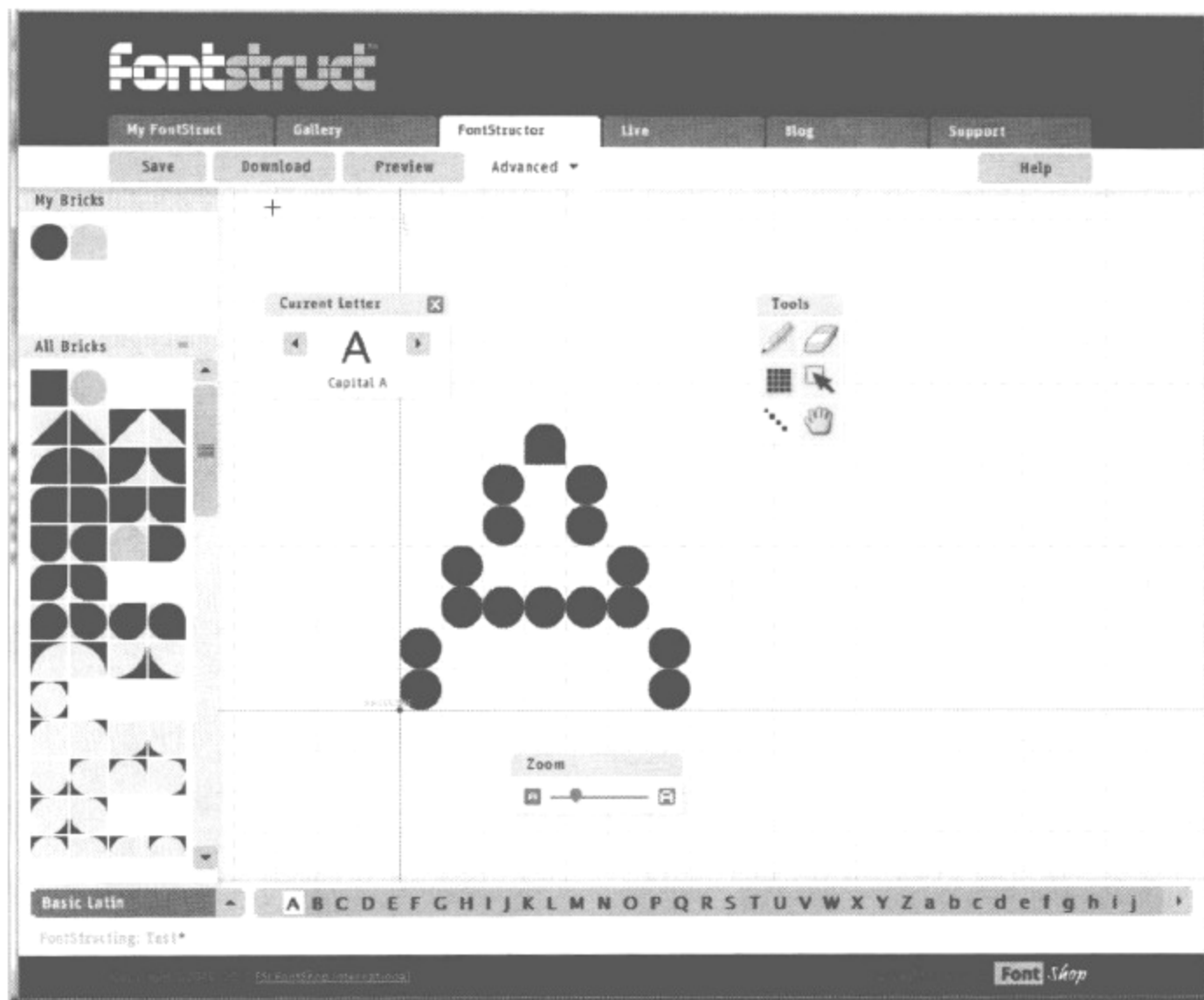


图 2.4 FontStruct TrueType 字体编辑器

## 2.3 声音工具

V3 游戏需要两类声音资源，每种类型都需要有创建与编辑的工具。音效是伴随游戏事件所发出的短暂声音（最多 5 秒，通常是 3 秒）。当玩家进入了某个场景并开始在其中进行游戏时，通常会播放一段稍长时间的背景声音。这种声音另存为音乐文件。

### 2.3.1 音效工具：Audacity

游戏没有音效能行吗？要创建出色的游戏音效，可以使用世界级的声音编辑工



具：Audacity (<http://audacity.sourceforge.net/>)，它是一款可运行在 Windows、Mac OS X 与 Linux 平台之上的免费软件。该软件可以导入或捕捉声音文件，加以编辑，并输出为多种声音格式。Android 系统可以很好地支持 AAC、MP3、MIDI、Ogg Vorbis、WAV 等文件格式，这些格式 Audacity 都能处理。图 2.5 是正在用 Audacity 编辑音效时的界面。

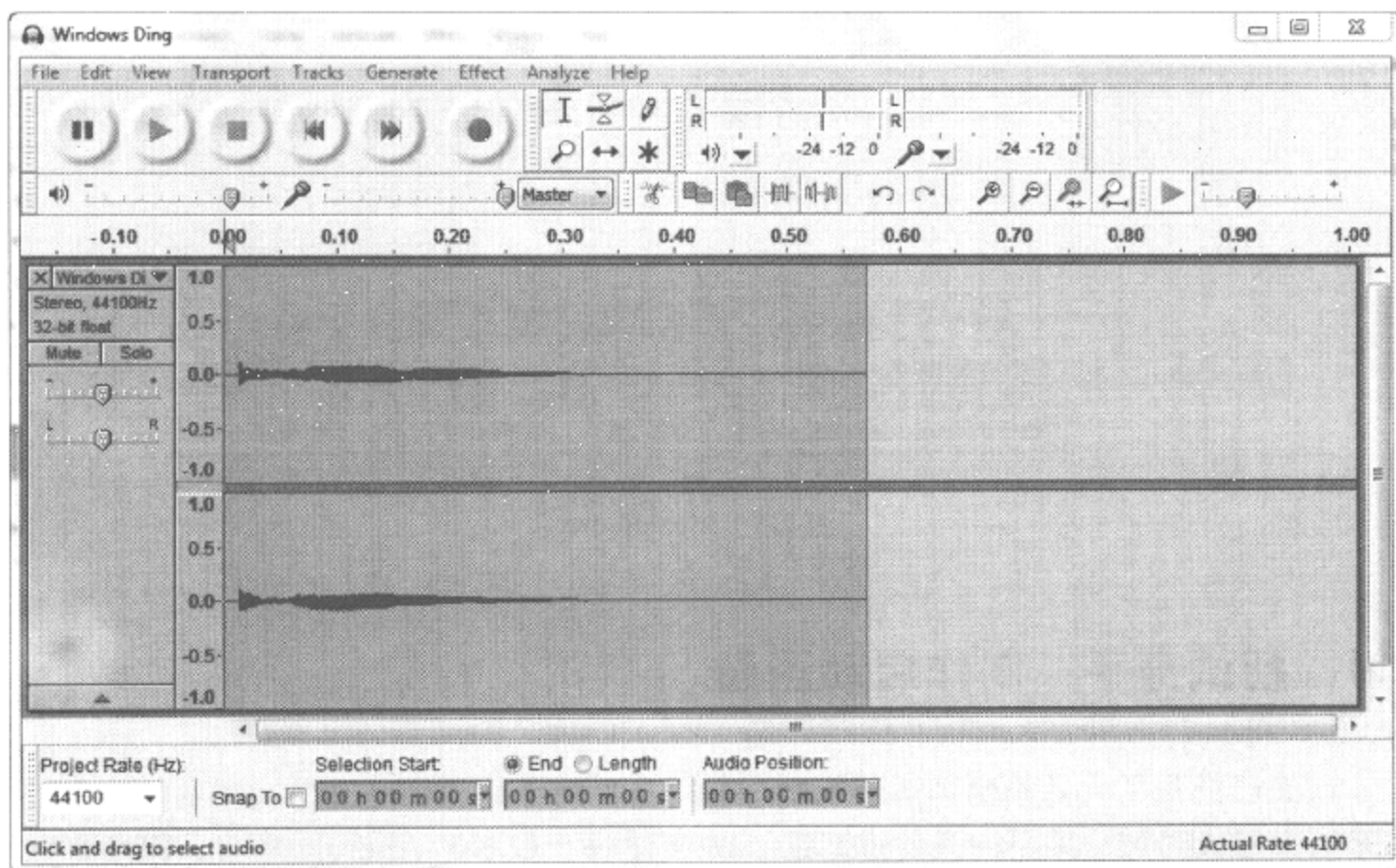


图 2.5 Audacity 声音编辑器

### 2.3.2 背景音乐工具：MuseScore

游戏也要有背景音乐，MP3、OGG、MIDI 格式均可，制作这些文件的方式有很多种。比方说，如果你有音乐天赋，可以用乐器演奏一段并录下来，或者也可以在网上查找可用作商业用途的免费音乐文件。

由于笔者缺乏音乐天赋，因此选择用一款名为 MuseScore (<http://www.musescore.org>) 的开源工具来制作 V3 游戏的音乐，该软件可以用电脑键盘或外接的 MIDI 键盘来制作或编辑音乐。图 2.6 显示了 MuseScore 的主界面，用户正在该界面中编辑一段音乐。

在演奏完你的作品之后，可以用 MuseScore 将其渲染并另存为 MIDI、MP3、OGG 或 WAV 文件，这些文件 Android 系统都可以播放。



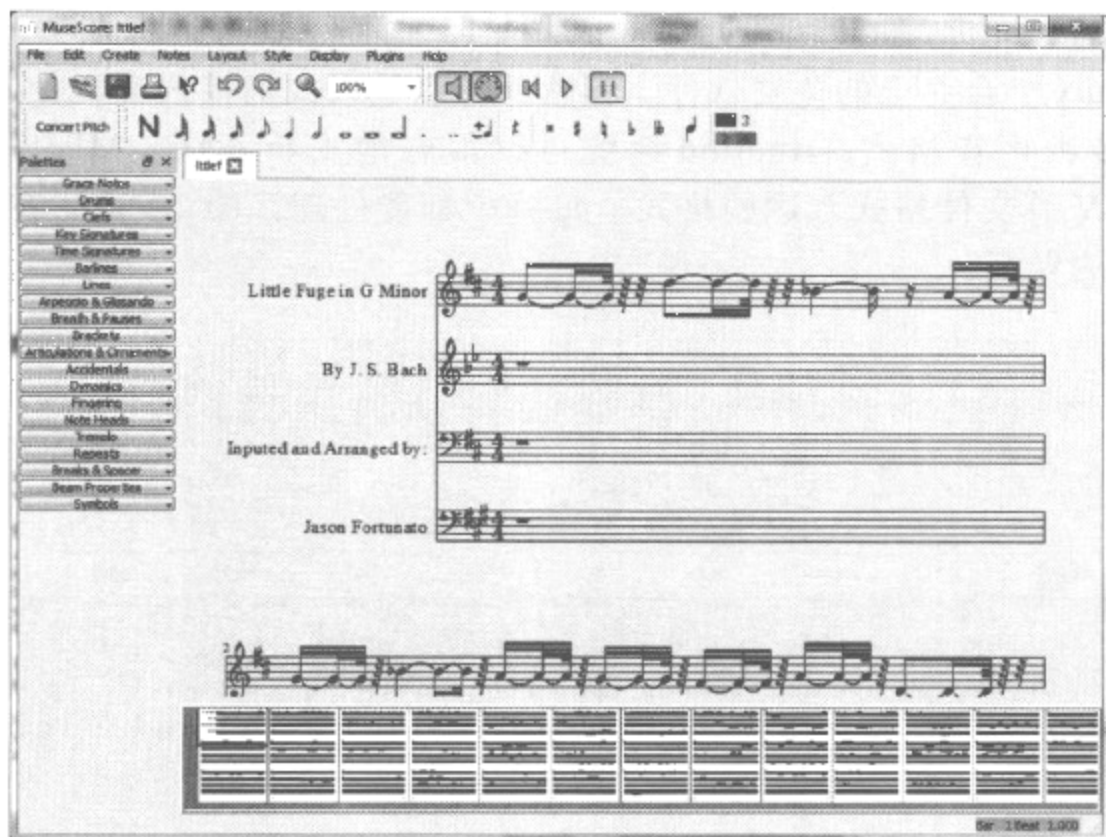


图 2.6 MuseScore

## 2.4 初试身手：制作启动画面

前面已经花了大量篇幅讲述游戏概念与工具，为了让我们进入游戏开发的状态，现在开始来写代码吧！采用的开发模型是：先创建一个基本的 Android 项目，以后每章都为其增添内容，直到游戏完成。

一开始我们先来写启动画面的程序。如图 2.7 所示，启动画面将持续 5 秒钟，然后切换到黑屏。

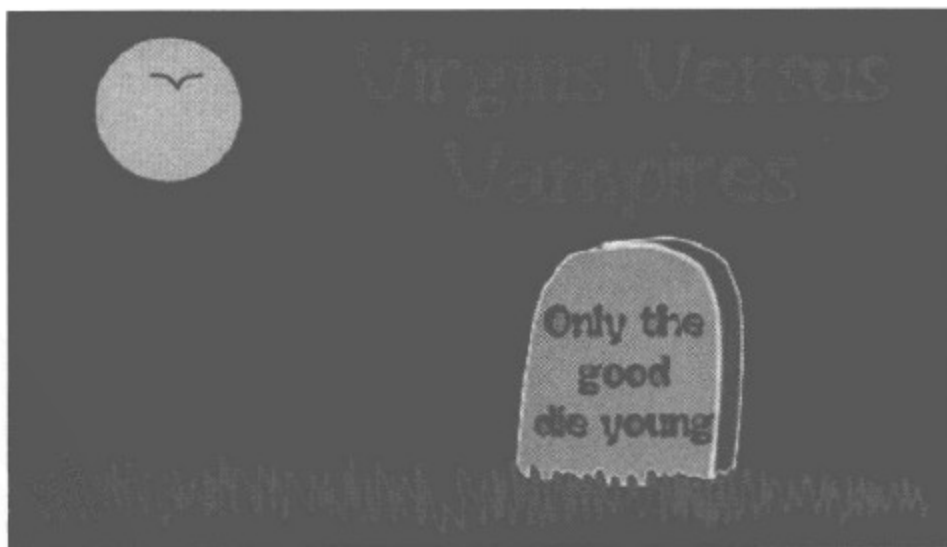


图 2.7 范例游戏《少女大战吸血鬼》(3V) 的启动画面

### 2.4.1 创建游戏项目

整本书的开发都将使用 Android SDK。为了开始用 SDK 开发 V3 游戏，我们先创建一个新的 Android 项目。选择“New”菜单，再选择其下的“Android Project”菜单项，在弹出的对话框中依次输入下列信息。

1. 项目名称：V3。
2. 勾选“在工作区中创建新的项目”（Create New Project in Workspace）这一选项。
3. 使用默认的工作区路径（或者读者想要设定的其他路径）。
4. 选择“Android 1.6”作为构建目标。
5. 应用程序名称：V3。
6. 包名：com.pearson.lagp.v3。
7. 勾选“创建 Activity”（Create Activity）选项并输入名称：StartActivity。
8. 最小的 SDK 版本：4。

执行完上述步骤之后，范例游戏的项目就创建好了，其中包含 StartActivity 类，该类是一个空的“Hello, World” Activity 类模板，本章稍后部分将会修改其代码。

### 2.4.2 加入 AndEngine 库

如果读者还未从网站上下载 AndEngine 库，现在就请下载它。在写作本书时，AndEngine 的 .jar 文件可以从位于 Google 的 AndEngineExamples 代码库中取得，网址如下：  
<http://code.google.com/p/andengineexamples/>

在源代码标签页下面，可以于 lib 文件夹下找到 .jar 文件，也可以将整个源代码复制到电脑上。推荐用后面一种方式，因为这样可以访问范例程序的源代码。AndEngine 使用 Mercurial 源代码管理系统，如果读者不熟悉它，可以阅读网站上的安装指导。在有了 .jar 文件之后，需要将其导入项目中。

- 在 Eclipse 的 Package Explorer 视图中，展开 V3 项目节点（如果还未展开的话）。右击项目，在弹出菜单中选择 New 菜单，继而选择 Folder 菜单项，在其后的对话框中输入新文件夹的名字 lib，并单击 Finish 按钮。
- 右击 lib 文件夹，在弹出菜单中选择“Import...”。
- 在其后弹出的对话框中选择“General”群组下的“File System”选项，用“Browse”按钮定位到 AndEngine 的 .jar 文件所在的目录（如 .../And Engine Examples/lib），选中该目录，并按确定按钮。
- Import 对话框的目录名旁边会显示一个未选中的复选框，右方的面板列出了当前目录下的文件，这其中就应该包括 andengine.jar。选中此文件名旁边的复选框，并单击 Finish 按钮。
- 若要让游戏项目包含其他构建路径下的 .jar 文件，则需在 Eclipse Package Explorer

视图中找到该文件，右击它，选择 Build Path 菜单，继而选择 Add to Build Path 菜单项，这样此文件就会直接显示在项目文件夹下。

### 2.4.3 加入启动画面代码

为了在载入游戏组件时显示启动画面，需要修改 StartActivity.java 的代码。在实际游戏中，在所有资源加载完成之后，将会显示游戏菜单。但在本章这里，启动画面将一直显示，直到用户返回系统的主桌面。程序清单 2.1 是修改后的 StartActivity.java 文件。

程序清单 2.1 StartActivity.java

```
package com.pearson.lagp.v3;
import org.anddev.andengine.engine.Engine;
import org.anddev.andengine.engine.camera.Camera;
import org.anddev.andengine.engine.options.EngineOptions;
import org.anddev.andengine.engine.options.EngineOptions.
    ScreenOrientation;
import org.anddev.andengine.engine.options.resolutionpolicy.
    RatioResolutionPolicy;
import org.anddev.andengine.entity.scene.Scene;
import org.anddev.andengine.entity.sprite.Sprite;
import org.anddev.andengine.entity.util.FPSLogger;
import org.anddev.andengine.opengl.texture.Texture;
import org.anddev.andengine.opengl.texture.TextureOptions;
import org.anddev.andengine.opengl.texture.region.TextureRegion;
import org.anddev.andengine.opengl.texture.region.TextureRegionFactory;
import org.anddev.andengine.ui.activity.BaseGameActivity;

public class StartActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;
    // =====
    // 字段
    // =====

    private Camera mCamera;
    private Texture mTexture;
    private TextureRegion mSplashTextureRegion;

    // =====
    // 覆写超类及接口中的方法
    // =====

    @Override
    public Engine onLoadEngine() {
```

```

        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT),
        this.mCamera));
}

@Override
public void onLoadResources() {
    this.mTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mSplashTextureRegion = TextureRegionFactory
        .createFromAsset(this.mTexture,
            this, "gfx/Splashscreen.png", 0, 0);
    this.mEngine.getTextureManager().loadTexture(this.mTexture);
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());
    final Scene scene = new Scene(1);
    /* 将摄像头置于启动画面中央。 */
    final int centerX =
        (CAMERA_WIDTH - this.mSplashTextureRegion.getWidth()) / 2;
    final int centerY =
        (CAMERA_HEIGHT -
            this.mSplashTextureRegion.getHeight()) / 2;
    /* 创建 Sprite 对象并将其加入 Scene 对象中。 */
    final Sprite splash = new Sprite(centerX,
        centerY, this.mSplashTextureRegion);
    scene.getLastChild().attachChild(splash);
    return scene;
}

@Override
public void onLoadComplete() {
}

}

```

看起来显示一个屏幕画面需要的代码不少，但实际上在 AndEngine 中，启动画面与其他精灵一样。之后我们需要做的是进行游戏的初始化。现在来分段看看代码。

□ 在写完 import 指令后，立即声明了一个继承自 BaseActivity 的类，该类完成了所有 AndEngine 的初始化工作，并提供了一些稍后将被覆写<sup>⊖</sup>的方法。

⊖ 原文为 Override，中文常称为“复写”、“覆盖”、“重写”。——译者注

- 接下来设置了摄像头视角的尺寸。因为这是个启动画面，所以需要占满整个屏幕。该画面所用的图像，`Splashscreen.png`，是  $480 \times 320$  像素的，正好是 Android 设备的默认屏幕大小。
- 其后为摄像头、贴图及贴图区域分别定义了字段。
- 代码覆写了三个由 `BaseGameActivity` 超类所定义的方法，每个都对应于载入过程中的时间点。
  - `onLoadEngine()` 方法会在游戏引擎被该 Activity 所加载时调用，这里初始化了摄像头与引擎对象。关于引擎的初始化，有两个问题要说明。首先，将屏幕方向设置为横屏（`LANDSCAPE`），大部分手机游戏都是横屏的。其次，代码需要一个名为 `RatioResolutionPolicy` 的 `ResolutionPolicy` 对象，引擎可以根据该策略在不同的设备屏幕大小下缩放图像，同时保持原有的纵横比。
  - `onLoadResources()` 方法会在需要加载资源时调用。这里需要创建 `Texture` 对象并用启动画面图片 `Splashscreen.png` 来填充它。这个文件必须放在项目的 `assets/gfx` 目录下，否则程序无法启动。现在不用担心 `TextureOption` 的设置，在第 5 章讲绘制与精灵时，会谈到它的。最后，将贴图对象载入引擎的 `TextureManager` 中。
  - `onLoadScene()` 方法会在引擎将要显示场景时调用。该方法的代码依次初始化了每秒帧数记录器，创建了场景，将摄像头放置于场景中央，创建了表示启动画面的精灵，将其挂接到场景上，最后将场景对象返回给引擎。实际上 `AndEngine` 为启动画面提供了一种特殊类型的场景，叫做 `SplashScene`，但这里所写的代码也能做出同样的效果。

写好 `StartActivity.java` 的代码之后，需要将启动画面所需图片导入 `assets/gfx` 目录，这只需要几步就好。

- 在 `Project Explorer` 视图中右击 `V3` 项目下的 `assets` 文件夹，选择 `New` 菜单下的 `Folder` 菜单项，创建名为 `gfx` 的子文件夹。
- 右击新建的 `gfx` 子文件夹，导入 `Splashscreen.png` 文件。将其放在 `assets` 文件夹下才能使 `createFromAsset()` 方法能够将其载入 `Texture` 对象中。

#### 2.4.4 用模拟器运行游戏

用 Android 模拟器运行游戏非常简单，只需在 Eclipse 中执行 `Run` 菜单下的 `Run` 菜单项即可。如果提示需要选择 AVD，那么就选择一个。现在应该能在模拟器的窗口中看见启动画面了。如果需要将模拟器旋转至水平方向，按 `Ctrl+F12` 即可。如果提示应用程序已强制关闭，那说明缺少所需的资源。LogCat 控制台的输出会显示具体是哪个资源没找到。

### 2.4.5 用 Android 设备运行游戏

读者请访问自己手机制造商的开发者网站，查找如何正确地将手机连接到电脑以进行 Android 开发。要保证手机 Android 版本不低于创建项目时所选择的目标平台版本（在范例游戏中，目标平台是 API Level 4，对应于 Android 1.6 版本）。

现在再选择 Eclipse Run 菜单下的 Run 菜单项，应该有在真机上运行程序的选项了。Android SDK 会将程序安装在手机上并开始执行。读者可以按下 Back 或 Home 键返回主桌面，但是与所有的 Android 程序一样，游戏会在后台继续运行。实际上程序在后台并没有做什么，因为显示画面是不可见的，而且 Android 系统很智能，不会浪费 CPU 资源去更新不可见的显示画面。尽管如此，后台程序仍然会占用一定的 CPU 资源。如果要停止或者卸载程序，请在手机菜单上依次选择“设定”（Settings）、“应用程序”（Applications）、“管理应用程序”（Manage Applications），在接下来的对话框中即可进行操作。

## 2.5 总结

本章讲述了许多基础问题，快速浏览了 Android SDK，介绍了 Android 系统以及一些游戏组件：

- ☐ 摄像机
- ☐ 场景
- ☐ 图层
- ☐ 精灵
- ☐ 实体
- ☐ 修改器
- ☐ 贴图
- ☐ 贴图区域
- ☐ 引擎
- ☐ BaseGameActivity 类
- ☐ 物理连接器

其后讲述了制作手机游戏组件所需的工具，包括如下几种：

- ☐ Inkscape：矢量图制作工具
- ☐ GIMP：位图制作工具
- ☐ AnimGet：动画捕捉工具
- ☐ Tiled：瓦片地图编辑工具
- ☐ Audacity：声音编辑工具



### ❑ MuseScore: 音乐制作工具

本章最后开始了 V3 游戏的制作, 用 Android SDK 向导创建了 Android 项目, 加入了 AndEngine 库, 增加了进行引擎初始化的代码, 并显示出了游戏的启动画面。

## 2.6 习题

1. 使用 Inkscape、GIMP 或喜欢的图形制作工具为自己的游戏绘制启动画面。不管用矢量图还是位图做都行, 但要保证最终结果是一张 480 像素宽、320 像素高的矩形图片。替换 StartActivity.java 中的启动画面图片, 并运行游戏。

2. 根据代码库网站上的指导手册, 将 AndEngine 和 AndEngineExamples 库复制到开发电脑上。复制过程中需要名为 Mercurial 的源代码管理工具 (本身也是免费的)。这两部分代码大约需要占用 40MB 的磁盘空间。

3. 在第 1 章中, 读者为自己的游戏绘制了故事板, 现在请为其中的每个场景列出如下的元素列表:

- ❑ 图层
- ❑ 精灵, 注意也包括动画
- ❑ 贴图
- ❑ 声音

4. 为读者自己的游戏制作一首主题音乐, 可以先决定要付费请他人制作音乐还是自己来做。



## 第 3 章

# 游戏循环与菜单

- 3.1 游戏循环概述
- 3.2 AndEngine 的游戏循环
- 3.3 为 V3 增加菜单屏幕
- 3.4 内存使用
- 3.5 “退出”选项
- 3.6 总结
- 3.7 习题

游戏循环<sup>①</sup>通常被看做电脑游戏的活力之源，它确实赋予了游戏生命力。游戏运行时，诸如输入、状态、输出等，这些都需要在某个例行的逻辑周期中处理。游戏循环就是对这类周期进行的代码实现，所以流畅的游戏需要有效率的游戏循环。

### 3.1 游戏循环概述

电脑游戏都是基于一系列被反复执行的动作的，通常包含如下这些操作：

- 获取用户输入。
- 更新处理用户输入之后的游戏状态。
- 更新根据时间因素计算之后的游戏状态。
- 检测物体间的碰撞。
- 更新基于游戏物理学的物体状态。
- 根据更新之后的游戏状态，刷新用户界面的内容。
- 渲染刷新之后的用户界面。
- 播放适当的声音。

在上述步骤中，“游戏状态”不仅仅包括玩家当前所在的关卡与得分，也包括每个被显示对象的位置与动画播放情况。在拥有很多对象的游戏里面，设法快速地执行这个循环以至于不让用户感觉到明显的动画播放延迟，是一项艰巨的任务。

时间可能是游戏循环考虑的因素之一，也可能不是，这要根据游戏的性质来定。举例来说，有些棋类与解谜类游戏有时间元素，有些则没有，如果有的话，通常表现为解决谜题的耗时。过关后，将根据耗时的多少来评定玩家得分。在棋类游戏中，玩家通过用户界面（User Interface，UI）来走棋，其后电脑会计算出自己该回合的走法，再更新游戏状态来反映这步棋的移动，这样用户就能看到结果了。上述循环过程将继续执行，电脑只关注达成目标所花的总时间。

在类似 V3 这样的游戏中，玩家相对来说是静止的，而游戏中的动画和实体都在频繁地移动，所以时间因素更显得重要了。游戏必须定期根据时间来刷新用户界面，也要刷新实体与动画的位置。要在拥有不同 CPU 速度、运算能力与 UI 设备的众多硬件平台上都能平滑地执行上述刷新才行，而且排定的刷新操作不能被突发的垃圾回收操作所阻滞。要使游戏好玩，刷新必须迅速，以便一个循环周期内能刷新很多对象。

把这些问题都处理好是很难的，所幸 AndEngine 已经替我们完成了这项工作，开发者只需使用内嵌于 AndEngine 库中的游戏循环机制即可。

① 原文为 game loop，又叫“游戏主循环”、“游戏回圈”。——译者注

## 3.2 AndEngine 的游戏循环

AndEngine 包含一个叫做 Engine 的类，该类同 Android 运行环境一起来实现游戏循环。从启动游戏开始，到关闭游戏为止，Engine 类始终是 Activity 的核心。开发 AndEngine 游戏首先要做的就是覆写 onLoadEngine() 方法，如程序清单 3.1 所示，在覆写方法中创建 Engine 对象并为其设置可选参数。

程序清单 3.1 初始化 Engine 对象

```

@Override
public Engine onLoadEngine() {
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
}

```

Engine 对象为游戏开发者完成如下事情：

- ☐ 初始化并持续维护着 OpenGL 的 SurfaceView 对象，该对象用来往屏幕上绘制内容。
- ☐ 管理键盘与触摸屏的用户输入。
- ☐ 管理重力加速仪与定位仪等传感器输入。
- ☐ 管理文本字体库。
- ☐ 创建并管理音效与音乐的加载和播放。
- ☐ 管理手机的震动器。
- ☐ 定期更新 AndEngine 其余部分的逻辑，以推动游戏状态的改变。

### 3.2.1 初始化 Engine 对象

Engine 类的默认构造器<sup>①</sup>是：

```
public Engine(final EngineOptions pEngineOptions)
```

EngineOption 类具有如下构造器：

```

public EngineOptions(final boolean pFullscreen,
    final ScreenOrientation pScreenOrientation,
    final IResolutionPolicy pResolutionPolicy, final Camera pCamera)

```

① 原文为 Constructor，中文俗称“构造方法”、“构造函数”，又叫“构建子”。——译者注

传给新 EngineOptions() 的参数告知 AndEngine 应如何设置该游戏的 Engine 对象:

- ☐ pFullscreen: 总是设定为 true, 因为游戏要全屏显示。
- ☐ ScreenOrientation 可以取以下两值之一:
  - ☐ LANDSCAPE: 横屏的手机游戏用此值
  - ☐ PORTRAIT: 如果需要竖屏则用此值
- ☐ pResolutionPolicy: 告知 Engine 对象如何处理不同的设备屏幕大小:
  - ☐ RatioResolutionPolicy。在保持原有长宽比不变的情况下尽可能地拉伸图像以填满屏幕。这是本书中常用的方式。
  - ☐ FillResolutionPolicy。忽略长宽比, 直接填满整个屏幕。
  - ☐ FixedResolutionPolicy。使用原有的固定图像大小, 不进行缩放以适应不同屏幕大小。
  - ☐ RelativeResolutionPolicy。将原图缩放到指定的倍数。如果要为某些特定设备开发, 可以使用此策略。

pCamera 对象表示玩家观察场景所用的摄像机视点。Camera 对象的构造器如下:

```
public Camera(final float pX, final float pY, final float pWidth,
              final float pHeight)
```

其中 pX 与 pY 表示原点坐标, pWidth 与 pHeight 表示场景可见部分的长和宽。长度单位都是像素。

### 3.2.2 其他 Engine 类

本书一般使用前述的默认 Engine 类, 如果读者需要, 也可以选择其他 Engine 类。以下这些类都是 Engine 类的子类, 故而继承了其基本特性。

#### FixedStepEngine

```
FixedStepEngine(EngineOptions pEngineOptions, int pStepsPerSecond)
```

默认的 Engine 对象将尽快执行游戏循环, 并立刻开始下一轮循环逻辑, 这种做法可以达到最高的游戏速度与帧速率<sup>⊖</sup>, 同时 Engine 对象会把握好所花的时间, 以使坐标变换等其他操作可以流畅、稳定地执行。然而如果基于某些原因, 开发者需要让游戏在所有设备上都以某个固定的步长运行, 则可以使用该类。

#### LimitedFPSEngine

```
LimitedFPSEngine(EngineOptions pEngineOptions, int pFramesPerSecond)
```

---

⊖ 原文为 frame rate, 又叫 frame per second, 简称 FPS, 中文为“帧速率”、“画面更新率”、“影格速度”, 通常指每秒钟游戏或动画的更新次数, FPS 越高, 效果越流畅。——译者注

如前所述，默认的 Engine 对象会尽可能快地运行，从而导致持续变动的帧速率。AndEngine 会将帧速率记录到 LogCat 控制台（细节将会在本章稍后的例子中详述），Engine 对象将根据两帧之间的实际时间差来计算移动对象的位置。如果由于某些原因，需要游戏在所有设备上都以固定的帧速率运行，那么可以试着使用 LimitedFPSEngine。这个类之所以叫做“Limited”（受限的）而非“Fixed”（固定的），是因为在某些设备上可能无法达到预期的帧速率，所以实际上是设定了帧速率的上限。

### SingleSceneSplitScreenEngine

```
SingleSceneSplitScreenEngine(EngineOptions pEngineOptions,
                             Camera pSecondCamera)
```

默认的 Engine 类一次只能显示一个场景，并且该场景只有一个摄像头。如果需要在两个窗口中分别从不同的视点显示同一个场景，那么应该使用这个类。比如说，在一个双人玩的第一人称射击类游戏中，每个玩家都有自己的视点。

### DoubleSceneSplitScreenEngine

```
DoubleSceneSplitScreenEngine(EngineOptions pEngineOptions,
                              Camera pSecondCamera)
```

如果游戏需要分屏显示不同的场景，那么应该使用这个类。例如，可以开发一个第一人称射击游戏，其中一个屏幕显示当前玩家所看到的内容，另一个显示整个战斗场景的俯视图。“第一个”场景将会显示在左边，“第二个”在右边。

## 3.3 为 V3 增加菜单屏幕

接下来，将为所开发的游戏增加开始菜单。有两种菜单结构可选。一种是 Android 系统的 MenuView 类，当然，它有一整套构建与显示菜单的 API，可以响应 Menu 按钮并接受菜单输入。另外，AndEngine 也提供了一个菜单系统，该系统将菜单集成到游戏当中，支持文本菜单与图形菜单。这些菜单与游戏的其余部分一样，都是用相同的 OpenGL 接口所绘制的。在本节中，这两种菜单都会讲到，此外还会讲解如何捕获 Menu 按钮的按键事件并显示弹出式菜单。Android 系统的 MenuView 在显示场景时是禁用的，开发者仍然可以在 AndEngine 所用的 SurfaceView 上使用 MenuView，但这么做会使玩家在操作上产生误解。

### 3.3.1 AndEngine 的菜单

菜单在 AndEngine 的场景类中算是特殊的，因为它显示的是一个由文本或图形组成的有序列表，而且在玩家触摸某个菜单项时，要处理这个触摸事件。菜单也可以显示



带动画效果的菜单项。

### MenuScene.java

MenuScene 类是 Scene 的子类，它用于实例化游戏的菜单场景。此类有 4 个构造器：

- ❑ MenuScene()
- ❑ MenuScene(final Camera pCamera)
- ❑ MenuScene(final IOnMenuItemClickListener pOnMenuItemClickListener)
- ❑ MenuScene(final Camera pCamera, final IOnMenuItemClickListener pOnMenuItemClickListener)

参数的含义如下。

- ❑ pCamera：用以显示场景的 Camera 对象，其值通常与 onLoadEngine() 方法中声明的那个相同，但也可以是任意的 Camera 对象。
- ❑ pOnMenuItemClickListener()：当用户点击某个菜单项时，通过此监听器通知该事件。pCamera 与 pOnMenuItemClickListener 的值也可以通过相关的 setter 与 getter 方法来存取。该类也包含了一些与下列实例变量相关的方法：

- ❑ mMenuItems：存放菜单项的 ArrayList 对象。
- ❑ mMenuAnimator：显示菜单场景时所播放的动画。

### MenuScene 类的菜单项与场景相关方法

这些方法如下所述：

- ❑ addMenuItem(final IMenuItem pMenuItem)

该方法将 pMenuItem 加入所显示的菜单项中。

- ❑ getMenuItemCount()

该方法返回菜单列表中所含菜单项的数目。

- ❑ setChildScene(final Scene pChildScene, final boolean pModalDraw, final boolean pModalUpdate, final boolean pModalTouch), getChildScene(), clearChildScene()

setChildScene 方法将某个场景作为子场景附加到当前菜单场景上。这些方法将在第 4 章详述。

- ❑ setMenuAnimator(final IMenuAnimator pMenuAnimator)

该方法设置菜单场景所用播放的动画。

### TextMenuItem.java

该类代表一个文字菜单项。它只有一个构造器。

```
TextMenuItem(final int pID, final Font pFont, final String pText)
```

其中参数的意义如下：

- `pID`: 唯一的整数标识符, 用于在 `onClick()` 回调方法中标识被点击的菜单, 其他地方也可能用到它。
- `pFont`: 显示菜单时所用的字体, 本书第7章将详细研究它。在这里, 只要知道这个参数代表字型(例如 Courier)、字号、字体风格等属性就好。
- `pText`: 该菜单项所显示的文本。

#### `SpriteMenuItem.java`

该类代表一个以精灵方式来显示的图形菜单项, 也是只有一个构造器。

```
SpriteMenuItem(final int pID, final TextureRegion pTextureRegion)
```

其中参数的意义如下:

- `pID`: 唯一的整数标识符, 意义与 `TextMenuItem` 中的参数相同。
- `pTextureRegion`: 用以显示图形菜单项的精灵贴图。本书第5章在讲到精灵的时候, 将详解贴图与贴图区域的使用法。

#### `AnimatedSpriteMenuItem.java`

这种菜单项是使用动画精灵来显示的, 只有一个构造器:

```
AnimatedSpriteMenuItem(final int pID,
    final TiledTextureRegion pTiledTextureRegion)
```

其参数含义与 `SpriteMenuItem` 的构造器类似:

- `pID`: 唯一的整数标识符。
- `pTiledTextureRegion`: 本书第6章会提到, 这种瓦片贴图是用于显示动画精灵的, 该参数告知 `AndEngine` 应在此菜单项上显示的图案。

#### `ColorMenuItemDecorator.java`

该对象会让其所修饰的菜单项在被按下时短暂地改变外观。正如读者所想, 它改变该菜单项的颜色。此类有一个构造器, 如下所示:

```
ColorMenuItemDecorator(final IMenuItem pMenuItem,
    final float pSelectedRed, final float pSelectedGreen,
    final float pSelectedBlue, final float pUnselectedRed,
    final float pUnselectedGreen, final float pUnselectedBlue)
```

这一长串参数看起来似乎很恐怖, 其实只是代表了菜单项的两种颜色而已, 一种是用用户选中时显示的颜色, 一种是未选中时显示的颜色。颜色值在 `AndEngine` 中通常以浮点数表示, 按照红、绿、蓝分量的顺序排列, 取值可以从 0.0f(完全不存在)到 1.0f(完全存在)。稍后的范例会演示如何使用它。

### ScaleMenuItemDecorator.java

另一种标识某菜单项被选中的方式就是改变其大小，也许读者已经猜到了，这个类正是完成这项任务的，它有一个如下的构造器：

```
ScaleMenuItemDecorator(final IMenuItem pItem,
    final float pSelectedScale, final float pUnselectedScale)
```

此构造器接受三个参数：

- pItem: 被修饰的菜单项
- pSelectedScale: 菜单项被选中时的缩放因子
- pUnselectedScale: 菜单项未被选中时的缩放因子（通常是 1.0f）

### 3.3.2 构建 V3 的开始菜单

我们先以文本方式创建 V3 的主菜单，然后再创建一个图形化的弹出式菜单，用以在按下手机 Menu 按钮时显示。主菜单（静态菜单）将提供玩家所需的主要功能，弹出式菜单将包含关于页面与退出按钮（尽管手机的 Back 按钮也能退出游戏）。为了演示菜单的动画效果，我们将制作一个从屏幕左方逐渐滑入屏幕中央的弹出式菜单。图 3.1 是一张主菜单的截图，该图未显示弹出式菜单。

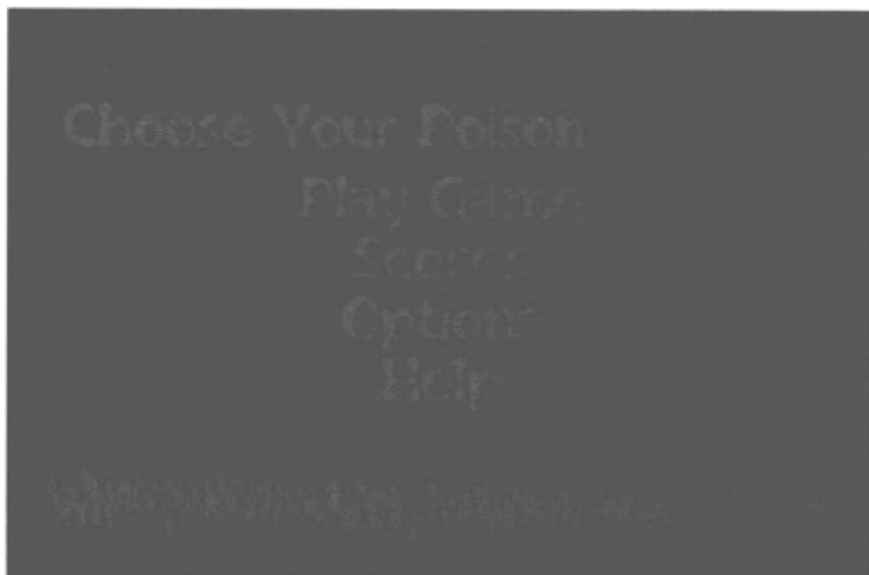


图 3.1 V3 开始菜单

### 3.3.3 创建菜单

创建菜单需要做两件事：

1. 创建需要显示的菜单场景。
2. 修改 StartActivity 类，让其在显示启动画面数秒之后，切入到菜单场景中来。

为了创建菜单场景，需要新建一个名为 MainMenuActivity 的 Activity 类，该类中将使用一个名为 MenuScene 的类，它继承自 AndEngine 中的 Scene 类。创建菜单场景所需的代码如程序清单 3.2 所示。

#### 程序清单 3.2 MainMenuActivity.java

```
package com.pearson.lagp.v3;

+imports

public class MainMenuActivity extends BaseGameActivity implements
```

```

IOnMenuItemClickListener {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    protected static final int MENU_ABOUT = 0;
    protected static final int MENU_QUIT = MENU_ABOUT + 1;
    protected static final int MENU_PLAY = 100;
    protected static final int MENU_SCORES = MENU_PLAY + 1;
    protected static final int MENU_OPTIONS = MENU_SCORES + 1;
    protected static final int MENU_HELP = MENU_OPTIONS + 1;

    // =====
    // 字段
    // =====

    protected Camera mCamera;

    protected Scene mMainScene;

    private Texture mMenuBackTexture;
    private TextureRegion mMenuBackTextureRegion;

    protected MenuScene mStaticMenuScene, mPopUpMenuScene;

    private Texture mPopUpTexture;
    private Texture mFontTexture;
    private Font mFont;
    protected TextureRegion mPopUpAboutTextureRegion;
    protected TextureRegion mPopUpQuitTextureRegion;
    protected TextureRegion mMenuPlayTextureRegion;
    protected TextureRegion mMenuScoresTextureRegion;
    protected TextureRegion mMenuOptionsTextureRegion;
    protected TextureRegion mMenuHelpTextureRegion;
    private boolean popupDisplayed;

    // =====
    // 构造器
    // =====

    // =====
    // Getter 和 Setter
    // =====

    // =====
    // 覆写超类及接口中的方法
    // =====

```

```

@Override
public Engine onLoadEngine() {
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
        CAMERA_HEIGHT), this.mCamera));
}

@Override
public void onLoadResources() {
    /* 载入 Font 与 Textures 对象。 */
    this.mFontTexture = new Texture(256, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);

    FontFactory.setAssetBasePath("font/");
    this.mFont = FontFactory.createFromAsset(this.mFontTexture,
        this, "Flubber.ttf", 32, true, Color.RED);
    this.mEngine.getTextureManager().loadTexture(this.mFontTexture);
    this.mEngine.getFontManager().loadFont(this.mFont);

    this.mMenuBackTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mMenuBackTextureRegion =
        TextureRegionFactory.createFromAsset(this.mMenuBackTexture,
            this, "gfx/MainMenu/MainMenuBk.png", 0, 0);
    this.mEngine.getTextureManager().loadTexture(this.mMenuBackTexture);

    this.mPopUpTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mPopUpAboutTextureRegion =
        TextureRegionFactory.createFromAsset(this.mPopUpTexture,
            this, "gfx/MainMenu/About_button.png", 0, 0);
    this.mPopUpQuitTextureRegion =
        TextureRegionFactory.createFromAsset(this.mPopUpTexture,
            this, "gfx/MainMenu/Quit_button.png", 0, 50);
    this.mEngine.getTextureManager().loadTexture(this.mPopUpTexture);
    popupDisplayed = false;
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    this.createStaticMenuScene();
    this.createPopUpMenuScene();

    /* 将摄像头置于背景中央。 */
    final int centerX = (CAMERA_WIDTH -
        this.mMenuBackTextureRegion.getWidth()) / 2;

```

```

        final int centerY = (CAMERA_HEIGHT -
            this.mMenuBackTextureRegion.getHeight()) /
            2;

        this.mMainScene = new Scene(1);
        /* 增加背景与静态菜单。 */
        final Sprite menuBack = new Sprite(centerX,
            centerY, this.mMenuBackTextureRegion);
        mMainScene.getLastChild().attachChild(menuBack);
        mMainScene.setChildScene(mStaticMenuScene);

        return this.mMainScene;
    }

    @Override
    public void onLoadComplete() {
    }

    @Override
    public boolean onKeyDown(final int pKeyCode,
        final KeyEvent pEvent) {
        if(pKeyCode == KeyEvent.KEYCODE_MENU &&
            pEvent.getAction() == KeyEvent.ACTION_DOWN) {
            if(popupDisplayed) {
                /* 移除菜单并重置 popupDisplayed 变量。 */
                this.mPopUpMenuScene.back();
                mMainScene.setChildScene(mStaticMenuScene);
                popupDisplayed = false;
            } else {
                /* 将菜单加入场景。 */
                this.mMainScene.setChildScene(
                    this.mPopUpMenuScene, false, true, true);
                popupDisplayed = true;
            }
            return true;
        } else {
            return super.onKeyDown(pKeyCode, pEvent);
        }
    }

    @Override
    public boolean onOptionsItemSelected(final MenuScene pMenuScene,
        final IMenuItem pMenuItem,
        final float pMenuItemLocalX,
        final float pMenuItemLocalY) {
        switch(pMenuItem.getID()) {
            case MENU_ABOUT:
                Toast.makeText(MainMenuActivity.this,
                    "About selected",
                    Toast.LENGTH_SHORT).show();
                return true;
            case MENU_QUIT:
                /* 结束 Activity 对象的生命期。 */

```



```

        this.finish();
        return true;
    case MENU_PLAY:
        Toast.makeText(MainMenuActivity.this,
            "Play selected", Toast.LENGTH_SHORT).show();
        return true;
    case MENU_SCORES:
        Toast.makeText(MainMenuActivity.this,
            "Scores selected",
            Toast.LENGTH_SHORT).show();
        return true;
    case MENU_OPTIONS:
        Toast.makeText(MainMenuActivity.this,
            "Options selected",
            Toast.LENGTH_SHORT).show();
        return true;
    case MENU_HELP:
        Toast.makeText(MainMenuActivity.this,
            "Help selected", Toast.LENGTH_SHORT).show();
        return true;
    default:
        return false;
    }
}

// =====
// 方法
// =====

protected void createStaticMenuScene() {
    this.mStaticMenuScene = new MenuScene(this.mCamera);
    final IMenuItem playMenuItem = new ColorMenuItemDecorator(
        new TextMenuItem(MENU_PLAY, mFont, "Play Game",
            0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    playMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mStaticMenuScene.addMenuItem(playMenuItem);

    final IMenuItem scoresMenuItem =
        new ColorMenuItemDecorator(
            new TextMenuItem(MENU_SCORES, mFont, "Scores",
                0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    scoresMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mStaticMenuScene.addMenuItem(scoresMenuItem);

    final IMenuItem optionsMenuItem =
        new ColorMenuItemDecorator(
            new TextMenuItem(MENU_OPTIONS, mFont, "Options",
                0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    optionsMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
}

```

```

        this.mStaticMenuScene.addMenuItem(optionsMenuItem);
        final IMenuItem helpMenuItem = new ColorMenuItemDecorator(
            new TextMenuItem(MENU_HELP, mFont, "Help"),
            0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
        helpMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
            GL10.GL_ONE_MINUS_SRC_ALPHA);
        this.mStaticMenuScene.addMenuItem(helpMenuItem);
        this.mStaticMenuScene.buildAnimations();

        this.mStaticMenuScene.setBackgroundEnabled(false);

        this.mStaticMenuScene.setOnMenuItemClickListener(this);
    }

    protected void createPopUpMenuScene() {
        this.mPopUpMenuScene = new MenuScene(this.mCamera);

        final SpriteMenuItem aboutMenuItem =
            new SpriteMenuItem(MENU_ABOUT,
                this.mPopUpAboutTextureRegion);
        aboutMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
            GL10.GL_ONE_MINUS_SRC_ALPHA);
        this.mPopUpMenuScene.addMenuItem(aboutMenuItem);
        final SpriteMenuItem quitMenuItem = new SpriteMenuItem(
            MENU_QUIT, this.mPopUpQuitTextureRegion);
        quitMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
            GL10.GL_ONE_MINUS_SRC_ALPHA);
        this.mPopUpMenuScene.addMenuItem(quitMenuItem);
        this.mPopUpMenuScene.setMenuAnimator(
            new SlideMenuAnimator());

        this.mPopUpMenuScene.buildAnimations();

        this.mPopUpMenuScene.setBackgroundEnabled(false);

        this.mPopUpMenuScene.setOnMenuItemClickListener(this);
    }

    // =====
    // 内部类与匿名类
    // =====
}

```

一眼望去，代码很多，不过不必担心，我们将其分成小段来研究，其中的许多问题会在本书的后续章节中讲述，这里只是提一下它们。

### 3.3.4 MainMenuActivity 类

在 import 指令之后，还是依惯例将 MainMenuActivity 类声明为 BaseGameActivity 的子类，同时令其实现 IOnMenuItemClickListener 接口，用以响应菜单项的点击事件。

### 3.3.5 常数与字段

其后定义了许多常数。名为 MENU\_XXX 的常数是菜单项的标识符，在稍后的菜单项创建与监听器回调方法中都将用到。代码中将其分成两组，一组用于创建静态菜单，另一组用于创建弹出式菜单，说实在的，这种分组方式有点儿武断。

### 3.3.6 onLoadResources() 方法

在声明完本 Activity 类所需字段之后，我们按惯例声明一组覆写方法。该类的 onLoadEngine() 方法与 StartActivity 类的基本相同，由此可见本书中 Activity 类的该方法基本都会采用这个模式。

onLoadResources() 方法载入贴图及游戏所需的各种资源。要显示文本菜单就得有字体，所以第一件事就是载入字体。创建一个 Texture 对象来存放字体贴图，再通过 FontFactory.createFromAsset() 方法将 Font 对象创建起来。编译游戏之前，需要在 assets 目录下创建名为 font 的子目录，并将 TrueType 字体文件 Flubber.ttf 导入。createFromAsset() 方法的参数表明，所加载的 Flubber 字体，高度为 32 像素，颜色为红色。第 7 章将会讲述字体与文本的知识。

在 onLoadResources() 方法中，紧接着创建了 Texture 与 TextureRegion 对象，它们用于生成静态菜单场景的背景图。如图 3.1 所示，菜单包含一幅黑色背景图，上面写有“Choose Your Poison”的标题，下方有一长条草地。这一部分已被独立做成一张图片，并存储为 PNG 格式，现在请将其导入 assets/gfx 文件夹。其后的代码将 Texture 对象载入到 Texture Manager 中，这两个类将在第 4 章与第 5 章深入讲述。

本段代码也为弹出式菜单的两个菜单项分别创建了 Texture 与 TextureRegion 对象，因为弹出菜单用的是图形式菜单，所以需要从事先为每个菜单项按钮准备好的 PNG 文件中创建 Texture 对象。

### 3.3.7 onLoadScene() 方法

该类也覆写了 onLoadScene() 方法。与 StartActivity 的相同，代码首先开启了帧速率记录器。其后调用两个方法分别创建两种菜单场景，一个用于创建首先要显示的静态菜单，另一个用于创建按下 Menu 键之后所弹出的菜单。然后创建一个用于放置所有其他场景对象的 MainScene 对象，并将摄像机的中心对准它。最后，为菜单背景创建一个精灵，将其加入菜单场景，并将静态菜单场景添加为主场景的一个子对象。

### 3.3.8 createStaticMenuScene<sup>⊖</sup>() 方法与 createPopUpScene() 方法

在分析 onKeyDown() 与 onOptionsItemSelected() 方法的代码之前，先跳过去看一

⊖ 原书此处为“createMenuScene”，根据代码应为 createStaticMenuScene，遂修改。——译者注

下这两个用于创建菜单场景的方法：`createStaticMenuScene()`与`createPopUpScene()`。`createStaticMenuScene()`首先创建场景对象，其次创建4个`TextMenuItem`对象，创建时传入菜单项的ID以便在其被点击时据此识别该菜单项，还会传入使用的字体及需要显示的文字。此时，可以很方便地使用在`onLoadResources()`方法中所加载的字体了。接着告知OpenGL在后面的场景中如何渲染菜单项上的文字，并将其加入菜单场景。然后再调用两个方法`buildAnimations()`与`setBackgroundEnabled()`，设置好预期的场景显示效果。最后设置监听器，以便在用户点击菜单项时，回调方法`onMenuItemClicked()`能够被通知到。

`createPopUpMenuScene()`方法也很相似，只是它创建的是图形菜单而非文字菜单。这段代码创建了`SpriteMenuItem`对象并用`onLoadResources()`方法中载入的Texture资源来填充它。

### 3.3.9 onKeyDown()方法与onMenuItemClicked()方法

前面的两个方法已将菜单场景设置好了，接下来看看两个相应用户动作的方法：`onKeyDown()`和`onMenuItemClicked()`。

`onKeyDown()`方法检测用户是否按下了Menu键，如果没有，它将键值回传给Android系统处理。用户可以使用Menu键来显示或隐藏弹出式菜单，所以，在侦测到Menu键被按下时，`onKeyDown()`方法要判断弹出式菜单是否已经显示出来了。如果已经显示了，则要将其移除，并将静态菜单再次作为子对象加入到主场景中；如果尚未显示，则将其加入主场景中即可。

`onMenuItemClicked()`方法内的switch语句根据创建菜单项时所用的ID来判断具体是哪个菜单项被按下了。在这里，可以仅用一个方法和一条switch语句去判断被按下的菜单项，因为所有的ID都是独一无二的，不需要区分是静态菜单还是弹出式菜单。现在还没办法切换到其他场景去，所以在用户选择菜单项后，只显示一个桌面提示信息框就好。退出选项是个例外，它并不需要真的结束游戏，而是返回至先前的启动画面就好。

### 3.3.10 从启动画面切换到菜单

接下来要做的事情，是在启动画面显示3秒钟之后，切换到菜单场景去。其实最终是要用显示启动画面的这段时间来加载图像与声音资源，不过现在并没有资源需要加载。程序清单3.3是修改后的`StartActivity.java`文件。

程序清单 3.3 StartActivity.java

```
package com.pearson.lagp.v3;

+imports
```

```

public class StartActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    // =====
    // 字段
    // =====

    private Camera mCamera;
    private Texture mTexture;
    private TextureRegion mSplashTextureRegion;
    private Handler mHandler;

    // =====
    // 构造器
    // =====

    // =====
    // Getter 和 Setter
    // =====

    // =====
    // 覆写超类及接口中的方法
    // =====

    @Override
    public Engine onLoadEngine() {
        mHandler = new Handler();
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        return new Engine(new EngineOptions(true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,
                CAMERA_HEIGHT), this.mCamera));
    }

    @Override
    public void onLoadResources() {
        this.mTexture = new Texture(512, 512,
            TextureOptions.BILINEAR_PREMULTIPLYALPHA);
        this.mSplashTextureRegion =
            TextureRegionFactory.createFromAsset(this.mTexture,
                this, "gfx/Splashscreen.png", 0, 0);
        this.mEngine.getTextureManager().loadTexture(this.mTexture);
    }
}

```

```

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    final Scene scene = new Scene(1);

    /* 将摄像头置于启动画面中央。 */
    final int centerX = (CAMERA_WIDTH -
        this.mSplashTextureRegion.getWidth()) / 2;
    final int centerY = (CAMERA_HEIGHT -
        this.mSplashTextureRegion.getHeight()) / 2;

    /* 创建 Sprite 对象并将其加入 Scene 对象中。 */
    final Sprite splash = new Sprite(centerX, centerY,
        this.mSplashTextureRegion);
    scene.getLastChild().attachChild(splash);
    return scene;
}

@Override
public void onLoadComplete() {
    mHandler.postDelayed(mLaunchTask, 3000);
}

private Runnable mLaunchTask = new Runnable() {
    public void run() {
        Intent myIntent = new Intent(StartActivity.this,
            MainMenuActivity.class);
        StartActivity.this.startActivity(myIntent);
    }
};

// =====
// 方法
// =====

// =====
// 内部类与匿名类
// =====
}

```

这段代码的意思是，在延迟 3 秒钟之后，向 Android 系统投递一个 Intent 对象，用以切换到 MainMenuActivity。AndEngine 将场景视作普通的 Activity 对象，这么做的好处之一是，开发者可以像调用其他 Activity 对象那样来调用场景对象。

现在运行一下程序（用模拟器或用真机都行），会看到屏幕上显示启动画面 3 秒钟之后，静态菜单出现了。如果按下某个菜单项，相应的桌面提示框就会弹出来；如果按下了 Menu 键，那么“关于 (About)”和“退出 (Quit)”按钮将会显示出来。如果按下“关于”按钮，会出现另一个桌面提示框来；如果按下“退出”按钮的话，会切换到



启动画面，并一直停在这里。onLoadComplete() 方法不会被调用，因为该 Activity 对象已经被加载好了。

### 3.4 内存使用

趁这个机会，我们花几分钟时间来讲一个手机软件设计所面临的重要问题，那就是内存使用与垃圾收集。如果读者曾经开发过手机软件的话，应该会了解手机应用开发与桌面应用开发的一个重大差别：手机的资源很有限。这里所说的资源包括电池电量、内存、固定存储器、屏幕尺寸、处理器速度以及其他一些东西。在做手机游戏时，要始终考虑如何节省资源。

例如，在 Android 手机游戏开发中，内存使用与垃圾回收就是个大问题。大家都知道，游戏的 Java 二进制码是运行在 Dalvik 虚拟机上的，当其耗尽可用内存后，会调用垃圾回收器将不再被引用的对象回收掉。这个办法能够解决大部分的内存泄漏问题，但是当垃圾回收器运行时，会拖慢游戏的运行。开发者当然想极力减少系统进行垃圾回收的次数，可以尽可能多地重用已有对象来达到目的。看看源代码就会发现，AndEngine 非常善于利用对象池技术来分配与回收对象，这个方法虽然不是万灵丹，但是如果运用得合理，也是非常有效的。

### 3.5 “退出”选项

现在告诉大家一个猥琐的小秘密。其实说起来也不算什么，就是关于前一个菜单上的那个“退出”按钮的事情。大家可能知道，Android 系统从来都没有真正的“退出”选项。当用户要停止（其实是暂停执行）一个程序时，Android 推荐的方法是按下手机的“Back”键，这样系统就会回到主桌面。如果读者觉得这话难以理解，请在 Android 开发者文档（<http://developer.android.com>）中找到“Activity Lifecycle”这一段并看一看。应用程序可以在当前 Activity 类中的 onPause() 方法中关闭所有服务，并停止所有的计时器。

为了符合 Android 开发规范，自此将从菜单中除去“退出”选项。

### 3.6 总结

本章开头综述了游戏循环的概念，然后介绍 AndEngine 用以实现游戏循环的方法。在绝大多数情况下，AndEngine 的游戏循环是不可见的。开发者编写代码将所要完成的任务告知游戏循环，它会自动安排执行。

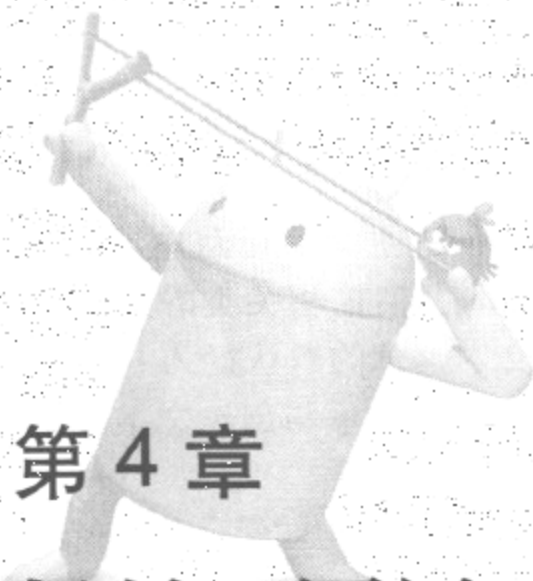
AndEngine 提供了许多有关菜单的类。本书给出了用于创建与管理文本菜单、图形化（精灵）菜单以及动画精灵菜单的若干方法。

范例游戏接下来要写的功能，是在启动画面结束后创建主菜单画面并显示给玩家。在写这部分代码的过程中，读者熟悉了 AndEngine 中菜单相关类的用法。这段代码创建了文本菜单并列出了几个菜单项，展示了文本菜单的默认行为；同时也创建了一个有若干选项的弹出式图形菜单，并比较了创建该菜单与创建文本菜单的异同。

现在大家已经掌握了基于 AndEngine 引擎开发游戏的基本结构，学会了如何启动游戏，如何处理不同的菜单项以及如何结束游戏。读者还了解到创建新场景与加载资源所使用的固定代码模式。现在应该研究一下游戏内容本身，以及游戏所需基本元素的细节问题了。这些将是第 4 章所要讲的内容。

### 3.7 习题

1. MainMenuActivity 代码在弹出式菜单上用 SlideAnimator 做出了滑动效果。如果将 SlideAnimator 应用于静态菜单，结果会如何？请解释为何静态菜单不会如预期的那样滑动。
2. 修改 MainMenuActivity.java，使得每个菜单项具有不同的颜色。
3. 修改 MainMenuActivity.java，使得菜单项在被选中时，尺寸稍微变大一些，而不要像现在这样变成灰色。



## 第4章

# 场景、图层、场景切换与实体修改器

- 4.1 AndEngine 的场景
- 4.2 创建游戏第1关的场景
- 4.3 总结
- 4.4 习题

在电影或戏剧中, 场景指的是在限定的时间段内产生动作的背景。电脑游戏也是以含有图像的数层场景组织起来的。如果读者曾经编辑过视频, 应该知道场景间的切换方式有很多种, 这些方式可以使得视频的播放过程看起来非常有趣。同样, 在电脑游戏中也要有场景间的切换, 并且要以一种统一的方式修改图形元素。开发者希望能修改它们的位置、缩放比例、颜色、旋转角度、透明度, 还可能想要修改移动路径。本章讲解如何用 AndEngine 创建与修改图层及图形元素。

## 4.1 AndEngine 的场景

在 AndEngine 中, 几乎所有可见的游戏元素都是 Entity 类的子类。Entity 对象可以通过 Modifier 对象来修改, 而且在特定的场景中, 可以通过修改来产生动画效果。本章主要关注场景问题, 不过, 由于其他的游戏可见元素(例如精灵、瓦片、图形、粒子、文本等)都是 Entity 对象, 故而关于 Modifier 的用法, 也适用于它们。

本章将会全面讲解 Entity 类的各种特性, 并深入讲解所有用于修改 Entity 属性的 Modifier 类和 Ease 函数。笔者感到抱歉的是, 这部分材料可能有些枯燥, 因为要讲很多的 Modifier 类, 每个类都有大量的选项设定, 还要讲若干个 Ease 函数。如果读者感到这些解释性的文字过于重复, 可以先跳过此部分, 把它当做 Modifier 类及 Ease 函数的参考手册来看。

### 4.1.1 实体/组件模型

在设计诸如 AndEngine 这样的游戏引擎时, 有多种不同的方法可选。鉴于大家已经知道, Android 应用几乎都是以 Java 语言来开发的, 故而首先考虑的方案应该用面向对象的方法设计。用这种方法设计的话, 游戏中的每种物体都有自己的类, 这些类会构成一个继承体系, 此外还会有用以划分功能组的抽象类存在。举例来说, V3 中可能会有 Vampire 类、Virgin 类以及其他与游戏有关的类(例如 Bullet 类、Hatchet 类、Cross 类、Tombstone 类), 还会有一个抽象类 Character 作为 Vampire 类和 Virgin 类的超类。Bullet、Hatchet 和 Cross 类可能都继承自一个抽象类 Weapon。每个类都将有其外观与行为。

这种方法当然可行, 但是会很快导致代码难于管理。如果开发者不会改变每种物体所具有的行为, 那么就可以先设计出这样一种符合逻辑的面向对象体系, 然后再用代码将其实现。当然, 如果读者曾经做过软件开发的话, 就会知道这么做不会如想象的那般容易。在编码过程中会产生新的想法, 而且准备发布游戏的过程中, 会持续地修改游戏。僵化地运用面向对象方法会导致代码变得脆弱, 难以应对随时出现的修改; 同时也会让持续的代码重构变得困难。

让游戏设计变灵活的方法之一, 就是使用实体/组件式设计方法, AndEngine

即是采用此法。该方法将所有的物体都看做 Entity 对象，而不是为每种物体设计一个类。这些实体对象稍后会被动态地设置属性（或组件），用以描述其显示图像（TextureRegion）、移动方式（Modifier）、外观的改变方式等。Entity 对象之间的互动（例如碰撞）以及 Entity 与玩家的互动（例如触摸事件）如果发生，会被内置于 Activity 类之中的机制所处理。

这两种方法（面向对象式与实体/组件式）都各有其缺点，AndEngine 在两者间做了恰当的折中。AndEngine 是围绕 Entity 类进行设计的，但是正如本章后面所述，Entity 类也定义了许多针对所有游戏中可见元素的实例变量。

如果以上这段讨论听起来有点抽象的话，随着 V3 游戏开发的深入，它会逐渐变得清晰起来。读者可以搜索“entity component game design”，将会看到很多解释此概念的信息。

### 4.1.2 Entity 类

作为包括 Scene 类在内所有可见物的超类，Entity 类包含了可见物体的公共属性，这些属性可以通过 getter 与 setter 方法存取，设置 Modifier 来修改 Entity 对象也是通过 setter 方法进行的。这些属性的含义如下所述：

- ❑ float mX, mY: 此 Entity 对象的当前位置
- ❑ float mInitialX, mInitialY: 此 Entity 对象的初始位置
- ❑ boolean mVisible: 此 Entity 对象是否可见
- ❑ boolean mIgnoredUpdate: 此 Entity 对象是否要更新
- ❑ int mZindex: 此 Entity 对象在显示序列之中的位置
- ❑ IEntity mParent: 此 Entity 对象的上层对象
- ❑ SmartList<IEntity> mChildren: 此 Entity 的子对象列表
- ❑ EntityModifierList mEntityModifiers: 应用于此 Entity 的 Modifier 对象列表
- ❑ UpdateHandlerList mUpdateHandlers: 应用于此 Entity 的 UpdateHandler 对象列表
- ❑ float mRed, mGreen, mBlue: 此 Entity 对象的颜色
- ❑ float mAlpha: 此 Entity 对象的透明度
- ❑ float mX, mY: 此 Entity 对象的当前位置
- ❑ float mRotation: 此 Entity 对象的旋转角度
- ❑ float mRotationCenterX, mRotationCenterY: 旋转此 Entity 对象所参考的中心点
- ❑ float mScale: 此 Entity 对象的缩放因子
- ❑ float mScaleCenterX, mScaleCenterY: 缩放此 Entity 对象所参考的中心点

### 4.1.3 构造器

尽管存在如下两个用于 Entity 的构造器，但实际上，开发者很少直接使用它们，而是使用其子类的构造器。



```
Entity()
Entity(final float pX, final float pY)
```

第二个构造器的参数，既代表初始坐标，也代表当前坐标。

#### 4.1.4 Entity 类的位置相关方法

在 AndEngine 中，Entity 对象的位置即是其中心点的位置。Entity 对象既保存了当前的位置，也保存了其初始位置。以下两个方法：

```
float getX()
float getY()
```

分别返回 Entity 对象当前位置的 x、y 坐标。

```
float getInitialX()
float getInitialY()
```

分别返回 Entity 对象初始位置的 x、y 坐标。

```
void setPosition(final float pX, final float pY)
```

设定 Entity 对象的当前位置。

```
setPosition(final IEntity pOtherEntity)
```

将传入的 Entity 对象位置设定为此 Entity 对象的位置。

```
void setInitialPosition()
```

将 Entity 对象的当前位置设定为其初始位置。（注意：该方法并不修改 Entity 对象的初始位置。）

#### 4.1.5 Entity 类的缩放相关方法

Entity 对象在绘制时的大小，是用其尺寸乘以其缩放因子计算得来的。Entity 有一个缩放操作所参考的中心点，它是唯一一个在缩放操作前后位置不改变的点。缩放因子设为 2.0f 的意思是，该 Entity 对象被绘制时的大小，是其“正常”尺寸的 2 倍。与缩放相关的方法总结如下：

```
boolean isScaled()
```

如果缩放因子不是 1.0f，则返回 true。

```
float getScaleX()
float getScaleY()
```



分别返回横向与纵向的缩放因子。

```
void setScaleX(final float pScaleX)
void setScaleY(final float pScaleY)
```

分别设定横向与纵向的缩放因子。

```
void setScale(final float pScale)
```

设定两个方向共用的缩放因子。

```
void setScale(final float pScaleX, final float pScaleY)
```

通过一次方法调用来设定两个方向的缩放因子。

```
float getScaleCenterX()
float getScaleCenterY()
```

返回缩放中心点的位置。

```
void setScaleCenterX(final float pScaleCenterX)
void setScaleCenterY(final float pScaleCenterY)
```

分别设定横向与纵向的缩放中心点坐标。

```
void setScaleCenter(final float pScaleCenterX, final float pScaleCenterY)
```

通过一次方法调用来设定缩放中心点的坐标。

#### 4.1.6 Entity 类的颜色相关方法

在 AndEngine 中，Entity 对象的颜色是一个乘数，会应用于该对象所包含的底层图像上。举例来说，如果该 Entity 是精灵，并且其上填充了贴图的话，这些颜色值将会与精灵的颜色值相乘<sup>①</sup>，运算的结果作为最终显示的颜色值。这个过程可以想象为给该 Entity 对象染色。

AndEngine 中的颜色值一般分解为红、绿、蓝、透明度分量，其取值从 0.0f（无颜色饱和度<sup>②</sup>）到 1.0f（颜色饱和度最大）<sup>③</sup>。

```
float getRed()
float getGreen()
float getBlue()
```

① 例如某 Entity 对象的颜色值是 0.5f，其上贴图的颜色值是 0.3f，则最终的颜色值是  $0.5f \times 0.3f = 0.15f$ 。——译者注

② 原文为 intensity。——译者注

③ 对于透明度来说，0.0f 代表完全透明，1.0f 代表完全不透明。——译者注

```
float getAlpha()
```

以上每个方法均返回 Entity 对象颜色的对应分量。

```
void setColor(final float pRed,final float pGreen,final float pBlue)
```

设定此 Entity 对象的三个颜色分量。

```
void setColor(final float pRed,final float pGreen,final float pBlue,final float pAlpha)
```

设定此 Entity 对象的三个颜色分量以及透明度的值。

### 4.1.7 Entity 类的旋转相关方法

与缩放相似, Entity 对象也有旋转角度与旋转中心点, 旋转角度可以小于 0 度或大于 360 度, 正角度值代表逆时针旋转。

```
float getRotation()
```

返回当前的旋转角度。

```
setRotation(final float pRotation)
```

设定当前的旋转角度。

```
float getRotationCenterX()  
float getRotationCenterY()
```

返回当前旋转中心点的横、纵坐标。

```
void setRotationCenterX(final float pRotationCenterX)  
void setRotationCenterY(final float pRotationCenterY)
```

分别设定旋转中心点的横、纵坐标。

```
void setRotationCenter(final float pRotationCenterX,final float pRotationCenterY)
```

通过一次方法调用来设定旋转中心点的坐标。

### 4.1.8 管理子对象

很多书籍都曾讲到子对象管理的话题, 不过这里专门谈一下 Entity 的子对象管理。Entity 对象通常是分层组织起来的, 以便于批量的修改。如果某个上层对象被修改了, 那么修改的效果会自动应用于它的所有子对象, 但是反过来说, 针对子对象的修改效果则不会自动传递到上层对象。管理子对象的相关方法如下所述:

```
IEntity IEntity getFirstChild()
```

```
IEntity getLastChild()
```

获得此 Entity 的第一个及最后一个子对象。最后一个子对象通常就是新加入对象的上级对象（请参看 attachChild()）。

```
void attachChild(final IEntity pEntity)
```

将新对象加入到此 Entity 对象，并作为其最后一个子对象。通常以这种方式使用此方法：

```
parent.getLastChild().attachChild(newchild);  
void detachChildren()  
boolean detachChild(final IEntity pEntity)
```

删除此 Entity 对象全部的或特定的某一子对象。

```
int getChildCount()
```

返回当前子对象的数目。

```
IEntity getChild(final int pIndex)
```

返回参数所在位置的（第 n 个）子对象。

```
void sortChildren()
```

按照 z 轴索引排列子对象（参见 setZindex()）。

#### 4.1.9 管理 Modifier

Modifier 的作用是以编程的方式来修改 Entity 对象的关键属性。开发者可以用它来修改 Entity 的位置、缩放因子、颜色、旋转角度、透明度等。修改效果可以立即生效，也可以在某个时间段内逐步生效。本章稍后将详述主要的 Modifier 类。Entity 类中与 Modifier 管理相关的方法如下所述：

```
void registerEntityModifier(final IEntityModifier pEntityModifier)
```

将指定的 Modifier 注册到此 Entity 对象上，该 Modifier 将在对象被显示时生效。

```
boolean unregisterEntityModifier(final IEntityModifier pEntityModifier)
```

将指定的 Modifier 移除，使它不再作用于此 Entity 对象（如果操作成功则返回 true）。

```
void clearEntityModifiers()
```

将此 Entity 对象的所有 Modifier 移除。

#### 4.1.10 其他有用的 Entity 类方法

有一些 Entity 类的方法无法归入上述类别：有的控制 Entity 对象的可见性，有的修

改渲染顺序，还有的支持将任意的数据添加到 Entity 对象上。

```
void setVisible(final boolean pVisible)
boolean isVisible()
```

设置与查询此 Entity 是否可见，可见性与透明度属性无关，一个对象由不可见变为可见，其透明度不会改变。

```
void setZIndex(final int pZIndex)
int getZIndex()
```

设置或查询此 Entity 的 z 轴索引 (Z-index)。高索引的对象将绘制在低索引者的前面。该方法设置了 z 轴索引，但是仍需调用上层对象的 sortChildren() 方法使之重新排列子对象的绘制顺序。

```
void setUserData(final Object pUserData)
Object getUserData()
```

设定或获取用户所定义的任意数据。开发者可以随意向此 Entity 上附加任意数据。

#### 4.1.11 Layer 类

AndEngine 中的 Layer 类是 Entity 类的子类，它几乎没有在超类的基础上添加什么，但是开发者可以用 Layer 组织 Scene 类所需的图形，为 Layer 设定 z 轴索引以及其他属性。该类有两个构造器：

```
Layer()
Layer(final float pX, final float pY)
```

第一个构造器的作用相当于用 0.0f 作为 pX 和 pY 的值调用第二个构造器，两者都会调用超类的构造器。以上就是 Layer 类的全部了。

#### 4.1.12 Scene 类

AndEngine 提供了 Scene 类，开发者可以用该类来创建游戏中的场景。其属性如下：

- Scene mParentScene：每个 Scene 对象都有一个可选的<sup>①</sup>上层 Scene 对象。
- Scene mChildScene：每个 Scene 对象都有一个可选的子 Scene 对象。
- SmartList<ITouchArea> mTouchAreas：Scene 对象知道如何接受用户的触摸，并且具有一系列可触摸的区域。
- IOnSceneTouchListener mOnSceneTouchListener：当此 Scene 对象被触摸时，则

① 原文为 optional，在 Java 程序设计中，可选的意味着在不需要该参数时，其值可以设为 null，下文同此含义。——译者注

回调该监听器。

- ❑ `IOAreaTouchListener mOnAreaTouchListener`：当此 Scene 对象的某一个触摸区域被触摸时，则回调监听器。
- ❑ `RunnableHandler mRunnableHandler`：每个 Scene 对象可以拥有一个 `RunnableHandler` 对象。
- ❑ `IBackground mBackground`：每个 Scene 对象都有背景，默认的背景是黑色色块。
- ❑ `boolean mOnAreaTouchTraversalBackToFront`：该标志位决定了触摸事件的处理顺序。
- ❑ 包含图形的 Layer 对象可以作为子对象附加到 Scene 对象上。

### 构造器

Scene 类有一个构造器：

```
Scene(final int pLayerCount)
```

Scene 对象一旦被创建，其图层数就不可改变，如果需要在游戏中加入一个 Layer 对象，就要用新的数值重新实例化 Scene 对象。

### 4.1.13 背景管理

有若干方法可以设置与获取 Scene 对象的背景，也有用于设置并测试背景是否已启用的方法。

```
IBackground getBackground()
void setBackground(final IBackground pBackground)
void setBackgroundEnabled(final boolean pEnabled)
boolean isBackgroundEnabled()
```

如先前所述，Scene 的默认背景是成黑色的，直到被设置成其他值为止。

### 4.1.14 子 Scene 对象管理

Scene 对象可以有一个非 Layer 类型的子对象。如下方法与管理该特殊子 Scene 对象有关：

```
void setChildScene(final Scene pChildScene)
void setChildScene(final Scene pChildScene, final boolean pModalDraw, final
    boolean pModalUpdate, final boolean pModalTouch)
void setChildSceneModal(final Scene pChildScene)
Scene getChildScene()
boolean hasChildScene()
void clearChildScene()
```

如果第二个 `setChildScene()` 方法的 boolean 参数均为 true 的话，将会使子 Scene 成为模态的，这意味着上级 Scene 在子对象拥有焦点时将暂停运作。第三个方法等同于用

true 作为 boolean 参数去调用第二个方法。

#### 4.1.15 Layer 对象管理

Layer 对象的管理要使用从 Entity 类继承下来的子对象管理方法。在 Scene 类中只多定义了一个用于管理 Layer 对象的方法：

```
void sortLayers()
```

该方法将 Scene 类的子 Layer 对象按照 z 轴顺序重新排列。

#### 4.1.16 上级 Scene 对象管理

Scene 对象可以拥有一个上级 Scene，当它作为子对象被加入到其他 Scene 对象上时，其上级 Scene 会被自动设置。开发者也可以通过以下方法自行设置上级 Scene：

```
void setParentScene(final Scene pParentScene)
```

该方法没有对应的获取方法，而且此实例变量（指 mParentScene）的访问级别为 protected<sup>⊖</sup>。该值用于管理后台 Scene 对象的切换。

#### 4.1.17 触摸区域管理

Scene 对象知道如何响应触摸事件，并且有如下方法用于创建触摸区域及响应其事件：

```
void registerTouchArea(final ITouchArea pTouchArea)
boolean unregisterTouchArea(final ITouchArea pTouchArea)
boolean unregisterTouchAreas(final ITouchAreaMatcher pTouchAreaMatcher)
void clearTouchAreas()
ArrayList<ITouchArea> getTouchAreas()
boolean onSceneTouchEvent(final TouchEvent pSceneTouchEvent)
boolean onAreaTouchEvent(final TouchEvent pSceneTouchEvent,
    final float sceneTouchEventX, final float sceneTouchEventY,
    final ITouchArea touchArea)
boolean onChildSceneTouchEvent(final TouchEvent pSceneTouchEvent)
```

本书第 8 章讲到用户输入时，将详述触摸区域以及触摸事件的响应。现在读者只要知道，可以设置 Scene 对象的触摸区域，可以覆写 onXTouchEvent 方法以响应触摸事件就行了。

#### 4.1.18 特殊 Scene 类

有些 Scene 类的子类是为了构建特殊类型的场景，其构造器罗列如下：

---

⊖ 意味着只有该类所在包以及其子类可以存取。——译者注



```
CameraScene(final int pLayerCount)
CameraScene(final int pLayerCount, final Camera pCamera)
```

此种 Scene 是在一般 Scene 的基础上加入摄像头功能。

```
SplashScene(final Camera pCamera, final TextureRegion pTextureRegion)
SplashScene(final Camera pCamera, final TextureRegion pTextureRegion,
            final float pDuration, final float pScaleFrom, final float pScaleTo)
```

创建启动画面（简化第 2 章中用于创建启动画面的代码）。

```
MenuScene()
MenuScene(final IOnMenuItemClickListener pOnMenuItemClickListener)
MenuScene(final Camera pCamera)
MenuScene(final Camera pCamera, final IOnMenuItemClickListener
            pOnMenuItemClickListener)
```

创建类似第 3 章中所创建的 Scene 那样的 MenuScene 对象。

```
PopupScene(final Camera pCamera, final Scene pParentScene,
            final float pDurationSeconds)
PopupScene(final Camera pCamera, final Scene pParentScene,
            final float pDurationSeconds, final Runnable pRunnable)
TextPopupScene(final Camera pCamera, final Scene pParentScene,
                final Font pFont, final String pText, final float pDurationSeconds)
TextPopupScene(final Camera pCamera, final Scene pParentScene,
                final Font pFont, final String pText, final float pDurationSeconds,
                final IEntityModifier pShapeModifier)
TextPopupScene(final Camera pCamera, final Scene pParentScene,
                final Font pFont, final String pText, final float pDurationSeconds,
                final Runnable pRunnable)
TextPopupScene(final Camera pCamera, final Scene pParentScene,
                final Font pFont, final String pText, final float pDurationSeconds,
                final IEntityModifier pShapeModifier, final Runnable pRunnable)
```

用于创建各种可在当前 Scene 上临时弹出的 Scene 对象。

#### 4.1.19 用于 Entity 的 Modifier 类

上一节中提到，Entity 对象上面可以附加 Modifier 对象以控制其参数，诸如位置、缩放因子、颜色、旋转角度、透明度等。本节将研究 Modifier 类本身，介绍各种可用的 Modifier 类型，并讲述如何将 Modifier 对象组合成序列来使用。

Entity 所附加的 Modifier 都是 EntityModifier 类的子类，其创建完成之后，通过前述的 registerEntityModifier() 方法注册到所要应用效果的 Entity 对象上。典型的用法如下：

```
entity.registerEntityModifier(new XxxModifier(p1, p2, ...));
```

在上述语句中, entity 是已经定义好的 Entity 对象, XxxModifier 是接下来将要列出的若干个 Modifier 类型中的一种。如果 Modifier 的构造需要一个指示作用时间段的参数, 那么通常是构造器的第一个参数, 其余的参数根据具体的 Modifier 类型而有所不同。

#### 4.1.20 EntityModifier 类的通用方法

有些 EntityModifier 类的通用方法非常有用。

```
XxxModifier clone()
```

返回此 Modifier 对象的拷贝。该方法可以在开发者需要有两个行为相同的 Modifier 对象时使用。每个 Modifier 对象都会在有效时间段内执行自己的动画序列, 如果开发者不想让 Entity 对象的动画一直播放。比如:

```
boolean isFinished()
```

如果 Modifier 完成了自己的动作, 则返回 true。

```
void setRemoveWhenFinished(final boolean pRemoveWhenFinished)
boolean isRemoveWhenFinished()
```

设置或者返回一个标志位, 代表该 Modifier 是否会在动作完成时被自动移除。

```
void setModifierListener(final IModifierListener<T> pModifierListener)
IModifierListener<T> getModifierListener()
```

设置或取得一个监听器对象, 它会在本 Modifier 对象完成动作之后被回调。只可以注册一个这样的监听器。

#### 4.1.21 位置相关的 EntityModifier 类

下列数种 Modifier 会改变 Entity 对象的当前位置, 它们不是某个类的方法, 而是 EntityModifier 类的子类, 用来被实例化之后, 通过前述代码模式<sup>⊖</sup>设定到 Entity 对象之上。

```
MoveModifier(final float pDuration, final float pFromX, final float pToX,
             final float pFromY, final float pToY, final IEntityModifierListener
             pEntityModifierListener, final IEaseFunction pEaseFunction)
MoveModifier(final float pDuration, final float pFromX,
             final float pToX,
```

⊖ 指的是 4.1.19 节中所提到的用法: “entity.registerEntityModifier(new XxxModifier(p1, p2, ...));” ——译者注

```

    final float pFromY, final float pToY, final IEntityModifierListener
    pEntityModifierListener, final IEaseFunction pEaseFunction)
    MoveModifier(final float pDuration, final float pFromX, final float pToX,
    final float pFromY, final float pToY, final IEntityModifierListener
    pEntityModifierListener, final IEaseFunction pEaseFunction)
    MoveModifier(final float pDuration, final float pFromX, final float pToX,
    final float pFromY, final float pToY, final IEntityModifierListener
    pEntityModifierListener, final IEaseFunction pEaseFunction)

```

上述构造器所创建之 Modifier 对象，都会使 Entity 对象发生横、纵坐标均改变的移动。pDuration 是移动动作所持续的秒数。本章稍后将会讲述 EaseFunction 类，这里只需要知道它是对 Modifier 对象的一个修正值即可。pEntityModifierListener 是一个 Modifier 完成其动作后将会回调的监听器。

```

    MoveXModifier(final float pDuration, final float pFromX, final float pToX)
    MoveXModifier(final float pDuration, final float pFromX, final float pToX, final
    IEaseFunction pEaseFunction)
    MoveXModifier(final float pDuration, final float pFromX, final float pToX, final
    IEntityModifierListener pEntityModifierListener)
    MoveXModifier(final float pDuration, final float pFromX, final float pToX, final
    IEntityModifierListener pEntityModifierListener, final IEaseFunction
    pEaseFunction)
    MoveXModifier(final MoveXModifier pMoveXModifier)
    MoveYModifier(final float pDuration, final float pFromY, final float pToY)
    MoveYModifier(final float pDuration, final float pFromY, final float pToY, final
    IEaseFunction pEaseFunction)
    MoveYModifier(final float pDuration, final float pFromY, final float pToY, final
    IEntityModifierListener pEntityModifierListener)
    MoveYModifier(final float pDuration, final float pFromY, final float pToY, final
    IEntityModifierListener pEntityModifierListener, final IEaseFunction
    pEaseFunction)
    MoveYModifier(final MoveYModifier pMoveYModifier)

```

顾名思义，以上的 Modifier 类对象只会使 Entity 对象在横向或者竖向一个方向进行运动。

### 路径相关的 EntityModifier 类

能从一点移动到另一点是很不错的，不过 AndEngine 还支持指定多点路径，让 Entity 对象沿着定义好的 Path 路径对象移动。此外，还可以注册监听器，以便在到达每个路径点时获得回调通知。

Path 类有三个构造器：

```

    Path(final int pLength)
    Path(final float[] pCoordinatesX, final float[] pCoordinatesY)
    Path(final Path pPath)

```

第一个构造器创建了一个包含 pLength 个点的路径，实际的路径点将通过以下方法逐个加入：

```
to(final float pX, final float pY)
```

第二个构造器使用横、纵坐标表示的数组来创建 Path 对象，第三个使用另外一个 Path 来创建新的 Path 对象。

PathModifier 类的构造器如下：

```
PathModifier(final float pDuration, final Path pPath)
PathModifier(final float pDuration, final Path pPath,
    final IEaseFunction pEaseFunction)
PathModifier(final float pDuration, final Path pPath,
    final IEntityModifierListener pEntityModifierListener)
PathModifier(final float pDuration, final Path pPath,
    final IEntityModifierListener pEntityModifierListener,
    final IEaseFunction pEaseFunction)
PathModifier(final float pDuration, final Path pPath,
    final IEntityModifierListener pEntityModifierListener,
    final IPathModifierListener pPathModifierListener)
PathModifier(final float pDuration, final Path pPath,
    final IEntityModifierListener pEntityModifierListener,
    final IPathModifierListener pPathModifierListener,
    final IEaseFunction pEaseFunction)
```

程序清单 4.1 展示了一个简短的范例代码，创建 Path 对象并传给 PathModifier 构造器，同时在其中注册一个回调监听器。

程序清单 4.1 PathModifier 范例代码

```
...
final Path path = new Path(5).to(10, 10).to(10, 50).to(50, 50).
    to(50, 10).to(10, 10);
entity.registerEntityModifier(new LoopEntityModifier(new PathModifier(30,
    path, null, new IPathModifierListener() {
    @Override
    public void onWaypointPassed(final PathModifier pPathModifier,
    final IEntity pEntity, final int pWaypointIndex) {
        switch(pWaypointIndex) {
            case 0:
                entity.setColor(1.0f, 0.0f, 0.0f);
                break;
            case 1:
                entity.setColor(0.0f, 1.0f, 0.0f);
                break;
            case 2:
```

```

        entity.setColor(0.0f, 0.0f, 1.0f);
        break;
    case 3:
        entity.setColor(1.0f, 0.0f, 0.0f);
        break;
    }
}
});
scene.getChildAt().attachChild(entity);

```

#### 4.1.22 缩放相关的 EntityModifier 类

ScaleModifier 修改 Entity 对象的缩放因子。以下这组构造器与 MoveModifier 的很像：

```

ScaleModifier(final float pDuration, final float pFromScale, final float pToScale)
ScaleModifier(final float pDuration, final float pFromScale, final float pToScale,
    final IEaseFunction pEaseFunction)
ScaleModifier(final float pDuration, final float pFromScale, final float pToScale,
    final IEntityModifierListener pEntityModifierListener)
ScaleModifier(final float pDuration, final float pFromScale, final float pToScale,
    final IEntityModifierListener pEntityModifierListener, final
    IEaseFunction pEaseFunction)

```

以上所有的构造器创建出来的 Modifier 对象，都会使 Entity 在一定的时间段内改变尺寸。构造器的参数都很直白，缩放因子 1.0f 代表和原有大小相同。

```

ScaleModifier(final float pDuration, final float pFromScaleX, final float
    pToScaleX, final float pFromScaleY, final float pToScaleY)

```

此种 Modifier 对象会使 Entity 对象在横向和纵向上立即按照不同的缩放因子改变其大小。

```

ScaleAtModifier(final float pDuration, final float pFromScale, final
    float pToScale, final float pScaleCenterX, final float pScaleCenterY)
ScaleAtModifier(final float pDuration, final float pFromScale, final
    float pToScale, final float pScaleCenterX, final float pScaleCenterY,
    final IEaseFunction pEaseFunction)
ScaleAtModifier(final float pDuration, final float pFromScale, final
    float pToScale, final float pScaleCenterX, final float pScaleCenterY,
    final IEntityModifierListener pEntityModifierListener)
ScaleAtModifier(final float pDuration, final float pFromScale, final
    float pToScale, final float pScaleCenterX, final float pScaleCenterY,
    final IEntityModifierListener pEntityModifierListener,
    final IEaseFunction pEaseFunction)

```



```

ScaleAtModifier(final float pDuration, final float pFromScaleX, final
    float pToScaleX, final float pFromScaleY, final float pToScaleY,
    final float pScaleCenterX, final float pScaleCenterY)
ScaleAtModifier(final float pDuration, final float pFromScaleX, final
    float pToScaleX, final float pFromScaleY, final float pToScaleY,
    final float pScaleCenterX, final float pScaleCenterY,
    final IEaseFunction pEaseFunction)
ScaleAtModifier(final float pDuration, final float pFromScaleX, final
    float pToScaleX, final float pFromScaleY, final float pToScaleY,
    final float pScaleCenterX, final float pScaleCenterY,
    final IEntityModifierListener pEntityModifierListener)
ScaleAtModifier(final float pDuration, final float pFromScaleX, final
    float pToScaleX, final float pFromScaleY, final float pToScaleY,
    final float pScaleCenterX, final float pScaleCenterY,
    final IEntityModifierListener pEntityModifierListener,
    final IEaseFunction pEaseFunction)

```

以上这些构造器所创建的 Modifier 同前述的 ScaleModifier 相似, 只是它缩放时所用的中心点不是 Entity 对象所固有的那个, 而是用构造器传入的那个中心点。

#### 4.1.23 颜色相关的 EntityModifier 类

正如大家所想, ColorModifier 将修改 Entity 对象的颜色, 所有颜色值的范围都是从 0.0f (无颜色饱和度) 到 1.0f (颜色饱和度最大)。

```

ColorModifier(final float pDuration, final float pFromRed, final float pToRed,
    final float pFromGreen, final float pToGreen, final float pFromBlue,
    final float pToBlue)
ColorModifier(final float pDuration, final float pFromRed, final float pToRed,
    final float pFromGreen, final float pToGreen, final float pFromBlue,
    final float pToBlue, final IEaseFunction pEaseFunction)
ColorModifier(final float pDuration, final float pFromRed, final float pToRed,
    final float pFromGreen, final float pToGreen, final float pFromBlue,
    final float pToBlue, final IEntityModifierListener
    pEntityModifierListener, final IEaseFunction pEaseFunction)
ColorModifier(final float pDuration, final float pFromRed, final float pToRed,
    final float pFromGreen, final float pToGreen, final float pFromBlue,
    final float pToBlue, final IEntityModifierListener
    pEntityModifierListener, final IEaseFunction pEaseFunction)

```

以上构造器所创建的 Modifier 对象会在一定的时间段内改变 Entity 的颜色。EaseFunction 函数和 IEntityModifierListener 函数的意思与其他类型的 EntityModifier 构造器相同。

#### 4.1.24 旋转相关的 EntityModifier 类

RotationModifier 对象将会使 Entity 对象围绕某点进行旋转。这个点可以是 Entity



对象自身定义的旋转中心点，也可以是构造 Modifier 时传入的，所有旋转操作的单位都是角度。

```
RotationModifier(final float pDuration, final float pFromRotation,
    final float pToRotation)
RotationModifier(final float pDuration, final float pFromRotation,
    final float pToRotation, final IEntityModifierListener pEntityModifierListener)
RotationModifier(final float pDuration, final float pFromRotation,
    final float pToRotation, final IEaseFunction pEaseFunction)
RotationModifier(final float pDuration, final float pFromRotation,
    final float pToRotation, final IEntityModifierListener pEntityModifierListener,
    final IEaseFunction pEaseFunction)
```

以上构造器所创建的 Modifier 对象将使用 Entity 对象自身定义的旋转中心点进行旋转。

```
RotationAtModifier(final float pDuration, final float pFromRotation,
    final float pToRotation, final float pRotationCenterX,
    final float pRotationCenterY)
RotationAtModifier(final float pDuration, final float pFromRotation,
    final float pToRotation, final float pRotationCenterX,
    final float pRotationCenterY, final IEntityModifierListener
    pEntityModifierListener)
RotationAtModifier(final float pDuration, final float pFromRotation,
    final float pToRotation, final float pRotationCenterX,
    final float pRotationCenterY, final IEaseFunction pEaseFunction)
RotationAtModifier(final float pDuration, final float pFromRotation,
    final float pToRotation, final float pRotationCenterX,
    final float pRotationCenterY, final IEntityModifierListener
    pEntityModifierListener, final IEaseFunction pEaseFunction)
```

这些构造器所创建的 Modifier 对象会在给定的时间段内围绕指定的点进行旋转，而不使用 Entity 对象所固有的旋转中心点。这种方法避免了在旋转前手工修改旋转中心点，而且开发者一般也不愿意去那么做。

```
RotationByModifier(final float pDuration, final float pRotation)
RotationByModifier(final RotationByModifier pRotationByModifier)
```

以上构造器所创建的 Modifier 对象将会使 Entity 立即围绕其旋转中心点转动。开发者也可以用 0.0f 作为 pDuration 的参数值构造 RotationModifier 来达到此效果，不过这么做更简洁高效。

#### 4.1.25 透明度相关的 EntityModifier 类

Entity 对象的透明度通过 Alpha 值来控制。以下构造器所创建的 Modifier 可以用来制作 Entity 的淡入或淡出效果。

```

AlphaModifier(final float pDuration, final float pFromAlpha, final float
    pToAlpha)
AlphaModifier(final float pDuration, final float pFromAlpha, final float
    pToAlpha, final IEntityModifierListener)
AlphaModifier(final float pDuration, final float pFromAlpha, final float
    pToAlpha, final IEaseFunction pEaseFunction)
AlphaModifier(final float pDuration, final float pFromAlpha, final float
    pToAlpha, final IEntityModifierListener pEntityModifierListener,
    final IEaseFunction pEaseFunction)

```

以上构造器的参数都很直白，Alpha 值的范围可以从 0.0f（不可见，即完全透明）到 1.0f（完全不透明）。

```

FadeInModifier(final float pDuration)
FadeInModifier(final float pDuration, final IEaseFunction pEaseFunction)
FadeInModifier(final float pDuration, final IEntityModifierListener
    pEntityModifierListener)
FadeInModifier(final float pDuration, final IEntityModifierListener
    pEntityModifierListener, final IEaseFunction pEaseFunction)

```

以上构造器所创建的 Modifier 对象可以用来制作 Entity 对象的淡入、淡出效果。这些对象的内部包裹了一个 AlphaModifier 对象，并用 0.0f 和 1.0f 作为其透明度的起止值。

#### 4.1.26 延迟相关的 EntityModifier 类

读者可能对使用 DelayModifier 的原因感到困惑。在接下来一节中，本书将讨论如何将 Modifier 对象组合成序列，在这种情况下，延时功能的 Modifier 就显得非常必要了。

```

DelayModifier(final float pDuration)
DelayModifier(final float pDuration, final IEntityModifierListener
    pEntityModifierListener)

```

这些构造器所创建的 Modifier 类会延迟指定的时间，后者会在延迟动作完成后通知注册的监听器。

#### 4.1.27 Modifier 的组合

开发者经常希望组合一系列的 Modifier 并应用于 Entity 对象上。举例来说，开发者可能希望 Entity 对象在移动的同时改变缩放因子与颜色，或者先移动，再改变大小，最后变更颜色。通过特殊的 Modifier 对象，可以轻松构建出复杂的 Modifier 序列。程序清单 4.2 展示了使用其中一种特殊对象来组合 Modifier。

程序清单 4.2 Modifier 的组合使用

```

entity.registerEntityModifier(

```

```

new SequenceEntityModifier(
    new ParallelEntityModifier(
        new MoveYModifier(3, 0.0f, CAMERA_HEIGHT - 40.0f),
        new AlphaModifier(3, 0.0f, 1.0f),
        new ScaleModifier(3, 0.5f, 1.0f)
    ),
    new RotationModifier(3, 0, 720)
);

```

所有用于组合的 Modifier，其构造器在需要其他 Modifier 做参数时，如果接受到非 Modifier 的对象，都会抛出 `IllegalArgumentException` 异常。

### ParallelEntityModifier

当需要为某个 Entity 对象应用两个或更多个 Modifier 时，应该用如下构造器创建 `ParallelEntityModifier` 对象：

```

ParallelEntityModifier(final IEntityModifier... pEntityModifiers)
ParallelEntityModifier(final IEntityModifierListener pEntityModifierListener,
    final IEntityModifier... pEntityModifiers)

```

这样组合出来的 `ParallelEntityModifier`，将在运行时并行地执行作为参数传入的 Modifier 对象的动作。当最后一个 Modifier 对象结束其动作后，`pEntityModifiers` 监听器会被回调。

### SequenceEntityModifier

当需要为某个 Entity 对象按顺序地应用多个 Modifier 时，应该使用如下构造器创建 `SequenceEntityModifier` 对象：

```

SequenceEntityModifier(final IEntityModifier... pEntityModifiers)
SequenceEntityModifier(final IEntityModifierListener pEntityModifierListener,
    final IEntityModifier... pEntityModifiers)
SequenceEntityModifier(final IEntityModifierListener pEntityModifierListener,
    final ISubSequenceShapeModifierListener
    pSubSequenceShapeModifierListener,
    final IEntityModifier... pEntityModifiers)

```

前两个构造器与 `ParallelEntityModifier` 的相仿。最后一个有些不同，它允许在每个 Entity 序列中的对象执行完其动作之后通过 `ISubSequenceShapeModifierListener` 监听器接口得到回调通知<sup>①</sup>。

① 此处原书文字为“it allows the creation of SubSequence”，意为“允许创建 SubSequence 对象”。经与 AndEngine 源代码核对后，确定原书此处描述有错误，故翻译时做出如上订正，特此注出。——译者注

### 4.1.28 EaseFunction

现在来看看 Modifier 构造器里面总是提到的 EaseFunction 类是什么意思。本节将介绍它的功能，以及所提供 EaseFunction 的种类。在 AndEngine 网站上 (<http://www.andengine.org>) 有一个使用 EaseFunction 的例子包含在 AndEngine 范例程序里面，如果读者还没有安装这个范例程序，现在请下载并安装，打开 EaseFunction 演示程序感受一下它的直观效果。

EaseFunction 对象改变了 Modifier 对象执行其动作的过程，在默认情况下（即不使用 EaseFunction 时），Modifier 将会线性地执行其动作。举例来说，如果用 MoveModifier 对象将 Entity 对象从 A 点移动到 B 点，那么其移动速度在运动的开始、中间及结束处，都是相同的。

EaseFunction 对象允许开发者在 Modifier 对象执行其动作的时间段内，在不同的小阶段中运用不同的速率去执行（甚至是后退执行）。想象一下高层酒店的电梯，它在不同楼层之间飞速穿梭，然而当快到目标楼层时，它会缓缓减速，使得乘客不会明显地感到电梯速度骤减。

EaseFunction 类的命名规范是：

Ease< 类型 >< 端点 >。

其中 < 类型 > 指的是缓动函数（ease function）的类型（例如向后、跳跃、弧线、三次曲线、二次曲线等），< 端点 > 指的是函数将影响动作的哪个阶段。“In” 函数影响时间段的初始阶段，“Out” 函数影响结束阶段，“InOut” 函数则两个阶段都影响。接下来依次看看每种类型的缓动效果。

在下文的描述中，用图指示每种类型的 “InOut” 效果<sup>⊖</sup>。

#### EaseBack 缓动函数

EaseBack 函数在动作开始时先从起点向后退，再向前移动，结束时先超越终点，再倒回至终点。其效果如图 4.1 所示。

```
EaseBackIn
EaseBackOut
EaseBackInOut
```

#### EaseBounce 缓动函数

EaseBounce 函数在动作开始和结束时会产生跳跃效果，如图 4.2 所示。该函数的一个常见用法是模拟某物落地反弹的运动（如果不使用在第 12 章描述的物理学扩展包的话）。此类函数有三个：

⊖ 图表中横轴为应用该缓动函数的时间，纵轴为经过缓动函数修正值后的 Modifier 相关行为数值，例如对于 MoveModifier 来说，这个值就是在该时间点所走过的路程。——译者注

EaseBounceIn  
EaseBounceOut  
EaseBounceInOut

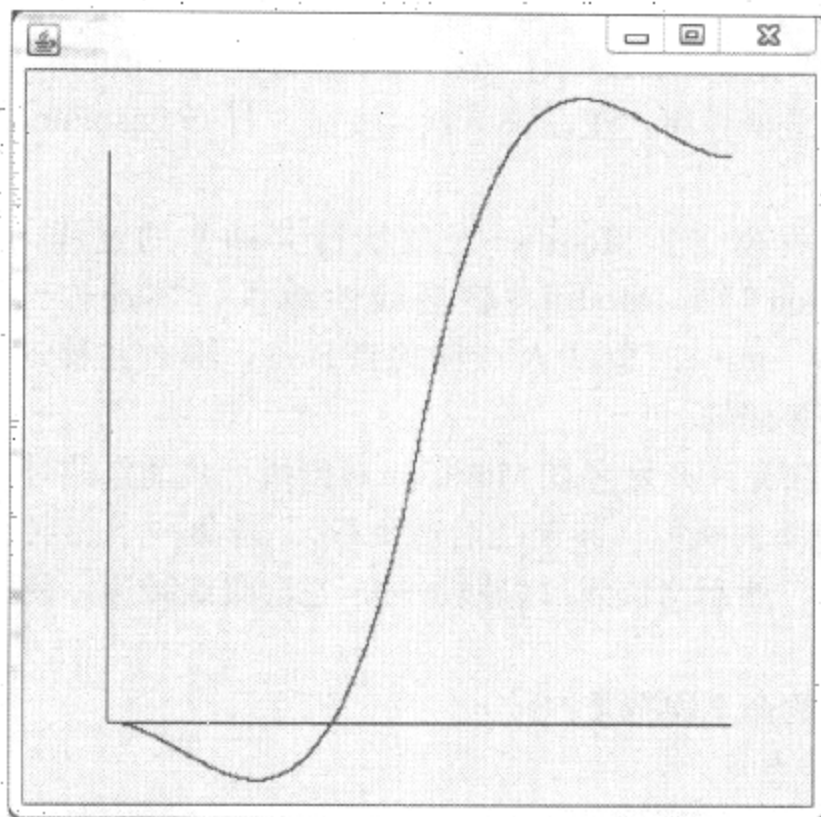


图 4.1 EaseBackInOut 效果图

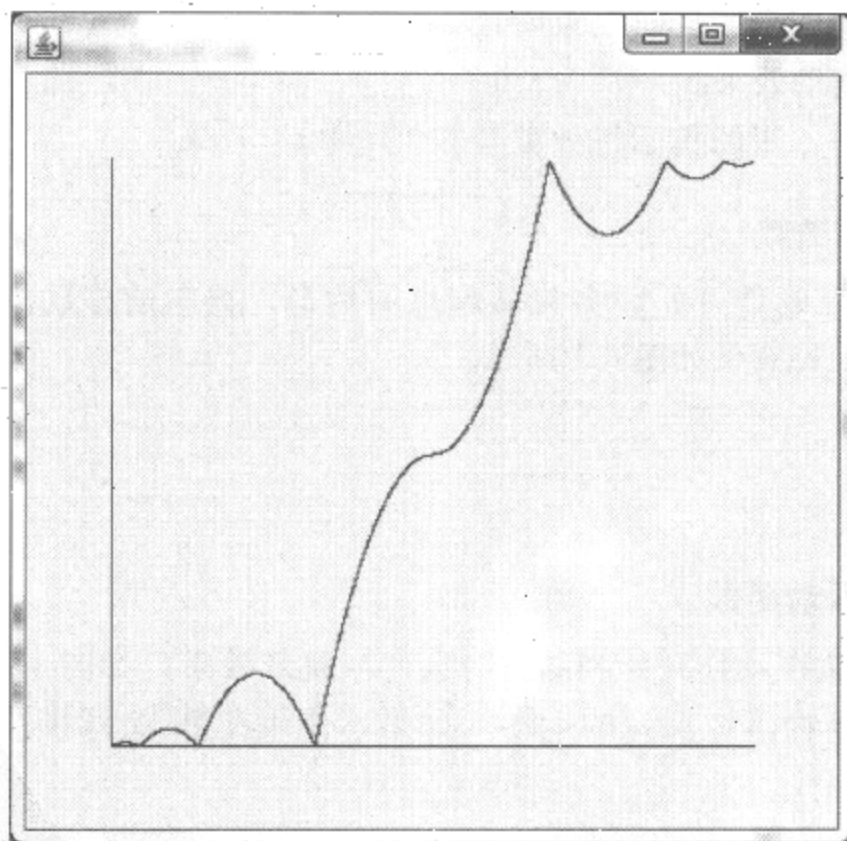


图 4.2 EaseBounceInOut 效果图



### EaseCircular 缓动函数

EaseCircular 函数在动作开始和结束时会将数值的改变放缓，做出圆弧加速度的效果，如图 4.3 所示。

```
EaseCircularIn  
EaseCircularOut  
EaseCircularInOut
```

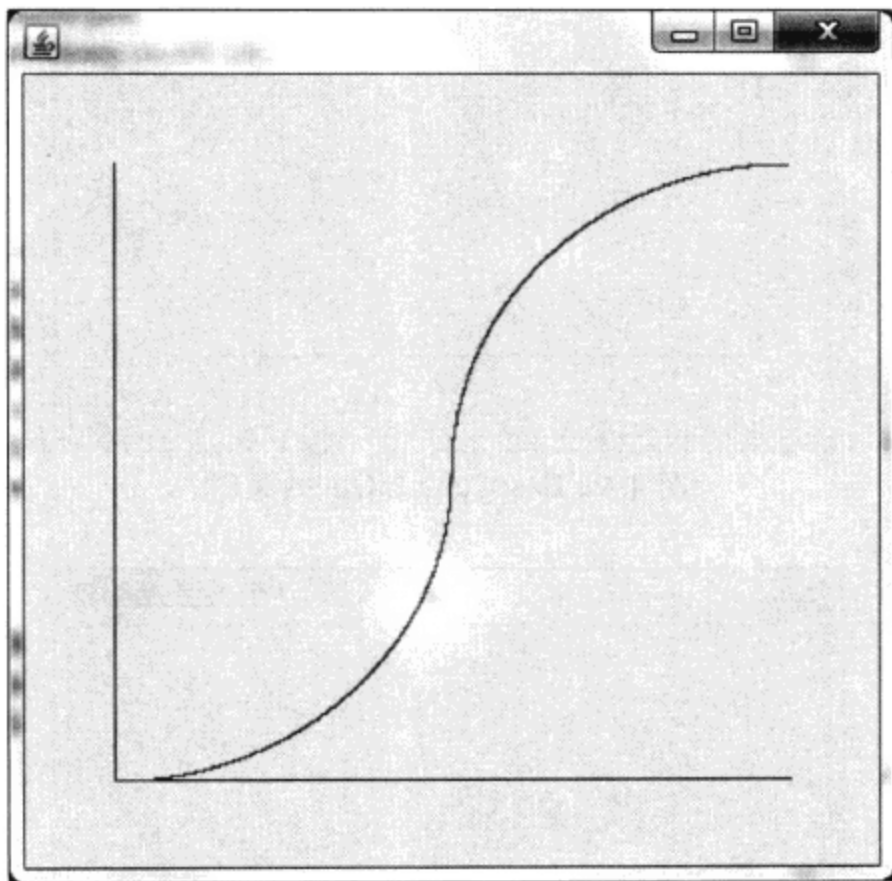


图 4.3 EaseCircularInOut 效果图

### EaseCubic 缓动函数

EaseCubic 函数在动作开始和结束时也会利用三次方程，将数值的改变放缓。如果开发者要模拟前述的电梯减速移动效果，那么可以使用以下的缓动函数：

```
EaseCubicIn  
EaseCubicOut  
EaseCubicInOut
```

### EaseElastic 缓动函数

EaseElastic 函数模拟橡皮筋或弹簧的效果，如图 4.5 所示。

```
EaseElasticIn  
EaseElasticOut  
EaseElasticInOut
```



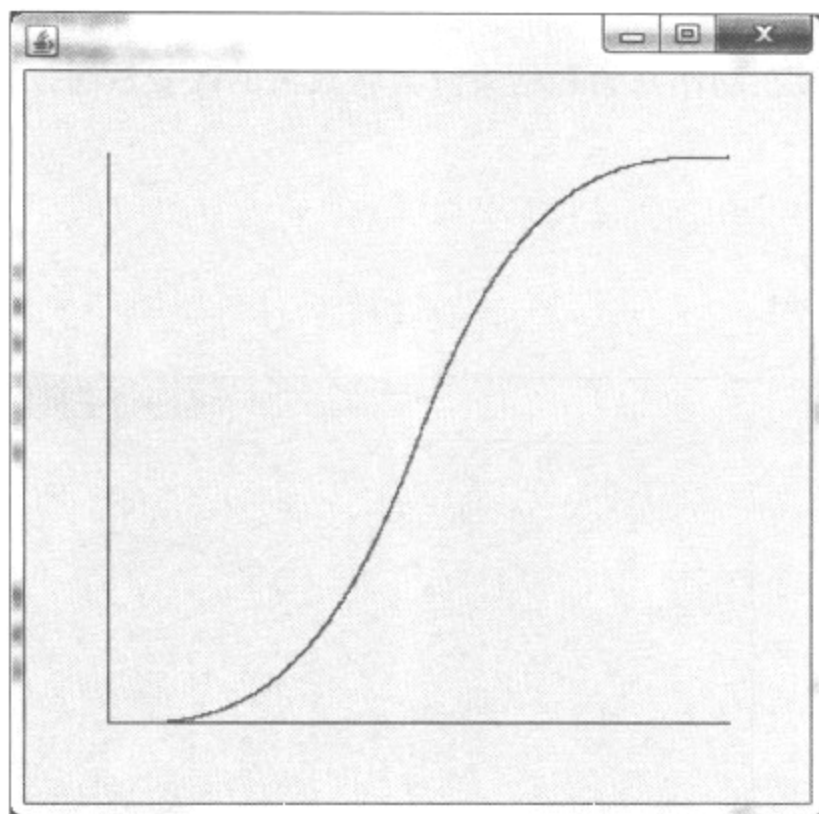


图 4.4 EaseCubicInOut 效果图

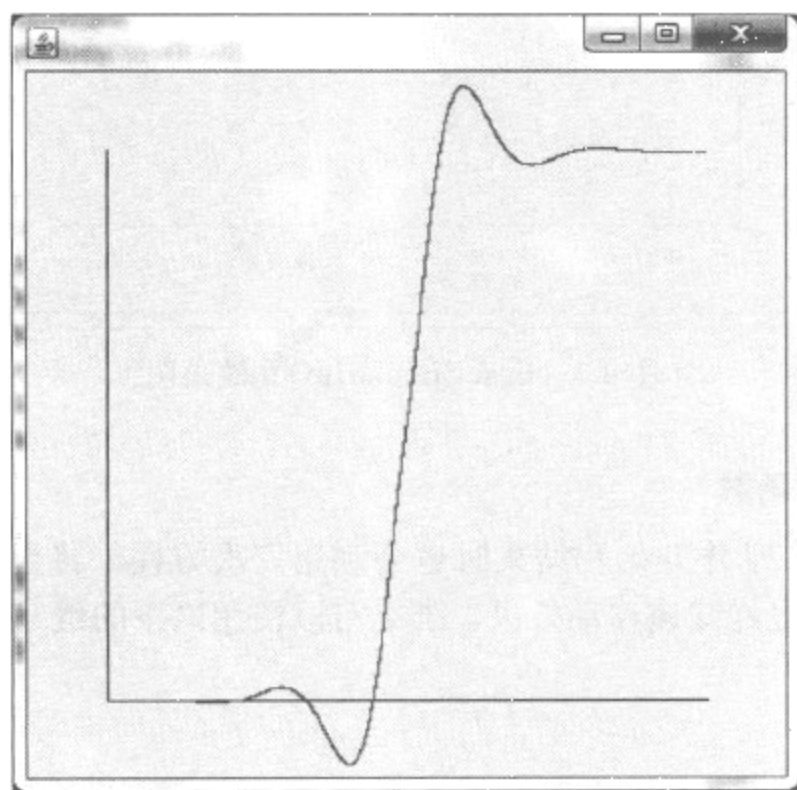


图 4.5 EaseElasticInOut 效果图

### EaseExponential 缓动函数

EaseExponential 函数在动作的开始和结束时，会使数值的变动产生指数曲线般的效果，如图 4.6 所示。

EaseExponentialIn  
EaseExponentialOut  
EaseExponentialInOut

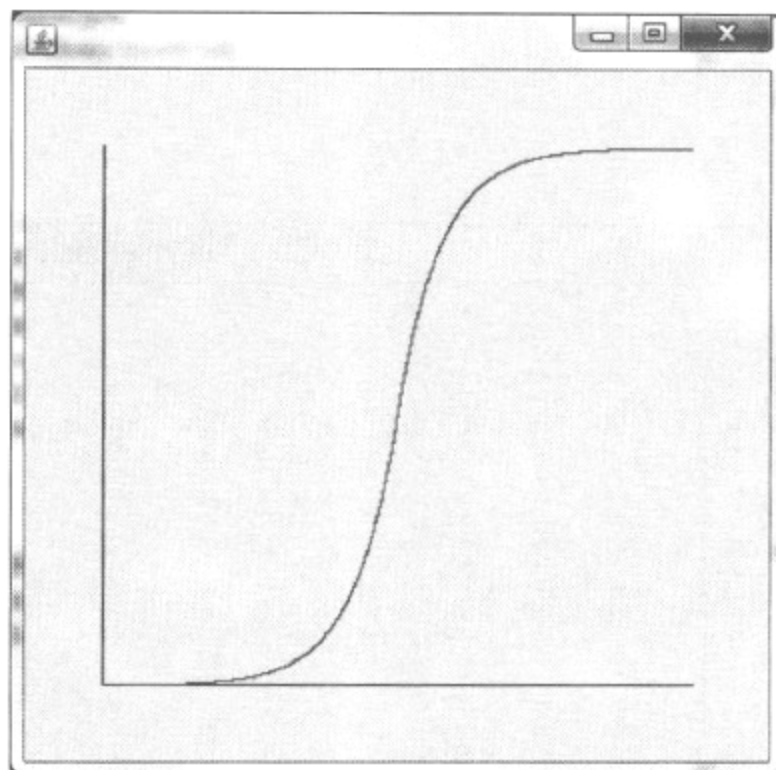


图 4.6 EaseExponentialInOut 效果图

### EaseLinear 缓动函数

的确存在这样一个 EaseLinear 函数，其效果就相当于默认的动作（即不使用 EaseFunction），如图 4.7 所示。它在组合 EaseFunction 序列时很有用。

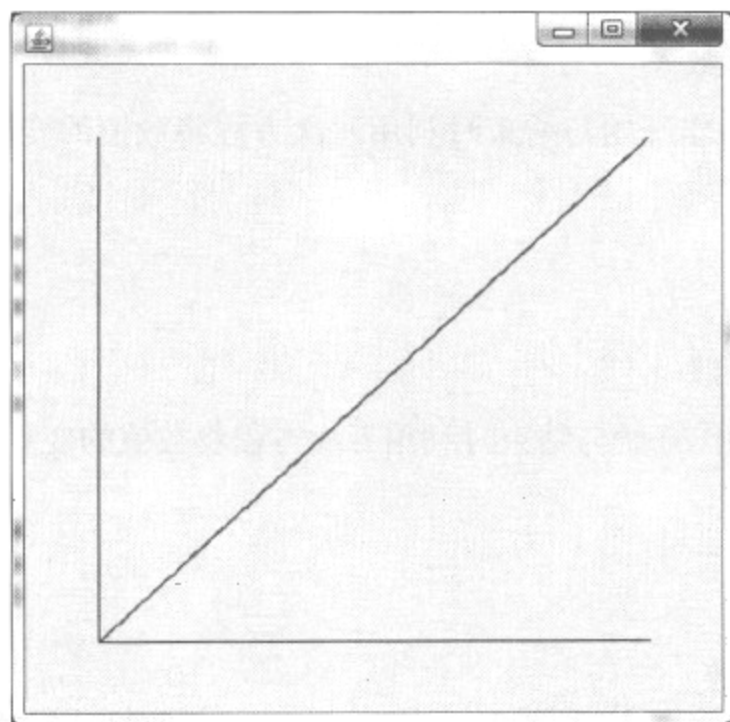


图 4.7 EaseLinear 效果图

### EaseQuad 缓动函数

EaseQuad 函数在动作开始与结束时利用二次方程将数值的变动放缓，如图 4.8 所示。

EaseQuadIn  
EaseQuadOut  
EaseQuadInOut

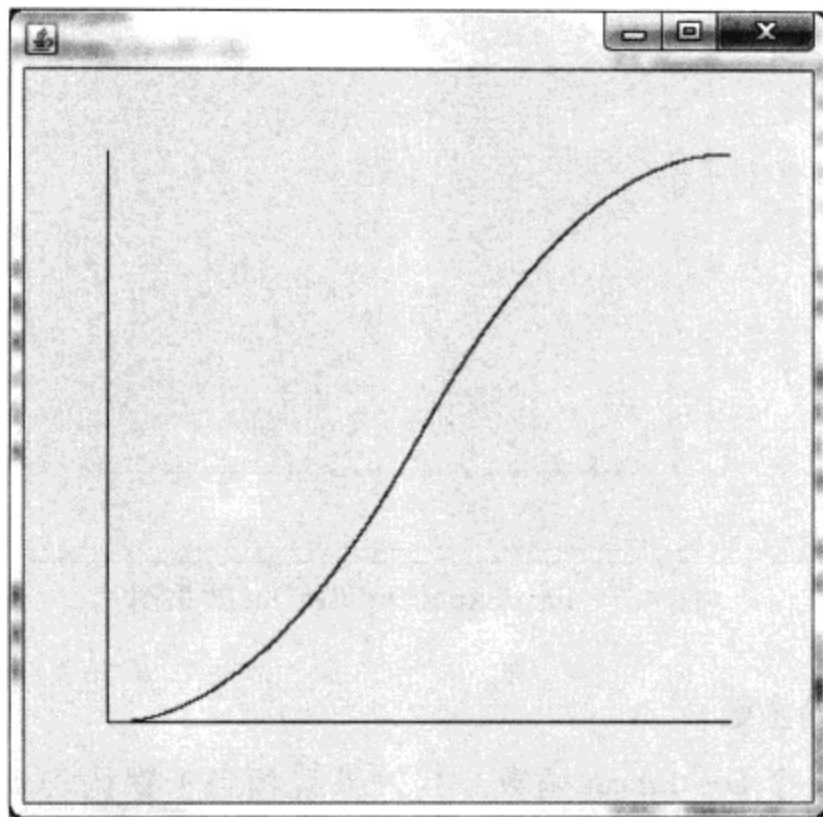


图 4.8 EaseQuadInOut 效果图

### EaseQuart 缓动函数

EaseQuart 函数在动作开始与结束时利用四次方程将数值的变动放缓，如图 4.9 所示。

EaseQuartIn  
EaseQuartOut  
EaseQuartInOut

### EaseQuint 缓动函数

EaseQuint 函数在动作开始与结束时利用五次方程将数值的变动放缓，如图 4.10 所示。

EaseQuintIn  
EaseQuintOut  
EaseQuintInOut

### EaseSine 缓动函数

EaseSine 函数在动作开始与结束时利用正弦函数将数值的变动放缓，如图 4.11 所示。

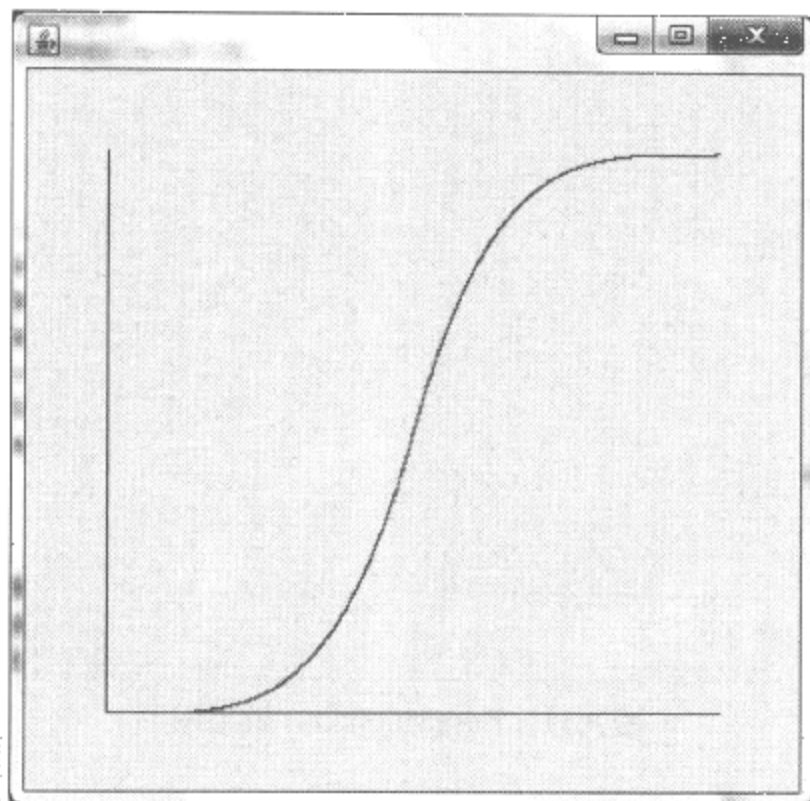


图 4.9 EaseQuartInOut 效果图

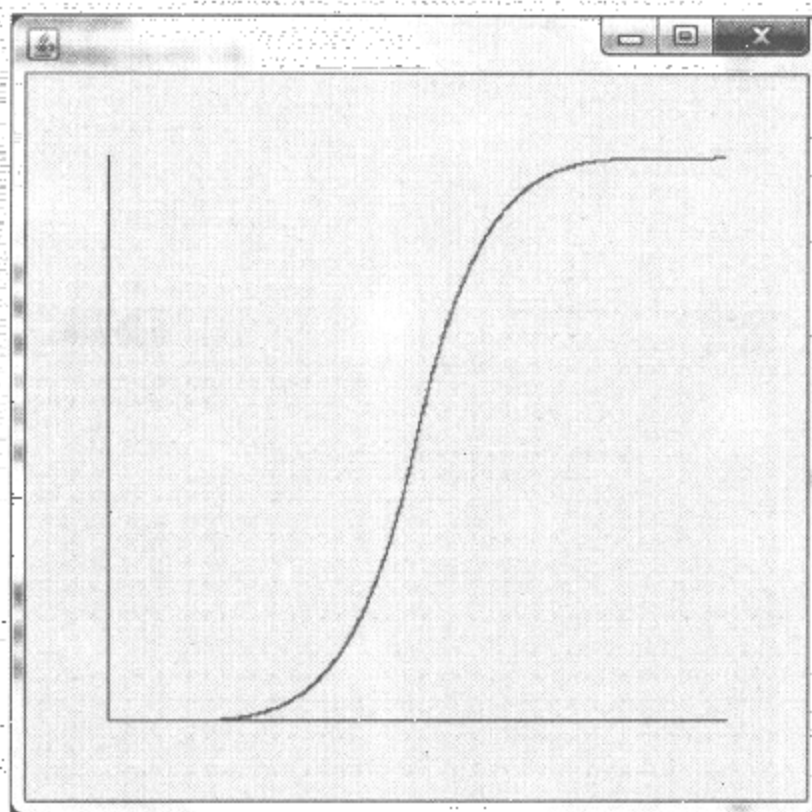


图 4.10 EaseQuintInOut 效果图

EaseSineIn  
EaseSineOut  
EaseSineInOut

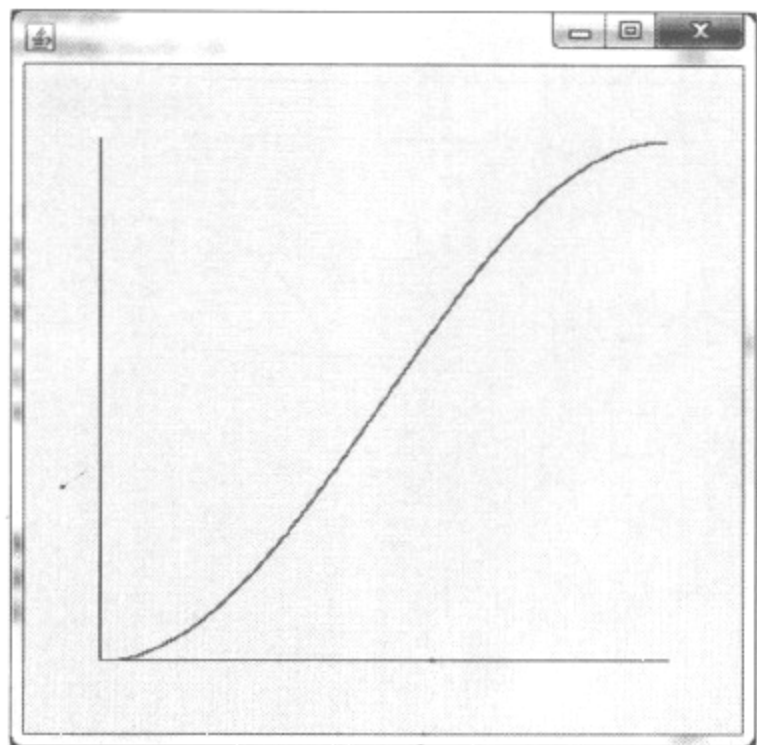


图 4.11 EaseSineInOut 效果图

### EaseStrong 缓动函数

尽管该函数作为一种缓动类型独立实现，但从数学角度来看，它与 EaseQuint 函数是一样的，如图 4.12 所示。

```
EaseStrongIn  
EaseStrongOut  
EaseStrongInOut
```

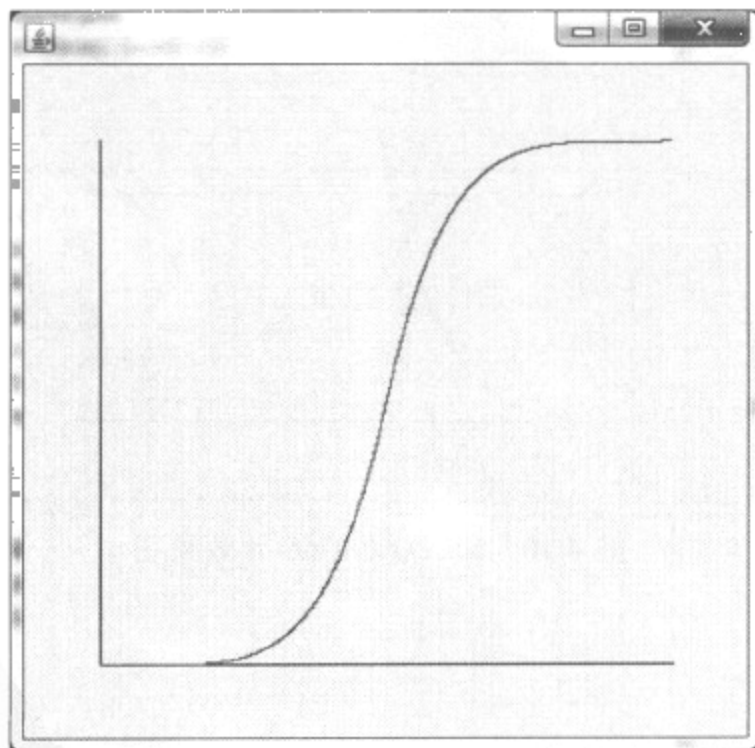


图 4.12 EaseStrongInOut 效果图

## 4.2 创建游戏第1关的场景

上述内容已经详尽地讲述了场景、图层、切换、修改器与缓动函数等知识，现在应该运用一点儿学到的知识来继续制作游戏了。当玩家选择菜单中的“Play Game”选项时，V3 游戏将会显示如图 4.13 所示场景。该场景描绘了住有几位倔强女孩的 Blossom 小姐家旁边的一片墓地。

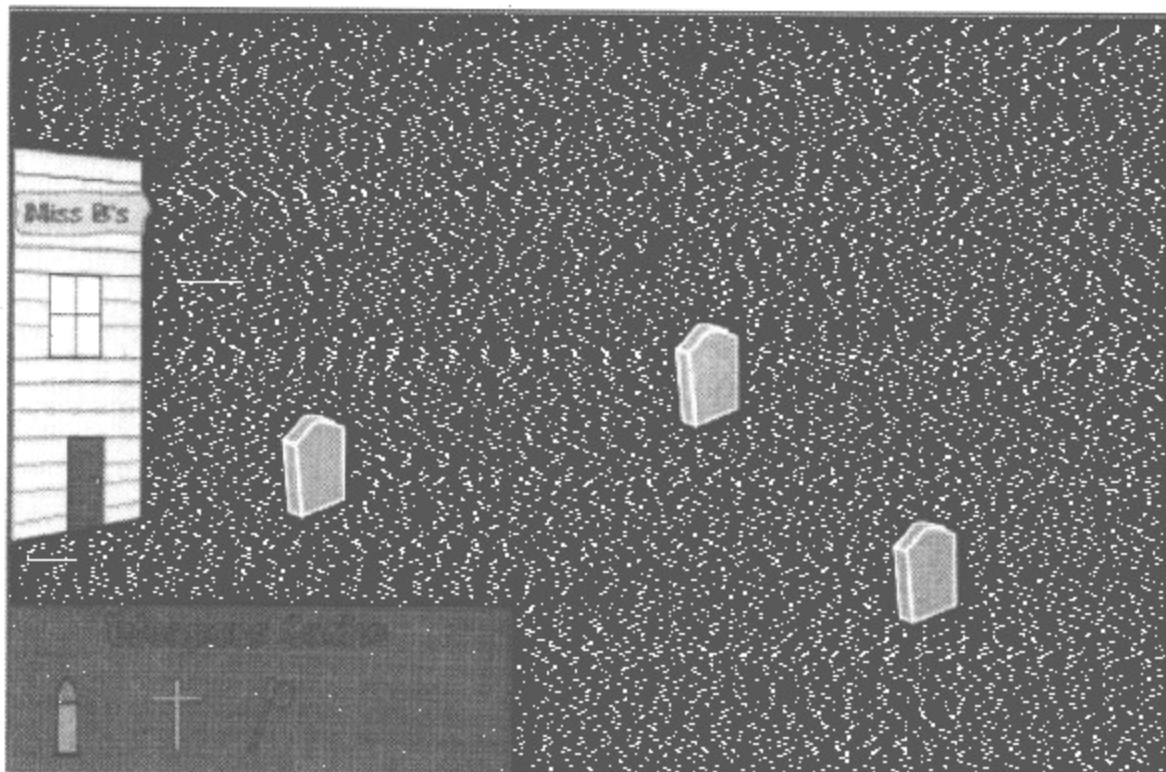


图 4.13 V3 初始游戏场景

该场景以后要加入触摸事件以支持将场景左下角的障碍物拖放到墓地之中。现在不需要实现触摸处理部分，只需要做出从菜单用户界面到该场景的切换即可。

程序清单 4.3 列出了对 MainMenuActivity.java 文件的修改，程序清单 4.4 列出了 Level1Activity.java 文件，此类实现了游戏的第 1 关。从现在开始，当玩家按下“Play Game”按钮后，菜单将渐渐缩小并远去，第 1 关的场景将逐渐浮现至眼前。每种武器将从屏幕上方落入武器库，并在就位后自转一周。

此时的代码读起来应该非常简单。这段代码依然按惯例创建了场景与图层对象，然后将新的子元素加入到图层之中。武器的运动效果是由精心安排的 Modifier 序列做出来的：在 MoveModifier 对象上应用 EaseQuadOut 缓动函数，这样武器会很快落下，然后在接近武器库时放缓速度。

程序清单 4.3 MainMenuActivity.java

```
protected Handler mHandler;
```



```

    . . .
    @Override
    public Engine onLoadEngine() {
        mHandler = new Handler();
    . . .
    @Override
    public void onResumeGame() {
        super.onResumeGame();
        mMainScene.registerEntityModifier(new ScaleAtModifier(0.5f,
            0.0f, 1.0f, CAMERA_WIDTH/2, CAMERA_HEIGHT/2));
        mStaticMenuScene.registerEntityModifier(
            new ScaleAtModifier(0.5f, 0.0f, 1.0f,
                CAMERA_WIDTH/2, CAMERA_HEIGHT/2));
    }
    . . .
    @Override
    public boolean onMenuItemClicked(final MenuScene pMenuScene,
        final IMenuItem pMenuItem, final float pMenuItemLocalX,
        final float pMenuItemLocalY) {
        switch(pMenuItem.getID()) {
    . . .
            case MENU_PLAY:
                mMainScene.registerEntityModifier(
                    new ScaleModifier(1.0f, 1.0f,
                        0.0f));
                mHandler.postDelayed(
                    mLaunchLevel1Task, 1000);
                return true;
            case MENU_SCORES:
    . . .
        private Runnable mLaunchLevel1Task = new Runnable() {
            public void run() {
                Intent myIntent = new Intent(MainMenuActivity.this,
                    Level1Activity.class);
                MainMenuActivity.this.startActivity(myIntent);
            }
        };
    . . .

```

在 switch 语句的 MENU\_PLAY 这个 case 分支上新增的两行语句促使我们又做出了如下修改：

- ❑ 增加了一个用于做出两个场景对象（mMainScene 与 mStaticMenuScene）缩放效果的 Modifier 对象，并在 1 秒钟内完成了两者的收缩。
- ❑ 为了使 Modifier 的效果能够完整地执行，请求 Android 系统在延迟 1 秒之后再启动 Level1Activity，实现此功能的机制与 StartActivity 所用的相同：将实际启

动 Activity 的 Intent 那个请求封装在 Runnable 对象中, 投递给系统。

- 覆写了 onResumeGame() 方法, 以便玩家在从 Level1Activity 切换回主菜单场景时, mMainScene 可以变回原来的大小。

程序清单 4.4 Level1Activity.java

```
package com.pearson.lagp.v3;
...
public class Level1Activity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;
    private String tag = "Level1Activity";

    // =====
    // 字段
    // =====

    protected Camera mCamera;

    protected Scene mMainScene;

    private Texture mLevel1BackTexture;
    private BuildableTexture mObstacleBoxTexture;
    private TextureRegion mBoxTextureRegion;
    private TextureRegion mLevel1BackTextureRegion;
    private TextureRegion mBulletTextureRegion;
    private TextureRegion mCrossTextureRegion;
    private TextureRegion mHatchetTextureRegion;

    // =====
    // 构造器
    // =====

    // =====
    // Getter 和 Setter
    // =====

    // =====
    // 覆写超类及接口中的方法
    // =====

    @Override
    public Engine onLoadEngine() {
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        return new Engine(new EngineOptions(true,
```

```

        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT),
        this.mCamera));
    }

    @Override
    public void onLoadResources() {
        /* 载入 Textures 对象。 */
        TextureRegionFactory.setAssetBasePath("gfx/Level1/");
        mLevel1BackTexture = new Texture(512, 512,
            TextureOptions.BILINEAR_PREMULTIPLYALPHA);
        mLevel1BackTextureRegion =
            TextureRegionFactory.createFromAsset(
                this.mLevel1BackTexture,
                this, "Level1Bk.png", 0, 0);
        mEngine.getTextureManager().loadTexture(
            this.mLevel1BackTexture);

        mObstacleBoxTexture = new BuildableTexture(512, 256,
            TextureOptions.BILINEAR_PREMULTIPLYALPHA);
        mBoxTextureRegion =
            TextureRegionFactory.createFromAsset(
                mObstacleBoxTexture,
                this, "Obstaclebox.png");
        mBulletTextureRegion =
            TextureRegionFactory.createFromAsset(
                mObstacleBoxTexture,
                this, "Bullet.png");
        mCrossTextureRegion =
            TextureRegionFactory.createFromAsset(
                mObstacleBoxTexture,
                this, "Cross.png");
        mHatchetTextureRegion =
            TextureRegionFactory.createFromAsset(
                mObstacleBoxTexture,
                this, "Hatchet.png");
        try {
            mObstacleBoxTexture.build(
                new BlackPawnTextureBuilder(2));
        } catch (final TextureSourcePackingException e) {
            Log.d(tag, "Sprites won't fit ");
        }
        this.mEngine.getTextureManager().loadTexture(
            this.mObstacleBoxTexture);
    }

    @Override
    public Scene onLoadScene() {
        this.mEngine.registerUpdateHandler(new FPSLogger());
    }

```

```

final Scene scene = new Scene(1);

/* 将摄像头置中。 */
final int centerX = (CAMERA_WIDTH -
    mLevel1BackTextureRegion.getWidth()) / 2;
final int centerY = (CAMERA_HEIGHT -
    mLevel1BackTextureRegion.getHeight()) / 2;

/* 创建 Sprite 对象并将其加入 Scene 对象中。 */
final Sprite background = new Sprite(centerX, centerY,
    mLevel1BackTextureRegion);
scene.getLastChild().attachChild(background);
final Sprite obstacleBox = new Sprite(0.0f, CAMERA_HEIGHT -
    mBoxTextureRegion.getHeight(), mBoxTextureRegion);
scene.getLastChild().attachChild(obstacleBox);
final Sprite bullet = new Sprite(20.0f,
    CAMERA_HEIGHT - 40.0f,
    mBulletTextureRegion);
bullet.registerEntityModifier(
    new SequenceEntityModifier(
        new ParallelEntityModifier(
            new MoveYModifier(3, 0.0f,
                CAMERA_HEIGHT - 40.0f,
                EaseQuadOut.getInstance()),
            new AlphaModifier(3, 0.0f, 1.0f),
            new ScaleModifier(3, 0.5f, 1.0f)
        ),
        new RotationModifier(3, 0, 360)
    )
);
scene.getLastChild().attachChild(bullet);
final Sprite cross = new Sprite(bullet.getInitialX() +
    40.0f, CAMERA_HEIGHT - 40.0f, mCrossTextureRegion);
cross.registerEntityModifier(
    new SequenceEntityModifier(
        new ParallelEntityModifier(
            new MoveYModifier(4, 0.0f,
                CAMERA_HEIGHT - 40.0f,
                EaseQuadOut.getInstance()),
            new AlphaModifier(4, 0.0f, 1.0f),
            new ScaleModifier(4, 0.5f, 1.0f)
        ),
        new RotationModifier(2, 0, 360)
    )
);
cross.registerEntityModifier(new AlphaModifier(
    10.0f, 0.0f, 1.0f));
scene.getLastChild().attachChild(cross);
final Sprite hatchet = new Sprite(cross.getInitialX() +
    40.0f,
    CAMERA_HEIGHT - 40.0f, mHatchetTextureRegion);

```

```

        hatchet.registerEntityModifier(
            new SequenceEntityModifier(
                new ParallelEntityModifier(
                    new MoveYModifier(5, 0.0f,
                        CAMERA_HEIGHT - 40.0f,
                        EaseQuadOut.getInstance() ),
                    new AlphaModifier(5, 0.0f, 1.0f),
                    new ScaleModifier(5, 0.5f, 1.0f)
                ),
                new RotationModifier(2, 0, 360)
            )
        );
        hatchet.registerEntityModifier(new AlphaModifier(
            15.0f, 0.0f, 1.0f));
        scene.getLastChild().attachChild(hatchet);
        scene.registerEntityModifier(new AlphaModifier(10, 0.0f,
            1.0f));
        return scene;
    }

    @Override
    public void onLoadComplete() {
    }
}

```

### 4.3 总结

本章讲述的内容很多，读者不可能一下子记住所有讲述过的 Scene 类、Modifier 类及 EaseFunction 类。不过不用担心，这部分内容可以作为参考手册，当需要选取适当的 Modifier 类以实现游戏场景的切换效果时，再回过头来查阅。

- ❑ 场景是 AndEngine 游戏的基本构建材料。
- ❑ 因为 Scene 类是从 Entity 类继承下来的，本章深入分析了 Entity.java 的代码，以及 Entity 类的构造器与方法，描述了每个重要属性的意义及其在 AndEngine 中的使用方式。
- ❑ 许多种 Modifier 都可以应用在 Entity 对象上（例如 Scene、Sprite）以改变其显示效果。本章描述了 AndEngine 所提供的每一种 Modifier，并给出了使用它们来改变 Entity 对象的方式。如果需要的话，可以用并行的或序列化的方式将多个 Modifier 对象联合起来使用。
- ❑ AndEngine 提供了多种 EaseFunction，用以修正 Modifier 对象执行动作的过程。
- ❑ 本章运用了一些新知识来继续制作 V3 游戏，创建并载入了主游戏画面，其后运用了一些动作效果，尽管它们现在看起来并不是十分灵巧。

下一章将详述 Sprite 类, 它会使 V3 游戏变得活跃起来, 不像现在这样略显死板。

## 4.4 习题

1. 写一个用于测试联合使用 Modifier 对象的 Activity 类, 该类将在白色背景上显示一个 Sprite 对象。该精灵的贴图可以使用本章下载代码中 Modifiers 项目内所包含的 mathead.png 文件。

2. 构造一个 Modifier 对象, 用它将某 Entity 对象按如下时间轴所述效果进行修改:

	第 0 秒	第 1 秒	第 2 秒	第 3 秒	第 4 秒	第 5 秒
移动	从原点到屏幕中心			不变		
颜色	由白至蓝		由蓝至红		由红至绿	
缩放因子	不变	由原有大小变为一半		再由一半大小复原		不变
旋转角度	不变		720 度		不变	

3. 现在当主菜单收缩时, 其收缩点位于坐标 (0.0f, 0.0f) 处。请修改代码让其向屏幕中心点收缩。(提示: 使用 setScaleCenter() 方法。)

4. 如果 AndEngine 所提供的 34 种 EaseFunction 无法满足游戏制作的需要, 那么开发者可以创建自己的类型。请新建一个名为 EaseWiggle 的缓动函数, 使其效果如图 4.14 所示。(提示: 在 AndEngine 源文件 EaseLinear.java 的基础上进行修改。可以在 getPercentageDone() 方法的代码内使用 float Math.sin() 方法。)

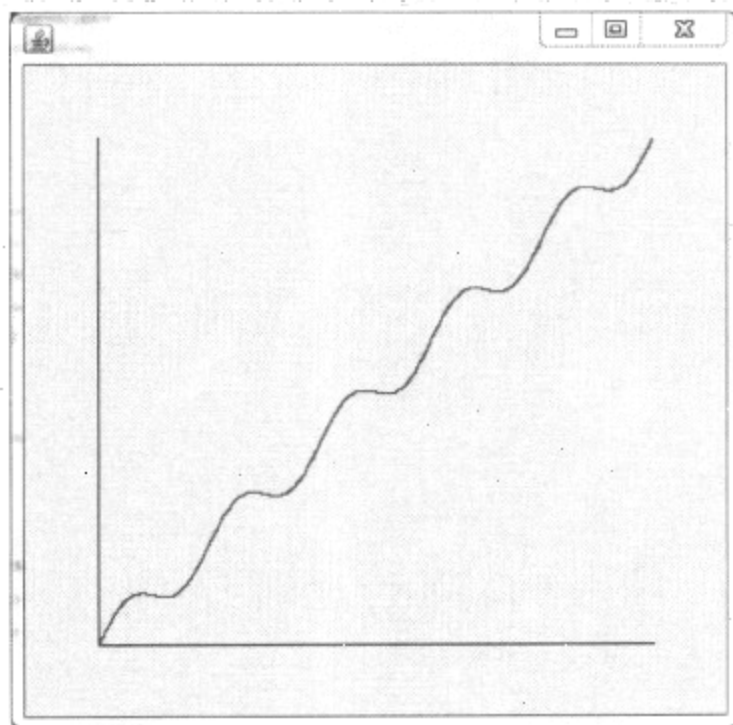


图 4.14 EaseWiggle 效果图





## 第5章

# 绘制与精灵

- 5.1 快速回顾 Entity 类
- 5.2 绘制线条与矩形
- 5.3 精灵
- 5.4 总结
- 5.5 习题

在范例代码与 V3 的开发过程中，始终在使用 Sprite 类，现在需要花些时间来详细研究一下它了。本章将会讲到 AndEngine 所提供的用于创建 Sprite、AnimatedSprite 及 TiledSprite 等精灵对象的方法，也会简述用于绘制诸如线条与矩形等基本几何图形所用的方法。

在游戏中，屏幕上所绘制的大部分物体都是精灵。“精灵”一词本来指放置在游戏背景上且带有动画的独立图形对象，随着电脑游戏的发展，该词可指代任何图形对象。例如，为了在 AndEngine 的场景中显示背景，需要先把背景载入为 Sprite 对象。

AndEngine 使用 OpenGL 将物体渲染至屏幕。在大多数 Android 设备上，OpenGL 都是以硬件加速形式实现的，这也是本书进行开发所预设的前提。AndEngine 让开发者不需要关注大部分 OpenGL 的细节，不过本书还是会讲解一些底层内容。

## 5.1 快速回顾 Entity 类

在 AndEngine 中，Sprite 的超类是抽象类 BaseSprite，后者的超类是抽象类 BaseRectangle；BaseRectangle 的超类是抽象类 RectangularShape，RectangularShape 的超类是抽象类 Shape；Shape 的超类是 Entity。如果这听起来很有点乱的话，用图来表示可能会清晰一些。图 5.1 描述了一张简化的 Entity 及其子类的类图。

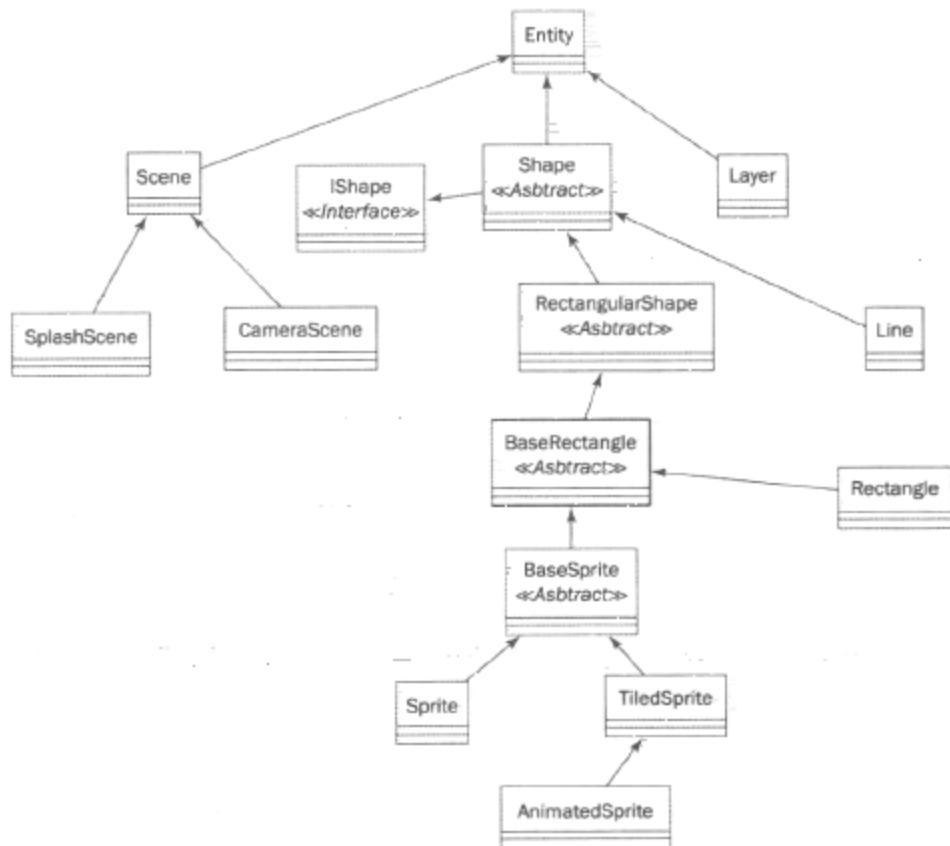


图 5.1 简化的 Entity 体系类图

通过图 5.1 可以检索到每一个类，并找到该类在超类基础上所增加的功能。尽管游戏开发者并不需要理解如此深入的细节，不过，了解它们组合起来的机制也是好的。下文

在讲述绘制与精灵时，将提到这些类之间的重要关系以及构建游戏所需要的重要方法。

## 5.2 绘制线条与矩形

AndEngine 没有提供大量的绘制功能。对于大多数游戏来说，图片资源都是由一系列的位图组成的，很少需要在游戏运行时绘制复杂的图形。AndEngine 提供了绘制线条和矩形的方式。

### 5.2.1 线条

除了从 Entity 类继承下来的属性之外，Line 还有指示其终点的位置属性，以及宽度属性。这些参数可以经由 Line 构造器传递进来：

```
Line(final float pX1, final float pY1, final float pX2, final float pY2)
Line(final float pX1, final float pY1, final float pX2, final float pY2,
    final float pLineWidth)
```

如果未传入 pLineWidth 属性，其值则默认为 1 个像素。第一组坐标视为该线条的一个端点位置，第二组视为其另一端的位置。其余诸如颜色、透明度等属性是经由超类 Entity 来处理的。

### 5.2.2 矩形

AndEngine 也提供了绘制基本矩形的方法。通过如下构造器构建之：

```
Rectangle(final float pX, final float pY, final float pWidth, final float pHeight)
Rectangle(final float pX, final float pY, final float pWidth, final float pHeight,
    final RectangleVertexBuffer pRectangleVertexBuffer)
```

pX 和 pY 的值表示其左上角坐标，pWidth 和 pHeight 代表宽度和高度。和 Line 一样，颜色与透明度等属性也是经由超类 Entity 来处理的。Rectangle 对象在绘制时是具有填充色的，如果仅需绘制矩形外框，则可通过绘制四条线段来实现。

可选的 pRectangleVertexBuffer 参数可以用来在绘制大量矩形时提高绘制速度，也可以用来扭曲矩形。这些功能超出了本书的讨论范围，如果读者有兴趣，请搜索“OpenGL vertex buffer”，可以找到大量的在线参考资料。

## 5.3 精灵

从图 5.1 下方可以看到，AndEngine 提供了三种不同的 Sprite：

□ Sprite 使用从 TextureRegion 所构建的单一贴图。

- TiledSprite 使用从 TiledTextureRegion 所构建的贴图数组中的贴图。
- AnimatedSprite 是 TiledSprite 的子类，其贴图会以固定频率变换，以展现动画效果。

### 5.3.1 贴图

在了解各种 Sprite 之前，需要讲解一些 AndEngine 如何使用贴图的相关背景知识。如果读者没有做过图形编程的话，将贴图理解成绘制在 Sprite 对象上的位图就可以了。AndEngine 在内存中以 Texture 对象的形式存储所有的贴图，并以 TextureManager 单例<sup>①</sup>来管理游戏中的所有 Texture 对象。

每个 Texture 对象内部可含有多个代表位图的 TextureRegion 对象。AndEngine 有两种基本的 TextureRegion 类型：

- TextureRegion 通常含有一张具有特定高度与宽度的图片，如图 5.2 所示。
- TiledTextureRegion 通常含有一张以上的图片，每张图片都具有相同的高度与宽度。这些图片以矩阵形式组织起来，每一张都可以通过其在矩阵中的位置来定位，如图 5.3 所示。

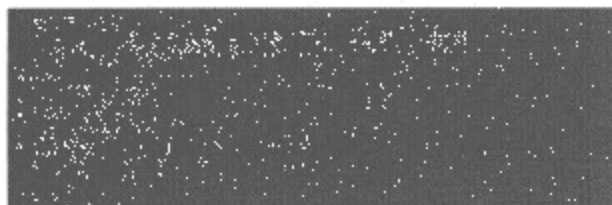


图 5.2 TextureRegion 图像

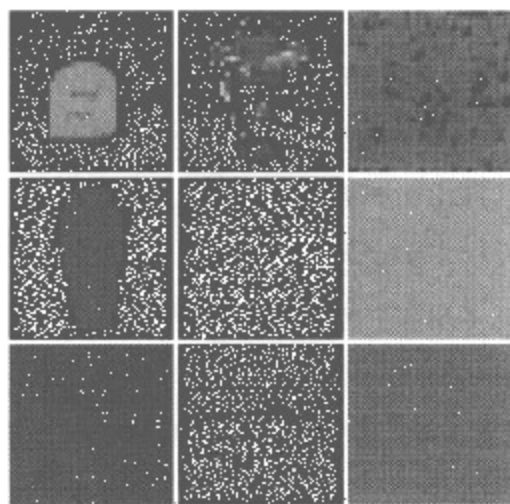


图 5.3 TiledTextureRegion 图像

#### Texture 类

如下构造器可以创建 Texture 对象：

```
Texture(final int pWidth, final int pHeight)
Texture(final int pWidth, final int pHeight, final ITextureStateListener
        pTextureStateListener)
Texture(final int pWidth, final int pHeight, final TextureOptions
        pTextureOptions)
Texture(final int pWidth, final int pHeight, final TextureOptions
```

① 原文为 singleton，中文一般译为“单例”或“单件”，是一种设计模式。参见 <http://zh.wikipedia.org/zh-cn/%E5%8D%95%E4%BE%8B%E6%A8%A1%E5%BC%8F>。——译者注

```
pTextureOptions, final ITextureStateListener pTextureStateListener)
```

所有的构造器都会构造出给定大小的空白桌布，开发者可以在其上为 Sprite 对象载入贴图。Texture 对象的大小，也就是 pWidth 与 pHeight，必须是 2 的正整数次幂（32、64、128 等等），而且要足够大，可以容纳下所有需要载入的贴图。如果长宽值不是 2 的幂，AndEngine 会抛出 IllegalArgumentException 异常。如果载入了该 Texture 无法容纳的位图，也会抛出异常。这些异常都会导致游戏强制关闭，如果没有被捕捉的话，在 LogCat 控制台上还会出现记录信息。

出现在第二个与第四个构造器中的 pTextureStateListener 参数是可选的，它所指示的监听器，会在该 Texture 对象被载入贴图、卸载贴图或发生错误时得到通知。本书不会用到这个功能，不过如果读者感兴趣的话，可以在 AndEngine Example 的源代码文件 ImageFormatsExample.java 中找到使用范例。

可选的 pTextureOptions 参数控制 OpenGL 渲染贴图的方式，通常使用默认值即可，但是也可以使用以下值：

- ☐ TextureOptions.NEAREST
- ☐ TextureOptions.BILINEAR
- ☐ TextureOptions.REPEATING
- ☐ TextureOptions.REPEATING\_BILINEAR
- ☐ TextureOptions.NEAREST\_PREMULTIPLYALPHA (默认值)
- ☐ TextureOptions.BILINEAR\_PREMULTIPLYALPHA
- ☐ TextureOptions.REPEATING\_PREMULTIPLYALPHA
- ☐ TextureOptions.REPEATING\_BILINEAR\_PREMULTIPLYALPHA

以上选项都将根据 AndEngine 源文件 TextureOption.java 中定义的模式来设定 OpenGL 贴图过滤器。针对每种选项进行描述属于 OpenGL 相关的话题，不在本书讨论范围之内，不过不同选项之间的差别还是非常明显的，尤其是在缩放或旋转贴图时。如果关心贴图渲染的细节，建议深入研究该问题，特别要看看如下这篇非常好的讨论贴图渲染的文章：<http://www.opengl.org/wiki/Texture>。

本书中的大部分范例都将使用 BILINEAR\_PREMULTIPLYALPHA 作为参数值，它将会让 OpenGL 以如下属性绘制：

- ☐ GL\_LINEAR：在放大或缩小贴图时使用线性插值法来决定 x、y 轴方向上的像素颜色。
- ☐ GL\_CLAMP\_TO\_EDGE：贴图取值不会回绕，而是固定在 [0,1] 这个区间中<sup>⊖</sup>。
- ☐ GL\_MODULATE：使用乘法计算两个贴图合成的颜色值<sup>⊖</sup>。

⊖ 此属性的意义参见 [http://www.opengl.org/wiki/Texture#Edge\\_value\\_sampling](http://www.opengl.org/wiki/Texture#Edge_value_sampling)。——译者注

⊖ 此属性的意义参见 [http://www.opengl.org/wiki/Texture\\_Coiners](http://www.opengl.org/wiki/Texture_Coiners)。——译者注

□ 使用预乘式透明度渲染法<sup>⊖</sup>，该方法在贴图重叠时会产生更加真实的色彩叠加效果。

### TextureRegionFactory 类

在创建了空白的贴图存储区后，需要使用 TextureRegionFactory 为 Texture 对象载入图像，V3 游戏中精灵所用的贴图都是通过此方式载入的。不过，现在先来看看该类提供的所有功能。TextureRegionFactory 可以使用三种素材来创建 TextureRegion 与 TiledTextureRegion 对象：

- Asset 资源：存放于游戏项目 assets 文件夹下的位图。V3 游戏一直用的是这种方式。
- Resource 资源：存放于游戏项目 res 文件夹下的可绘制文件。
- TextureResource 资源：涵盖了前两种资源类型的更加通用的资源形式。以前两种素材为参数的工厂方法也适用于该类型的参数。当需要复用已有的 TextureResource 对象时，也可以使用这些方法。

每种素材来源都有两个对应的方法，一个是创建瓦片式的，一个是创建非瓦片式的。还有一系列用于创建 BuildableTexture 对象的方法将在下一节讲到，这里先看看针对上述素材来源所提供的基本方法：

```
TextureRegion createFromAsset(final Texture pTexture,
    final Context pContext, final String pAssetPath,
    final int pTexturePositionX, final int pTexturePositionY)
TiledTextureRegion createTiledFromAsset(final Texture pTexture,
    final Context pContext, final String pAssetPath,
    final int pTexturePositionX, final int pTexturePositionY,
    final int pTileColumns, final int pTileRows)
TextureRegion createFromResource(final Texture pTexture,
    final Context pContext, final int pDrawableResourceID,
    final int pTexturePositionX, final int pTexturePositionY)
TiledTextureRegion createTiledFromResource(final Texture pTexture,
    final Context pContext, final int pDrawableResourceID,
    final int pTexturePositionX, final int pTexturePositionY,
    final int pTileColumns, final int pTileRows)
TextureRegion createFromSource(final Texture pTexture,
    final ITextureSource pTextureSource, final int pTexturePositionX,
    final int pTexturePositionY)
TiledTextureRegion createTiledFromSource(final Texture pTexture,
    final ITextureSource pTextureSource, final int pTexturePositionX,
    final int pTexturePositionY, final int pTileColumns, final int pTileRows)
```

参数意义如下：

- Texture pTexture：需要为其载入贴图的 Texture 对象。

⊖ 原文为 premultiplied alpha blending，具体意义参见：[http://en.wikipedia.org/wiki/Alpha\\_compositing](http://en.wikipedia.org/wiki/Alpha_compositing)。——译者注



- ❑ Context pContext: 当前 Activity 对象的 Context 环境对象<sup>⊖</sup>。
- ❑ String pAssetPath: 该贴图资源的文件名, 默认会在 assets 目录下查找, 如果需要改变查找路径, 可以使用后文将要讲到的 setAssetPath() 方法。AndEngine 现在支持 PNG、JPG 及 BMP 格式的图像。
- ❑ pDrawableResourceID: 在 R.java 中为资源所定义的整数 ID, 通常以 R.drawable.<资源名> 的形式引用它。
- ❑ ITextureSource pTextureSource: 需要载入的 TextureSource 对象。
- ❑ int pTexturePositionX, int pTexturePositionY: 载入的贴图在 Texture 对象中所占据的位置。该位置为图像左上角的位置, 而且 Texture 必须足够大, 能容下该图片。Texture 中所载入的图像不应该重叠, 除非开发者刻意打算做出奇怪的效果。
- ❑ int pTileColumns, int pTileRows: 对于 TiledTextureRegion 来说, 表示瓦片图所在的列与行坐标。

TextureRegionFactory 也包含了一个非常有用的方法, 用来将查找资源所用的路径设置为 assets 目录下的子目录。如果不同类型的图片放在不同的子目录下, 使用该方法可以避免在加载图片时重复输入整个路径名:

```
void setAssetBasePath(final String pAssetBasePath)
```

参数为局部路径名, 必须以 “/” 结尾, 而且不能是空字符串, 否则都会抛出异常。

#### 范例: 用 assets 目录下的资源创建 TextureRegion 对象

在第 4 章的代码中, 使用了 setAssetBasePath() 和 TextureRegionFactory.createFromAsset() 为 V3 游戏的第 1 关设置了 TextureRegion 对象。程序清单 5.1 是一段代码节选, 展示了使用 assets 目录下的图像文件创建 Sprite 对象的通常方式。

程序清单 5.1 Level1Activity.java 代码节选: 使用 TextureRegionFactory.createFromAsset()

```
@Override
public void onLoadResources() {
    /* 载入 Textures 对象。 */
    TextureRegionFactory.setAssetBasePath("gfx/Level1/");
    mLevel1BackTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mLevel1BackTextureRegion =
        TextureRegionFactory.createFromAsset(
            this, mLevel1BackTexture,
            this, "Level1Bk.png", 0, 0);
```

⊖ Context 是 Android 系统特有的类, 代表某应用程序的运行环境信息, 具体意义参见: <http://developer.android.com/reference/android/content/Context.html>。——译者注

```

        mEngine.getTextureManager().loadTexture(
            this.mLevel1BackTexture);
    }

```

代码开始处先将 asset 基路径设置为“gfx/Level1/”，因为我们是按照这种结构来组织 assets 目录结构的。所有的图片都在 gfx 子目录下，每一关的图像都有单独的子目录存放于其下。记得结尾的“/”字符，否则 AndEngine 会抛出异常。

其后的代码创建了一个足够大的 Texture 对象，以存放第 1 关的背景。背景图是一个 480 像素 × 320 像素的 PNG 文件，不小于其长宽的下一个 2 的整数幂是 512，故而这里需要创建 512 像素 × 512 像素的空间。为了和范例游戏相仿，这里使用了 BILINEAR\_PREMULTIPLYALPHA 作为 TextureOptions 的参数。

然后，用 assets 目录下的 PNG 文件创建 TextureRegion 对象并将其载入到 Texture 对象的 (0,0) 位置。如果还要载入其他的 TextureRegion 对象到 Texture 的话，注意载入的位置不要和已有的背景图片冲突。

最后用 TextureManager 单例将 Texture 对象载入到缓存中，这样在游戏中就可以使用它了。

#### 范例：用 Resource 资源创建 TextureRegion 对象

用 Resource 资源创建 TextureRegion 对象与用 Asset 资源相比，有一个非常大的好处。读者应该知道，Android 定义了三种屏幕分辨率模式：hdpi、mdpi 与 ldpi，并且将资源按照此模式分类管理，以应对多种屏幕分辨率。先将资源放在 res/drawable-hdpi、res/drawable-mdpi 及 res/drawable-ldpi 目录下，然后通过 R.drawable.<资源名> 的形式引用它，Android 系统就会根据应用程序所运行设备的屏幕大小，找出最符合的图片来。这种能力对开发在不同屏幕大小下运行的软件很有帮助。

程序清单 5.2 展示了如何在 Level1Activity.java 中使用 Resource 不是 Asset 资源来创建背景障碍物贴图。

程序清单 5.2 Leve1Activity.java 代码节选：使用 createFromResource() 来创建 TextureRegion

```

package com.pearson.lagp.v3;

...

@Override
public void onLoadResources() {
    /* 载入 Textures 对象。 */
    mLevel1BackTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mLevel1BackTextureRegion =
        TextureRegionFactory.createFromResource(
            this.mLevel1BackTexture, this, R.drawable.level1bk,

```

```

        0, 0);
        mEngine.getTextureManager().loadTexture(
            this.mLevel1BackTexture);
        ...
    }

```

要让上述代码运行，必须在 `res/drawable-xdpi` 目录下放入适当的图片文件，文件名必须全部小写，例如代码中的“`level1bk.png`”。`createFromResource()` 方法需要传入对该资源的引用，该引用以“`R.drawable`”开头，加上除去后缀名部分之后的文件名。此类图片必须是 PNG 或者 JPG 格式的，尽管 Android 系统支持 GIF 格式，但是 AndEngine 不支持，所以开发者应该首选 PNG 格式。

### 范例：用矢量图（SVG）创建 TextureRegion 对象

AndEngine 有个扩展包可以在程序运行时绘制 SVG 格式的矢量图，这是一个对基础游戏引擎非常有用的增强功能，尤其是为高清屏幕（high-definition, HD）开发游戏的时候。在运行时绘制的矢量图，可以根据不同大小的屏幕进行优化，同时保持图像不失真；如果在开发阶段绘制 SVG 矢量图的话，则必须创建至少三份位图图像，低分辨率屏幕一份、中等分辨率屏幕一份、高分辨率屏幕一份。

要使用此扩展包，必须从如下网站下载 .jar 文件（Java 压缩文件）并载入到游戏项目中：<http://code.google.com/p/andenginesvgtextureregionextension>

将 .jar 文件放入游戏项目的 lib 文件夹下，右击文件名，在弹出式菜单中选择“Build Path”菜单，再选择“Add to Build Path”菜单项。添加好扩展之后，就可以将 SVG 载入 Texture 对象中了，如程序清单 5.3 所示。

程序清单 5.3 Level1Activity.java 代码节选：使用 SVG 矢量图创建 Texture 对象

```

@Override
public void onLoadResources() {
    /* 载入 Textures 对象。 */
    mLevel1BackTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mLevel1BackTextureRegion =
        mLevel1BackTextureSource = new SVGAssetTextureSource(
            this, "svg/hatchet40.svg", 1.0f);
    TextureRegionFactory.createFromSource(
        this.mLevel1BackTexture,
        this.mLevel1BackTextureSource, 0, 0);
    mEngine.getTextureManager().loadTexture(
        this.mLevel1BackTexture);
    ...
}

```

## BuildableTexture

以上几套方法都能很好地构建 TextureRegion 对象并将其载入到 Texture 对象中, 不过如果在实际开发中使用它们, 开发者可能就会注意到一个缺陷: 每次必须告诉 TextureRegionFactory 所加载的图片在 Texture 对象中所处的位置。

假设有三张图片需要载入:

- One.png: 50 像素 × 121 像素
- Two.png: 78 像素 × 324 像素
- Three.png: 233 像素 × 43 像素

开发者必须规划出如何安排这些图像, 小心地计算出调用 TextureRegionFactory 方法所使用的坐标。这非常麻烦, 所以 AndEngine 提供了一组可以自动安排图片位置的方法。下面的 Level1Activity.java 代码将 BuildableTexture 与第 4 章中用到的构建方法结合起来使用。程序清单 5.4 节选了部分代码, 以展示如何使用这种方式构建 Texture 对象。

程序清单 5.4 Level1Activity.java 代码节选: 使用 BuildableTexture

```
package com.pearson.lagp.v3;
...
@Override
public void onLoadResources() {
    /* 载入 Textures 对象。 */
    ...
    mObstacleBoxTexture = new BuildableTexture(512, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mBoxTextureRegion =
        TextureRegionFactory.createFromAsset(mObstacleBoxTexture,
            this, "Obstaclebox.png");
    mBulletTextureRegion =
        TextureRegionFactory.createFromAsset(mObstacleBoxTexture,
            this, "Bullet.png");
    mCrossTextureRegion =
        TextureRegionFactory.createFromAsset(mObstacleBoxTexture,
this, "Cross.png");
    mHatchetTextureRegion =
        TextureRegionFactory.createFromAsset(mObstacleBoxTexture,
            this, "Hatchet.png");
    try {
        mObstacleBoxTexture.build(
            new BlackPawnTextureBuilder(2));
    } catch (final TextureSourcePackingException e) {
        Log.d(tag,
            "Sprites won't fit in mObstacleBoxTexture");
    }
    this.mEngine.getTextureManager().loadTexture(this
        .mObstacleBoxTexture);
}
```

```

    }
    ...
}

```

BuildableTexture 的构造器与先前的 Texture 类非常相似，也具有同样的参数限制条件（例如尺寸必须是 2 的正整数幂，大小必须足够容纳所有贴图等等）。createFrom... 系列方法也与先前所属的那一组方法很相似，只不过 Texture 类型的参数换成了 Buildable-Texture 类型，并且没有坐标参数了。

这两种方式的主要区别是，当创建完所有 TextureRegion 对象后，在使用 BuildableTexture 之前必须先调用它的 build 方法，并且传入一个类似 BlackPawnTextureBuilder 的构建器对象（现在只提供了这一个构建器，但是以后可能还有别的）。如果 build 方法成功执行，那么所有 TextureRegion 对象都会被打包入 BuildableTexture 中，并且可以按照资源名字访问它们；如果执行不成功（例如在分配的 Texture 空间内无法放至所有贴图），可以如程序清单 5.4 中所示，将发生的异常捕捉下来。

### 构建 Texture 对象的另一种方式

开发者有可能不想在运行时构建 Texture 对象，如果是这样的话，可能会用到一个叫做 Zwoptex 的流行工具来制作贴图（有时也叫做精灵表格，sprite sheet）。尽管 Zwoptex 所做的精灵表不能直接导入 AndEngine 中，但是可以利用该工具安排图像并记录其坐标。它有两个版本可选：

- ❑ 基于网页（Adobe Flash）的旧版本仍然可用，但是已经不再维护（<http://zwoptexapp.com/flashversion>）。
- ❑ 全功能版本仅能在 Mac OS X（10.6 及以后版本）系统上运行，而且需要一点费用来购买（<http://zwoptexapp.com/buy>）。

这两个版本都可以自动地将导入的图像拼接成一大张单一的精灵表，或者说贴图。最后的成品是一张精灵表图像文件和一个记录了各个小图片坐标列表的 XML 文件。尽管 AndEngine 目前仍然无法直接导入这个图片与列表（也许在读者阅读本书时，该功能已经做出来了），但是开发者可以通过查看 XML 列表的内容而轻松地获取到调用 createFromAsset() 方法所需的坐标值，这样就不需要自己去计算了。

作为演示，笔者使用 Flash 版的 Zwoptex 将 V3 游戏第 1 关的武器盒所需的图片组合起来。图 5.4 展示了最终的成品图，程序清单 5.5 提供了 XML 文件中与贴图大小及图片坐标相关的部分。

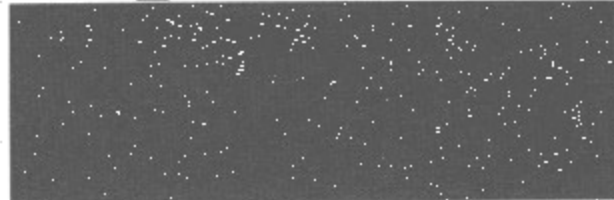


图 5.4 武器盒的贴图

程序清单 5.5 XML 文件中与贴图大小及图片坐标相关部分的节选

---

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>texture</key>
    <dict>
        <key>width</key>
        <integer>256</integer>
        <key>height</key>
        <integer>128</integer>
    </dict>
    <key>frames</key>
    <dict>
        <key>Bullet.png</key>
        <dict>
            <key>x</key>
            <integer>45</integer>
            <key>y</key>
            <integer>1</integer>
            <key>width</key>
            <integer>11</integer>
            <key>height</key>
            <integer>33</integer>
            <key>offsetX</key>
            <real>0</real>
            <key>offsetY</key>
            <real>0</real>
            <key>originalWidth</key>
            <integer>11</integer>
            <key>originalHeight</key>
            <integer>33</integer>
        </dict>
        <key>Cross.png</key>
        <dict>
            <key>x</key>
            <integer>1</integer>
            <key>y</key>
            <integer>1</integer>
            . . .
        </dict>
    </dict>
</plist>

```

---

对于每一张图片，其 pX 和 pY 的值都对应于 XML 文件中的 x、y 属性值。可以使用 Zwoptex 观察拼接好的贴图，找出每张图片在大贴图上的坐标，同时确保贴图足够大，能够容纳下所有图片。



## Sprite 类

有了 Sprite 所用的 TextureRegion 对象之后，就可以创建 Sprite 对象本身了。Sprite 类有 4 个构造器：

```
Sprite(final float pX, final float pY, final TextureRegion pTextureRegion)
Sprite(final float pX, final float pY, final float pWidth, final float pHeight,
        final TextureRegion pTextureRegion)
Sprite(final float pX, final float pY, final TextureRegion pTextureRegion,
        final RectangleVertexBuffer pRectangleVertexBuffer)
Sprite(final float pX, final float pY, final float pWidth, final float pHeight,
        final TextureRegion pTextureRegion,
        final RectangleVertexBuffer pRectangleVertexBuffer)
```

以上所有构造器均需要通过 pX 与 pY 指定精灵的初始坐标，同时需要通过 pTextureRegion 告诉 AndEngine 及 OpenGL 绘制该精灵时所用的贴图。可以为精灵对象指定宽度及高度，如果未指定的话，则使用 TextureRegion 的大小作为其宽高值，也可以像创建 Rectangle 时那样，指定一个 RectangleVertexBuffer 对象。

因为 Sprite 类是 Entity 类的子类，所以 Entity 类的所有方法 Sprite 类也可使用。Sprite 类还导入了一个新的方法：

```
TextureRegion getTextureRegion()
```

该方法正如其名称所示，返回 Sprite 对象所用的 TextureRegion 对象。注意，并没有设置 TextureRegion 对象的方法，此对象必须在创建时传入，除非此精灵使用下一节所述的瓦片式贴图。

## TiledSprite 类

可以用 TiledSprite 创建一种使用矩阵式贴图的精灵。这些贴图存储在 TiledTextureRegion 对象中，它通常含有一个以上的贴图，这些贴图大小相同，存储在一个大的矩阵中。在前文的 TextureRegionFactory 一节中，已经讲述了创建 TiledTextureRegion 的方法，现在可以用与 Sprite 类相似的构造器来创建 TiledSprite 对象了：

```
TiledSprite(final float pX, final float pY, final TiledTextureRegion
            pTiledTextureRegion)
TiledSprite(final float pX, final float pY, final float pTileWidth,
            final float pTileHeight, final TiledTextureRegion pTiledTextureRegion)
TiledSprite(final float pX, final float pY, final TiledTextureRegion
            pTiledTextureRegion, final RectangleVertexBuffer
            pRectangleVertexBuffer)
TiledSprite(final float pX, final float pY, final float pTileWidth,
            final float pTileHeight, final TiledTextureRegion pTiledTextureRegion,
            final RectangleVertexBuffer pRectangleVertexBuffer)
```

参数与 Sprite 类的构造器很像, 不过这里要用 TiledTextureRegion, 而非 TextureRegion。这里的宽度与高度参数, 指的是每个瓦片贴图的大小。

TiledSprite 类在 BaseSprite 类的基础之上, 又增加了一些特有的方法, 其中有如下四个:

```
int getCurrentTileIndex()
void setCurrentTileIndex(final int pTileIndex)
void setCurrentTileIndex(final int pTileColumn, final int pTileRow)
void nextTile()
```

前三个方法用于获取与设置当前 TiledSprite 所用的贴图索引, 可以通过它查询当前精灵所显示的是哪张贴图, 也可以根据需要改变当前所用的贴图。最后一个方法使得当前所用的贴图索引向后移动一个位置, AndEngine 排列瓦片贴图的顺序如图 5.5 所示。

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

图 5.5 瓦片贴图的顺序

### AnimatedSprite 类

如果需要带有动画效果的精灵, 则必须提供所有的动画贴图, 以便让 AndEngine 应用于 Sprite 对象上。如图 5.1 那张简化版的类图所述, AnimatedSprite 是 TiledSprite 类的子类, 它会安排 TiledTextureRegion 中的贴图以做出动画效果来。该类的构造器同 TiledSprite 类很相似:

```
AnimatedSprite(final float pX, final float pY, final TiledTextureRegion
    pTiledTextureRegion)
AnimatedSprite(final float pX, final float pY, final float pTileWidth,
    final float pTileHeight, final TiledTextureRegion pTiledTextureRegion)
AnimatedSprite(final float pX, final float pY, final TiledTextureRegion
    pTiledTextureRegion, final RectangleVertexBuffer
    pRectangleVertexBuffer)
AnimatedSprite(final float pX, final float pY, final float pTileWidth,
    final float pTileHeight, final TiledTextureRegion pTiledTextureRegion,
    final RectangleVertexBuffer pRectangleVertexBuffer)
```

在第 6 章深入讲解动画时, 会详述该类的用法。

### 5.3.2 效率问题

随着游戏的开发, Android 设备需要执行的任务会越来越多, 这会带来执行效率的问题。显示大量的图像、运行修改器、实现游戏逻辑, 这些都需要占用 CPU 周期, 从而影响游戏的运行速度。

迄今为止本书所展示的代码范例中, 都使用了 AndEngine 所提供的标准帧速率计数器 FPSLogger, 它会将帧速率打印到 LogCat 控制台上。打印的信息格式类

似这样：

```
D/AndEngine( 448): FPS: 9.73 (MIN: 96 ms | MAX: 134 ms)
```

这条信息是在用 Android 模拟器运行游戏时打印的，所以帧速率非常慢——低于 10 帧/秒（游戏在这里仅仅是显示一张位图）。帧速率是越高越好，这样游戏运行效果会很流畅。

创建 Sprite 对象的方式会影响到游戏的整体效率。如果每个 Sprite 对象都绑定独立的 Texture 对象，那么 AndEngine 在渲染它之前必须做一系列工作，以伪代码表示如下：

```
< 遍历所有将要绘制的精灵 >
    < 初始化 OpenGL 渲染机制 >
    < 渲染某个精灵对象 >
    < 清理渲染引擎 >
< 结束循环 >
```

如果很多个 Sprite 对象都使用同一个 Texture 对象中的贴图，那么初始化和清理工作可以放在循环外完成。也就是说，这两项工作只需要执行一次就好，这就大大提高了游戏执行效率：

```
< 初始化 OpenGL 渲染机制 >
< 遍历所有将要绘制的精灵 >
    < 渲染某个精灵对象 >
< 结束循环 >
< 清理渲染引擎 >
```

游戏中通常会用到大量的精灵，所以将很多贴图放入一个 Texture 对象，让精灵公用，可以提高渲染速度，从而提高游戏的帧速率。

### 5.3.3 复合精灵

到现在为止我们所见的 Sprite 对象都是没有子对象的，然而，Sprite 和 Entity 一样，是可以含有子 Sprite 对象的。这样就可以将整个精灵组作为一个整体来看待。这么做有时是必要的，比如游戏中要将某个演员和其所持武器绑定起来。如果稍后该演员移动或者旋转，那么武器当然也要随之改变了。向 Sprite 上附加子对象的方法同其他的 Entity 对象一样：

```
Entity.attachChild(final IEntity pEntity)
```

pEntity 就是要加入的子 Sprite，如果上级 Sprite 对象应用了某修改器效果的话，那么在子对象身上也会应用同样的效果。

为了演示这种用法，本章的范例代码写了一个 Activity 类，叫做 SpriteTestActivity。它描绘了一个新形象：挥动着斧头的 Mad Mat，图 5.6 是游戏运行的截屏。

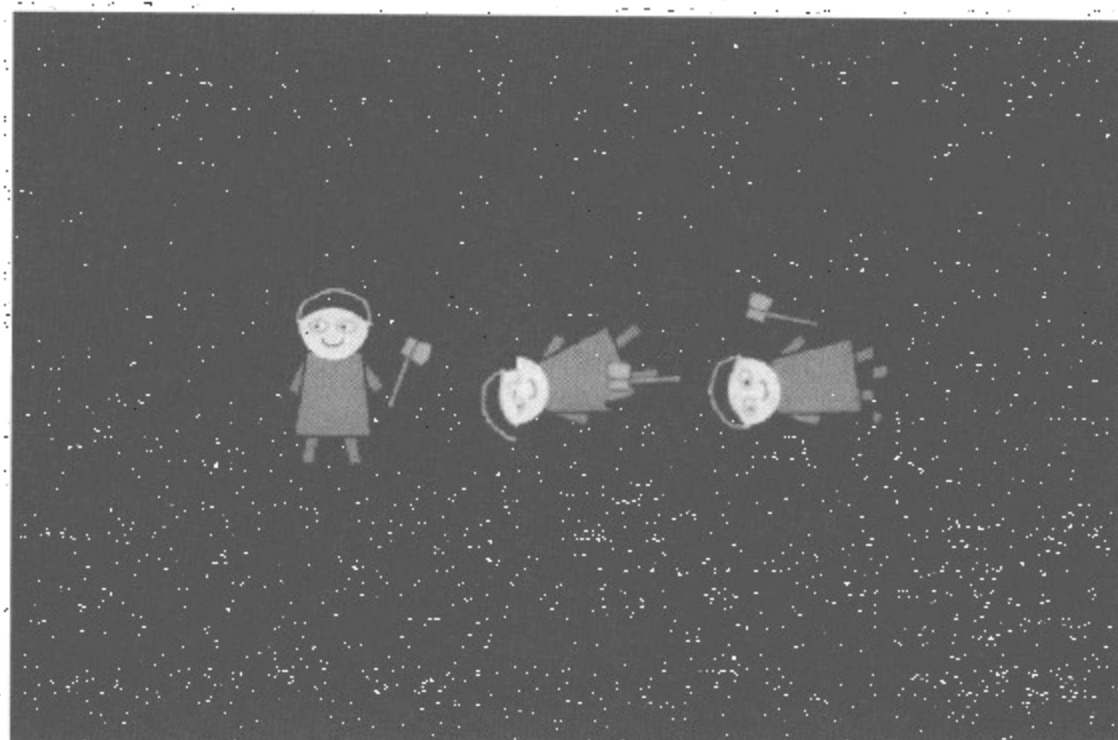


图 5.6 SpriteTestActivity 运行截屏

左边静止的那个可以叫做 0 号 Mad Mat, 因为它在代码中也叫这个名字。中间那个 1 号 Mad Mat 就地旋转, 手持的斧头也就地旋转。右边的 2 号 Mad Mat 手中的斧头则随着角色本身一起旋转, 这才是通常需要的效果。相关代码如程序清单 5.6 所示。

程序清单 5.6 SpriteTestActivity.java 与复合精灵的使用

```
package com.pearson.lagp.v3;

import org.anddev.andengine.engine.Engine;
import org.anddev.andengine.engine.camera.Camera;
import org.anddev.andengine.engine.options.EngineOptions;
import org.anddev.andengine.engine.options.EngineOptions
    .ScreenOrientation;
import org.anddev.andengine.engine.options.resolutionpolicy
    .RatioResolutionPolicy;
import org.anddev.andengine.entity.modifier.RotationModifier;
import org.anddev.andengine.entity.scene.Scene;
import org.anddev.andengine.entity.scene.background.ColorBackground;
import org.anddev.andengine.entity.sprite.Sprite;
import org.anddev.andengine.entity.util.FPSLogger;
import org.anddev.andengine.opengl.texture.BuildableTexture;
import org.anddev.andengine.opengl.texture.TextureOptions;
import org.anddev.andengine.opengl.texture.builder
    .BlackPawnTextureBuilder;
import org.anddev.andengine.opengl.texture.builder.ITextureBuilder
    .TextureSourcePackingException;
import org.anddev.andengine.opengl.texture.region.TextureRegion;
import org.anddev.andengine.opengl.texture.region.TextureRegionFactory;
import org.anddev.andengine.ui.activity.BaseGameActivity;
```

```

import android.util.Log;

public class SpriteTestActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;
    private String tag = "SpriteTestActivity";

    // =====
    // 字段
    // =====

    protected Camera mCamera;

    protected Scene mMainScene;

    private BuildableTexture mTestTexture;
    private TextureRegion mMadMatTextureRegion;
    private TextureRegion mHatchetTextureRegion;

    // =====
    // 构造器
    // =====

    // =====
    // Getter 和 Setter
    // =====

    // =====
    // 覆写超类及接口中的方法
    // =====

    @Override
    public Engine onLoadEngine() {
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        return new Engine(new EngineOptions(true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,
                CAMERA_HEIGHT),
            this.mCamera));
    }

    @Override
    public void onLoadResources() {
        /* 载入 Textures 对象。 */
        TextureRegionFactory.setAssetBasePath("gfx/SpriteTest/");
        mTestTexture = new BuildableTexture(512, 256,
            TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    }
}

```



```

mMadMatTextureRegion =
    TextureRegionFactory.createFromAsset(mTestTexture,
        this, "madmat.png");
mHatchetTextureRegion =
    TextureRegionFactory.createFromAsset(mTestTexture,
        this, "hatchet40.png");
try {
    mTestTexture.build(
        new BlackPawnTextureBuilder(2));
} catch (final TextureSourcePackingException e) {
    Log.d(tag,
        "Sprites won't fit in mTestTexture");
}
this.mEngine.getTextureMahager().loadTexture(
    this.mTestTexture);
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    final Scene scene = new Scene(1);
    scene.setBackground(new ColorBackground(0.0f, 0.0f, 0.0f));
    /* 将摄像头置中。 */
    final int centerX = CAMERA_WIDTH / 2;
    final int centerY = CAMERA_HEIGHT / 2;
    /* 创建 Sprite 对象并将其加入 Scene 对象中。 */
    final Sprite madMat0 = new Sprite(centerX,
        (mMadMatTextureRegion.getWidth() / 2),
        100.0f,
        centerY, (mMadMatTextureRegion.getHeight()
            / 2),
        mMadMatTextureRegion);
    scene.getLastChild().attachChild(madMat0);
    final Sprite hatchet0 = new Sprite(madMat0.getInitialX() +
        44.0f,
        madMat0.getInitialY() + 20.0f,
        mHatchetTextureRegion);
    scene.getLastChild().attachChild(hatchet0);

    final Sprite madMat1 = new Sprite(centerX,
        (mMadMatTextureRegion.getWidth() / 2),
        centerY, (mMadMatTextureRegion.getHeight()
            / 2),
        mMadMatTextureRegion);
    scene.getLastChild().attachChild(madMat1);
    final Sprite hatchet1 = new Sprite(madMat1.getInitialX() +
        44.0f,
        madMat1.getInitialY() + 20.0f,
        mHatchetTextureRegion);
    madMat1.registerEntityModifier(

```



```

        new RotationModifier(3, 0, 360)
    );
    hatchet1.registerEntityModifier(
        new RotationModifier(3, 0, 360)
    );
    scene.getLastChild().attachChild(hatchet1);

    final Sprite madMat2 = new Sprite(centerX -
        (mMadMatTextureRegion.getWidth() / 2) +
        100.0f,
        centerY - (mMadMatTextureRegion.getHeight()
            / 2),
        mMadMatTextureRegion);
    final Sprite hatchet2 = new Sprite( 44.0f, 20.0f,
        mHatchetTextureRegion);
    madMat2.attachChild(hatchet2);
    madMat2.registerEntityModifier(
        new RotationModifier(3, 0, 360)
    );
    scene.getLastChild().attachChild(madMat2);
    return scene;
}

@Override
public void onLoadComplete() {
}
}

```

代码的重点是 `onLoadScene()` 方法，它为每一个 Mad Mat 角色创建了一个表示自身的 Sprite 对象和一个表示斧头的 Sprite 对象。对于 1 号 Mad Mat，为其两个 Sprite 都运用了修改器，所以它们分别旋转了。旋转是根据其各自的位置进行的，但这并不是通常想要的效果。

对于 2 号 Mad Mat，代码将斧头 Sprite 作为子对象加入到角色本身的 Sprite 上，然后仅对该 Sprite 运用修改器，并将该对象加入到 Scene 对象上，这样它的子对象也会随之加入。当修改器执行其效果时，会将整个复合精灵视为一个单元来旋转。注意，创建 `hatchet2` 这个 Sprite 对象时所使用的坐标是相对于它的上级对象，也就是 2 号 Mad Mat 来说的，并非像 `hatchet0` 与 `hatchet1` 对象那样，相对于整个场景。实际上，Sprite 对象的位置总是相对于其上级对象来说的，在这里强调这个问题，是因为其他所有的精灵都是用 Scene 作为其上级对象的。

## 5.4 总结

本章深入讲述了 Sprite 类和 Entity 类。尽管没有继续推进 V3 游戏的制作，但是仔

细地研究了创建与绘制 Sprite 对象的方式。

本章讲述了许多内容，包括 Sprite 类在 AndEngine 的类体系中的地位、Sprite 类的构造器以及该类的方法，现在可以总结一下创建并显示 Sprite 对象所需的步骤了：

1. 所有的 Sprite 对象都需要 TextureRegion 对象，而后者又需要被载入到 Texture 对象中去。在 onLoadResources() 方法中，需要创建一个足够大的 Texture 对象，以便容纳所有的 TextureRegion。如果需要在 AndEngine 在加入 TextureRegion 时自动安排其位置，则可以使用 BuildableTexture。

2. 使用 TextureRegionFactory.createFrom...() 方法创建 Sprite 所需的 TextureRegion 对象，并载入到 Texture 对象中去。创建 TextureRegion 可以通过三种来源：assets 目录下的图像文件、Android 系统的 Resource 资源以及其他已经创建的 TextureRegion 对象。

3. 如果使用了 BuildableTexture，则可调用其 build() 方法完成 Texture 对象的构建。

4. 通过调用 Engine.getTextureManager().loadTexture() 方法，使 TextureManager 将新建的 Texture 对象载入缓存中。

5. 在 onLoadScene() 方法中，创建 Scene 对象及其下属的 Sprite 对象。Sprite 对象的创建可以使用已经载入到 Texture 对象中的 TextureRegion 对象，同时可以创建所需的修改器并注册到 Sprite 对象上。Sprite 对象将作为 Scene 对象的子对象加入其中。


6. 在方法代码的最后，将 Scene 对象作为返回值返回给 AndEngine，这样整个场景及其精灵就会显示出来。

## 5.5 习题

1. 写一个简单的程序，绘制一个红色的五角星（只要外框线，不填充），然后令其围绕中心点旋转一圈。可以使用复合对象的手段做出适当的旋转效果来。

2. 修改 SpriteTestActivity.java，令其使用 SVG 文件而非 PNG 文件来创建 TextureRegion 对象。

3. 试着用 Zwoptex 为读者自己的游戏所用的图片制作出一个瓦片贴图来。该软件有一些效果本章并未讲到，例如设定间距，选择不同的打包策略等。用该软件将一些大小相同的图片组合成一张用以创建 TiledTexture 的大图。



## 第6章 动画

6.1 动画所需素材

6.2 动画的瓦片贴图

6.3 AndEngine 的动画

6.4 动画范例

6.5 将动画加入 LevelActivity 类

6.6 动画制作的问题

6.7 高级话题：从 3D 模型中制作 2D 动画

6.8 总结

6.9 习题

现在真正有趣的部分开始了：制作带有动画效果的精灵。尽管不是每个游戏都需要动画，但是导入动画确实可以提高游戏品质。

## 6.1 动画所需素材

游戏中的动画指的是会根据游戏动作改变其形态的精灵。在 AndEngine 的语境里，动画有别于不改变物体形态的简单变换与旋转，即使带有移动效果的简单操作也不能算动画。旋转与变换在本书第 4 章讲述修改器时已经讲过了。

播放动画效果，需要有描述该物体各种形态的帧图像，每个时间点一帧，在给定的时间段内每一帧都会显示出来。在观察者的眼中，这种图片序列会产生一幅运动的场景。帧速率越高，看起来越流畅。美国的电视信号以每秒 30 帧的速度播放，电影是 24 帧。尽管有些人宣称能感知到更高的帧速率之间的差别，大部分人觉得每秒 30 帧与每秒 60 帧没有任何区别。

本章第一个范例将使用一个带动画效果的蝙蝠，其动画效果中的每一帧如图 6.1、图 6.2、图 6.3 所示。



图 6.1 bat0.png



图 6.2 bat1.png



图 6.3 bat2.png

这些图片是用 Inkscape 手绘的（笔者的水平不及 Andy Warhol<sup>①</sup>），不过也可以使用任何绘图工具来做。如果读者是位资深的动画师，那么可能会比笔者了解更多有关图形制作过程的知识；如果不是，也可以通过研究其他游戏的精灵表来学习如何做出更好的动画。找一找网上的资源，就会发现很多精灵表都是从已有的游戏图像中提取出来，供大家学习之用的。开发者当然不能直接在自己的游戏中免费使用这些图像，但是可以参考它们从而更好地为自己的游戏制作动画。

开发者也可以使用动画制作软件来做动画图像（帧），例如 Windows 系统的 Anime Studio 或 Mac OS X 系统的 Pixen。这些软件可用来非常方便地在关键帧之间生成过渡帧。在蝙蝠动画范例中，只有 3 帧图像，不过通常来说游戏中的动画可能需要更多的帧。

有些软件可以将动画中的每一帧导出为独立的图像文件（像前面 3 幅图那样），有些则没有此功能，那些软件会生成 GIF 格式或 Adobe Flash 的 SWF 格式的文件。可以

① 安迪·沃霍尔（1928—1987），美国艺术家，是波普艺术最有名的开创者之一。以商业插画取得巨大成功。——译者注

使用第 2 章介绍的 AnimGet 软件从这些文件中捕捉每一帧的图片，此软件会监视屏幕上的某区域，动画的每一帧必然会引发像素的改变，从而被该软件记录为快照。

## 6.2 动画的瓦片贴图

从第 5 章关于精灵效率的讨论可以知道，将所有的动画帧都做成一张共用的精灵表是个好主意。根据已有的知识，将所有的贴图做成一张单一的精灵表并载入，可以大大提升绘制速度。

笔者使用 GIMP（开源的位图编辑器）将蝙蝠的瓦片贴图拼合成如图 6.4 所示的动画。AndEngine 要求瓦片贴图以矩阵方式排列，而且每张贴图大小要相同。所以笔者将每张瓦片贴图放置在 100 像素 × 100 像素的透明桌布上，拼合成一张 200 像素 × 200 像素的大图像。这里将 bat0.png 图像复制了一份以填满矩阵，不过也可以按照 1×3 的方式将整个动画横着排列，这样就不需要重复的那一帧了。后面的第二个例子将会演示这种排列方式。



图 6.4 bat\_tiled.png

现在应该讲讲 TiledTextureRegion 与精灵表的区别了。读者可能比较熟悉其他使用精灵表的游戏引擎，并且将它们当做制作动画效果的范例来学习。精灵表的图像在任何位置、任何方向上均可放置，引擎会提取并使用它们；然而 TiledTextureRegion 的图像则必须大小相同且按照播放动画的顺序放置。截至本书写成时，AndEngine 尚未支持使用精灵表，播放动画所需的帧都是从 TiledTextureRegion 对象中选取的。

## 6.3 AndEngine 的动画

AndEngine 中有专门为动画精灵设置的名为 AnimatedSprite 的类，它从 TiledTextureRegion 对象中提取动画的帧。除此之外，它与其他 Sprite 的行为相同，所有第 4 章所述的修改器均可应用于它。

### AnimatedSprite 类

AnimatedSprite 有 4 个构造器：

```
AnimatedSprite(final float pX, final float pY,
               final TiledTextureRegion pTiledTextureRegion)
```



```

AnimatedSprite(final float pX, final float pY, final float pTileWidth,
               final float pTileHeight, final TiledTextureRegion pTiledTextureRegion)
AnimatedSprite(final float pX, final float pY,
               final TiledTextureRegion pTiledTextureRegion,
               final RectangleVertexBuffer pRectangleVertexBuffer)
AnimatedSprite(final float pX, final float pY, final float pTileWidth,
               final float pTileHeight, final TiledTextureRegion pTiledTextureRegion,
               final RectangleVertexBuffer pRectangleVertexBuffer)

```

其参数意义如下:

- float pX 与 float pY: Sprite 对象的位置。正如第 5 章所见, 其位置是相对于上级对象的。
- TiledTextureRegion pTiledTextureRegion: 存放精灵的动画帧的 TiledTextureRegion 对象。与普通的 Sprite 对象一样, 也是需要通过 TextureRegionFactory 用一张图片创建 TextureRegion 对象, 并载入到 Texture 中去, 然后再用 TextureManager 将 Texture 载入缓存中。
- RectangleVertexBuffer pRectangleVertexBuffer: 像第 5 章讲 Sprite 类时说到的那样, 该参数是 OpenGL 的数据结构, 用来修改精灵显示的方式。
- float pTileWidth 与 float pTileHeight: 可以指定 TiledTextureRegion 对象中每个瓦片图的大小, 如果不指定的话, AndEngine 将使用 TiledTextureRegion 的默认设定。

### 动画相关方法

AnimatedSprite 类在 Sprite 的基础上增加了一系列用于控制动画的方法。播放动画的方式是先用上述构造器创建 AnimatedSprite 对象, 然后调用其 animate() 方法来启动动画效果。为了方便讨论, 可将 animate() 方法分为两组: 每帧持续时间相等的为一组, 不等的为另一组。先来看播放每帧持续时间恒定的动画所用的方法:

```

AnimatedSprite animate(final long pFrameDurationEach)
AnimatedSprite animate(final long pFrameDurationEach, final boolean pLoop)
AnimatedSprite animate(final long pFrameDurationEach, final int pLoopCount)
AnimatedSprite animate(final long pFrameDurationEach, final boolean pLoop,
                       final IAnimationListener pAnimationListener)
AnimatedSprite animate(final long pFrameDurationEach, final int pLoopCount,
                       final IAnimationListener pAnimationListener)

```

通用的参数意义如下:

- long pFrameDurationEach: 该参数指定了每帧播放的时长, 以毫秒为单位。注意, 该设定仅仅是期望值, 意为请求设备以相应的帧速率播放动画。根据运行平台的实际资源状况, 有时可能无法达到预期的帧速率。



- ❑ boolean pLoop : 如果值为 true, 动画会持续播放, 如果是 false 则只播放一次。默认是 true。
- ❑ int pLoopCount: 如果需要动画播放指定的次数, 则将其值通过此参数传入。
- ❑ IAnimationListener pAnimationListener: IAnimationListener 接口实现了 IOnAnimationListener 接口所定义的单一回调方法, 该方法会在动画播放完毕时调用:

```
void onAnimationEnd(final AnimatedSprite pAnimatedSprite)
```

如果想让每帧持续不同的时间, 则可以使用如下一组类似的方法, 传入表示每帧持续时间的数组。该组方法还可以更加灵活地使用瓦片贴图数组。如果 pFrameDuration 的长度与精灵所含帧数不相等, 则会抛出异常。

```
AnimatedSprite animate(final long[] pFrameDurations)-
AnimatedSprite animate(final long[] pFrameDurations, final boolean pLoop)
AnimatedSprite animate(final long[] pFrameDurations, final int pLoopCount)
AnimatedSprite animate(final long[] pFrameDurations, final boolean pLoop,
    final IAnimationListener pAnimationListener)
AnimatedSprite animate(final long[] pFrameDurations, final int pLoopCount,
    final IAnimationListener pAnimationListener)
AnimatedSprite animate(final long[] pFrameDurations, final int pFirstTileIndex,
    final int pLastTileIndex, final boolean pLoop)
AnimatedSprite animate(final long[] pFrameDurations, final int pFirstTileIndex,
    final int pLastTileIndex, final int pLoopCount)-
AnimatedSprite animate(final long[] pFrameDurations, final int[] pFrames,
    final int pLoopCount)
```

以上方法允许指定在 TiledTextureRegion 中所使用的第一张及最后一张瓦片贴图。该功能在需要播放动画的某一小部分或瓦片贴图未能填满整个 TiledTextureRegion 时会非常有用。瓦片贴图的帧号从 0 开始, 最后一张帧号是总瓦片数减 1。

### 其他方法

AnimatedSprite 也提供了两个用于中止动画的方法 (如果通过 pAnimationListener 参数注册了监听器的话, 它也会在停止时被调用):

```
void stopAnimation()
void stopAnimation(final int pTileIndex)
```

第一个方法使动画停止在当前帧, 第二个方法则使其停止在指定的帧上。

## 6.4 动画范例

为了演示蝙蝠的动画效果, 需要将其添加至在游戏开始时的启动画面上。新的启动画面如图 6.5 所示, 该图像是静止的截屏, 实际运行游戏时可以看到蝙蝠在上下拍打翅膀。

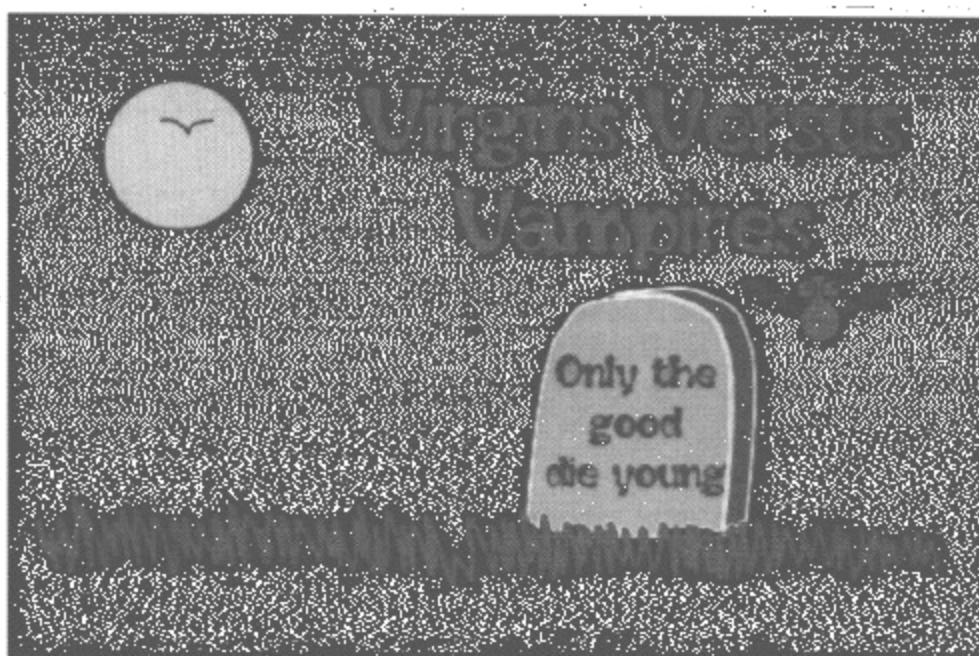


图 6.5 带有飞行蝙蝠的启动画面

程序清单 6.1 是加入了飞行蝙蝠的新版 StartActivity.java 文件。

程序清单 6.1 加入蝙蝠动画的 StartActivity.java 文件

```
package com.pearson.lagp.v3;

import org.anddev.andengine.engine.Engine;
import org.anddev.andengine.engine.camera.Camera;
import org.anddev.andengine.engine.options.EngineOptions;
import org.anddev.andengine.engine.options.EngineOptions.ScreenOrientation;
import org.anddev.andengine.engine.options.resolutionpolicy
    .RatfoResolutionPolicy;
import org.anddev.andengine.entity.scene.Scene;
import org.anddev.andengine.entity.sprite.AnimatedSprite;
import org.anddev.andengine.entity.sprite.Sprite;
import org.anddev.andengine.entity.util.FPSLogger;
import org.anddev.andengine.opengl.texture.Texture;
import org.anddev.andengine.opengl.texture.TextureOptions;
import org.anddev.andengine.opengl.texture.region.TextureRegion;
import org.anddev.andengine.opengl.texture.region.TextureRegionFactory;
import org.anddev.andengine.opengl.texture.region.TiledTextureRegion;
import org.anddev.andengine.ui.activity.BaseGameActivity;
import android.content.Intent;
import android.os.Handler;

public class StartActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
```

```

private static final int CAMERA_HEIGHT = 320;

// =====
// 字段
// =====

private Camera mCamera;
private Texture mTexture, mBatTexture;
private TextureRegion mSplashTextureRegion;
private TiledTextureRegion mBatTextureRegion;
private Handler mHandler;

// =====
// 构造器
// =====

// =====
// Getter 和 Setter 方法
// =====

// =====
// 覆写超类及接口中的方法
// =====

@Override
public Engine onLoadEngine() {
    mHandler = new Handler();
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH, CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH, CAMERA_HEIGHT),
        this.mCamera));
}

@Override
public void onLoadResources() {
    TextureRegionFactory.setAssetBasePath("gfx/Splash/");
    this.mTexture = new Texture(512, 1024,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mSplashTextureRegion =
        TextureRegionFactory.createFromAsset(this.mTexture,
            this, "Splashscreen.png", 0, 0);
    this.mBatTexture = new Texture(256, 256,
        TextureOptions.DEFAULT);
    this.mBatTextureRegion =
        TextureRegionFactory.createTiledFromAsset(
            this.mBatTexture, this, "bat_tiled.png", 0, 0, 2,
            2);
    this.mEngine.getTextureManager().loadTexture(this.mTexture);
    this.mEngine.getTextureManager().loadTexture(this.mBatTexture);
}

```

```

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    final Scene scene = new Scene(1);

    /* 将摄像头置于启动画面中央。 */
    final int centerX = (CAMERA_WIDTH -
        this.mSplashTextureRegion.getWidth()) / 2;
    final int centerY = (CAMERA_HEIGHT -
        this.mSplashTextureRegion.getHeight()) / 2;

    /* 创建表示背景的 Sprite 对象并将其加入 Scene 对象中。 */
    final Sprite splash = new Sprite(centerX, centerY,
        this.mSplashTextureRegion);
    scene.getLastChild().attachChild(splash);

    /* 创建表示蝙蝠动画的 Sprite 对象并将其加入 Scene 对象中。 */
    final AnimatedSprite bat = new AnimatedSprite(350, 100,
        this.mBatTextureRegion);
    bat.animate(100);
    scene.getLastChild().attachChild(bat);
    return scene;
}

@Override
public void onLoadComplete() {
    mHandler.postDelayed(mLaunchTask, 5000);
}

private Runnable mLaunchTask = new Runnable() {
    public void run() {
        Intent myIntent = new Intent(StartActivity.this,
            MainMenuActivity.class);
        StartActivity.this.startActivity(myIntent);
    }
};

// =====
// 方法
// =====

// =====
// 内部类与匿名类
// =====
}

```

关键的代码修改在 `onLoadResources()`、`onLoadScene()` 及 `onLoadComplete()` 方法之中。

### onLoadResources()

- mBatTexture：这里引入的另一个 Texture 对象，用以载入保存蝙蝠动画帧的 TiledTextureRegion 对象。瓦片图大小是 100 像素 × 100 像素，故选择 Texture 的尺寸为更大一点的 2 的整数次幂。
- mBatTiledTextureRegion：用代表动画帧的瓦片图创建此 TextureRegion 对象，它共有 2 行 2 列。
- 最后依然用 TextureManager 将新的 Texture 对象载入缓存。

### onLoadScene()

- bat：创建了用于显示蝙蝠动画的 AnimatedSprite 对象，将其放置在背景图中的墓碑旁。
- 让蝙蝠的每个动画帧播放 100 毫秒（即每秒 10 帧），以造成运动效果。
- 将蝙蝠对象加入到 Scene 对象中，这样它就显示出来了。

### onLoadComplete()

- 将启动画面的停留时间由 3 秒增加到 5 秒，以便有更长的时间来观察蝙蝠动画。

AndEngine 显示动画图片所使用的 TiledTextureRegion 对象要求所有瓦片贴图必须大小相同，而且最好是完全填满整个对象的区域。当显示动画时，将按照从左至右、由上到下的顺序依次显示每张瓦片贴图，并在到达最后一张后折回第一张。在调用简单形式的 createTiledFromAsset()、createTiledFromResource() 与 createTiledFromSource() 方法时，并不需要告知 AndEngine 瓦片贴图的大小，只需要简单地说明要创建的 TiledTextureRegion 对象共有几行几列即可，AndEngine 会根据这些值平均地将大图切割为小瓦片贴图。使用更为复杂的 createTileFromXX() 方法可以指定每个瓦片贴图的大小，不过这样需要在运行时进行更多的运算。

## 6.5 将动画加入 Level1Activity 类

现在回到游戏第 1 关的制作上来。需要给屏幕右方增加一些带有动画效果的吸血鬼作为敌人，让他们向左移动。此时还不需要检测碰撞，所以让这些坏家伙一直走到屏幕左方并聚集在那里就好。随机生成 10 个这种角色并让其以任意路线穿过墓地。

首先，需要做出敌人动画的每一帧来，笔者承认在这个问题上走了个捷径：在 Anime Studio 网站上以 12 美元购买了动画素材，用该软件在黑色背景上播放动画并录制为 QuickTime 格式（文件后缀名 .mov）的电影文件，再用 AnimGet 软件将电影捕捉成帧。该动画的细致程度远远超出了 V3 游戏所需标准，不过没关系，笔者用 GIMP 将其缩小至 60 像素 × 60 像素大小，为每帧配上了透明背景，并反转了图像。因为原图像

恰好是从左向右走的，而本游戏中所有精灵图像都是从右向左走的。

用 GIMP 将所有的图像拼接成如图 6.6 所示的精灵表。



图 6.6 排列好的瓦片贴图

程序清单 6.2 是修改之后的 Level1Activity.java 文件。

程序清单 6.2 加入了动画吸血鬼之后的 Level1Activity.java 文件

```
package com.pearson.lagp.v3;

import java.util.Arrays;
import java.util.Random;

import org.anddev.andengine.engine.Engine;
import org.anddev.andengine.engine.camera.Camera;
import org.anddev.andengine.engine.options.EngineOptions;
import org.anddev.andengine.engine.options.EngineOptions.ScreenOrientation;
import org.anddev.andengine.engine.options.resolutionpolicy
    .RatioResolutionPolicy;
import org.anddev.andengine.entity.modifier.AlphaModifier;
import org.anddev.andengine.entity.modifier.DelayModifier;
import org.anddev.andengine.entity.modifier.FadeInModifier;
import org.anddev.andengine.entity.modifier.MoveModifier;
import org.anddev.andengine.entity.modifier.MoveYModifier;
import org.anddev.andengine.entity.modifier.ParallelEntityModifier;
import org.anddev.andengine.entity.modifier.RotationModifier;
import org.anddev.andengine.entity.modifier.ScaleModifier;
import org.anddev.andengine.entity.modifier.SequenceEntityModifier;
import org.anddev.andengine.entity.scene.Scene;
import org.anddev.andengine.entity.sprite.AnimatedSprite;
import org.anddev.andengine.entity.sprite.Sprite;
import org.anddev.andengine.entity.util.FPSLogger;
```



```

import org.anddev.andengine.opengl.texture.BuildableTexture;
import org.anddev.andengine.opengl.texture.Texture;
import org.anddev.andengine.opengl.texture.TextureOptions;
import org.anddev.andengine.opengl.texture.builder.BlackPawnTextureBuilder;
import org.anddev.andengine.opengl.texture.builder.ITextureBuilder-
    .TextureSourcePackingException;
import org.anddev.andengine.opengl.texture.region.TextureRegion;
import org.anddev.andengine.opengl.texture.region.TextureRegionFactory;
import org.anddev.andengine.opengl.texture.region.TiledTextureRegion;
import org.anddev.andengine.ui.activity.BaseGameActivity;
import org.anddev.andengine.util.modifier.ease.EaseQuadOut;

import android.content.Intent;
import android.os.Handler;
import android.util.Log;
public class Level1Activity extends BaseGameActivity {
    // =====
    // 常量
    // =====
    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;
    private String tag = "Level1Activity";

    // =====
    // 字段
    // =====

    private Handler mHandler;

    protected Camera mCamera;

    protected Scene mMainScene;

    private Texture mLevel1BackTexture;
    private Texture mScrumTexture;
    private BuildableTexture mObstacleBoxTexture;
    private TextureRegion mBoxTextureRegion;
    private TextureRegion mLevel1BackTextureRegion;
    private TextureRegion mBulletTextureRegion;
    private TextureRegion mCrossTextureRegion;
    private TextureRegion mHatchetTextureRegion;
    private TiledTextureRegion mScrumTextureRegion;

    private AnimatedSprite[] asprVamp = new AnimatedSprite[10];
    private int nVamp;
    Random gen;

    // =====
    // 构造器
    // =====

```

```
// =====
// Getter 和 Setter 方法
// =====

// =====
// 覆写超类及接口中的方法
// =====
```

```
@Override
public Engine onLoadEngine() {
    mHandler = new Handler();
    gen = new Random();
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT),
        this.mCamera));
}
```

```
@Override
public void onLoadResources() {
    /* 载入 Texture 对象。 */
    TextureRegionFactory.setAssetBasePath("gfx/Level1/");
    mLevel1BackTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mLevel1BackTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mLevel1BackTexture, this, "level1bk.png", 0,
            0);
    mEngine.getTextureManager().loadTexture(
        this.mLevel1BackTexture);

    mObstacleBoxTexture = new BuildableTexture(512, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mBoxTextureRegion =
        TextureRegionFactory.createFromAsset(
            mObstacleBoxTexture,
            this, "obstaclebox.png");
    mBulletTextureRegion =
        TextureRegionFactory.createFromAsset(
            mObstacleBoxTexture,
            this, "bullet.png");
    mCrossTextureRegion =
        TextureRegionFactory.createFromAsset(
            mObstacleBoxTexture,
            this, "cross.png");
    mHatchetTextureRegion =
        TextureRegionFactory.createFromAsset(
            mObstacleBoxTexture,
```

```

        this, "hatchet.png");
    try {
        mObstacleBoxTexture.build(
            new BlackPawnTextureBuilder(2));
    } catch (final TextureSourcePackingException e) {
        Log.d(tag,
            "Sprites won't fit in mObstacleBoxTexture");
    }
    this.mEngine.getTextureManager().loadTexture(
        this.mObstacleBoxTexture);

    mScrumTexture = new Texture(512, 256,
        TextureOptions.DEFAULT);
    mScrumTextureRegion =
        TextureRegionFactory.createTiledFromAsset(
            mScrumTexture,
            this, "scrum_tiled.png", 0, 0, 8, 4);
    mEngine.getTextureManager().loadTexture(
        this.mScrumTexture);
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    final Scene scene = new Scene(1);

    /* 将摄像头置于屏幕中央。 */
    final int centerX = (CAMERA_WIDTH -
        mLevel1BackTextureRegion.getWidth()) / 2;
    final int centerY = (CAMERA_HEIGHT -
        mLevel1BackTextureRegion.getHeight()) / 2;

    /* 创建精灵并将其加入 Scene 对象。 */
    final Sprite background = new Sprite(centerX, centerY,
        mLevel1BackTextureRegion);
    scene.getLastChild().attachChild(background);
    final Sprite obstacleBox = new Sprite(0.0f, CAMERA_HEIGHT -
        mBoxTextureRegion.getHeight(), mBoxTextureRegion);
    scene.getLastChild().attachChild(obstacleBox);
    final Sprite bullet = new Sprite(20.0f, CAMERA_HEIGHT -
        40.0f,
        mBulletTextureRegion);
    bullet.registerEntityModifier(
        new SequenceEntityModifier(
            new ParallelEntityModifier(
                new MoveYModifier(3, 0.0f, CAMERA_HEIGHT -
                    40.0f, EaseQuadOut.getInstance()),
                new AlphaModifier(3, 0.0f, 1.0f),
                new ScaleModifier(3, 0.5f, 1.0f)
            )
        )
    );
}

```

```

    ),
    new RotationModifier(3, 0, 360)
);
scene.getLastChild().attachChild(bullet);
final Sprite cross = new Sprite(bullet.getInitialX() +
    40.0f,
    CAMERA_HEIGHT - 40.0f, mCrossTextureRegion);
cross.registerEntityModifier(
    new SequenceEntityModifier(
        new ParallelEntityModifier(
            new MoveYModifier(4, 0.0f, CAMERA_HEIGHT -
                40.0f, EaseQuadOut.getInstance()),
            new AlphaModifier(4, 0.0f, 1.0f),
            new ScaleModifier(4, 0.5f, 1.0f)
        ),
        new RotationModifier(2, 0, 360)
    )
);
cross.registerEntityModifier(new AlphaModifier(10.0f, 0.0f,
    1.0f));
scene.getLastChild().attachChild(cross);
final Sprite hatchet = new Sprite(cross.getInitialX() +
    40.0f,
    CAMERA_HEIGHT - 40.0f, mHatchetTextureRegion);
hatchet.registerEntityModifier(
    new SequenceEntityModifier(
        new ParallelEntityModifier(
            new MoveYModifier(5, 0.0f, CAMERA_HEIGHT -
                40.0f, EaseQuadOut.getInstance()),
            new AlphaModifier(5, 0.0f, 1.0f),
            new ScaleModifier(5, 0.5f, 1.0f)
        ),
        new RotationModifier(2, 0, 360)
    )
);
hatchet.registerEntityModifier(new AlphaModifier(15.0f,
    0.0f,
    1.0f));
scene.getLastChild().attachChild(hatchet);
scene.registerEntityModifier(new AlphaModifier(10, 0.0f,
    1.0f));

    // 加入第一个吸血鬼 (其他的也会随之加入)
    nVamp = 0;
    mHandler.postDelayed(mStartVamp, 5000);
    return scene;
}

@Override
public void onLoadComplete() {

```

```

    }
    private Runnable mStartVamp = new Runnable() {
        public void run() {
            int i = nVamp++;
            Scene scene = Level1Activity.this.mEngine.getScene();
            float startY = gen.nextFloat()*(CAMERA_HEIGHT - 50.0f);
            asprVamp[i] = new AnimatedSprite(CAMERA_WIDTH - 30.0f,
            startY,
            mScrumTextureRegion.clone());
            final long[] frameDurations = new long[26];
            Arrays.fill(frameDurations, 500);
            asprVamp[i].animate(frameDurations, 0, 25, true);
            asprVamp[i].registerEntityModifier(
                new SequenceEntityModifier (
                    new AlphaModifier(5.0f, 0.0f, 1.0f),
                    new MoveModifier(60.0f,
            asprVamp[i].getX(), 30.0f,
                    asprVamp[i].getY(),
                    (float)CAMERA_HEIGHT/2));
            scene.getLastChild().attachChild(asprVamp[i]);
            if (nVamp < 10){
                mHandler.postDelayed(mStartVamp,5000);
            }
        }
    };
}

```

运行游戏之后，可以看见吸血鬼们一个个冒出来，向屏幕左边移动，很像电影《Animal House》<sup>①</sup>中挤成一团的乐队。注意，不管他们走到哪里，总是可见的，因为他们是最后加入 Scene 对象的物体，所以显示时会出现在图层的最上方。

现在来仔细看看代码吧。一开始定义了很多新的变量：

- ❑ Handler mHandler：用以投递 Runnable 任务的 Android Handler 对象，如果读者对此不熟悉的话，请看接下来的注解。
- ❑ Texture mScrumTexture：新创建的用于保存动画帧的 Texture 对象。
- ❑ TiledTextureRegion mScrumTextureRegion：mScrumTexture 所含的贴图对象。
- ❑ AnimatedSprite[] asprVamp[10]：代表吸血鬼精灵的 AnimatedSprite 对象数组。
- ❑ int nVamp：吸血鬼计数器。
- ❑ Random gen：随机数生成器。

### 注解 Android Handler 与 Runnable

Android 应用程序的运行是基于名为“main”或者“UI”的线程的，这些线程的核

① 中文名《动物屋》，1978 年美国喜剧电影。讲述了某大学中两群不服学校管教的富人子弟组织之间的事。——译者注

心是消息循环。Handler对象可以用来同消息队列沟通，它可以向线程投递 Message 或 Runnable 对象，使其即刻或延迟一定时间后执行。

该机制在 Android 应用程序开发中大量运用，V3 游戏中也有两处用到它：

1. 在 `StartActivity` 与 `MainMenuActivity` 类中，用 Handler 触发其他的 Activity。代码通过 Handler 向线程投递了 Runnable 对象，其以 Intent 对象为参数，执行 `startActivity()` 方法。Intent 对象描述了将要运行的另一个 Activity 对象。

2. `Level1Activity` 使用 Handler 来延迟某段代码的执行。如果需要让某段代码延迟执行，不能强制 Android 消息循环停下来等待至指定时间再继续，而是要将该段代码封装入 Runnable 的 `run()` 方法中，再将该 Runnable 对象投递到执行线程。Handler 对象会把这个 Runnable 对象安排到消息队列中去，以便 Android 系统延时后执行。

### `onLoadEngine()`

该方法仅做了少量修改：

- `Handler mHandler`：用以延时执行 `mStartVampire` 任务的 Handler 对象。
- `Random gen`：初始化随机数生成器，以使用其生成摇摇摆摆的吸血鬼。

### `onLoadResources()`

为了载入吸血鬼动画所用贴图，对代码进行了如下修改：

- `Texture mScrumTexture`：为了载入动画贴图而新建的 Texture 对象。
- `TiledTextureRegion mScrumTextureRegion`：使用该对象从位图文件 `assets/gfx/Level1/scrum_tiled.png` 中创建并保存动画所需瓦片贴图。因为我们正创建的是 `TiledTextureRegion`，我们使用 `createTiledFromAsset()` 方法并以贴图图像中行和列的数目作为参数。
- 照例使用 `TextureManager` 将 Texture 对象载入缓存。

### `onLoadScene()`

“加入第一个吸血鬼”这句注释之前的部分没有改动，之后的改动如下。

- `int nVamp`：使用该计数器追踪当前的吸血鬼数量。代码将其值初始化为 0。
- 通过 Handler 向线程投递 `mStartVampire` 所表示的 Runnable 任务，令其在 5 秒种后执行。此后每隔 5 秒执行一次该任务，直至生成 10 个吸血鬼为止。

### `mStartVampire`

该 Runnable 对象的 `run()` 方法会创建吸血鬼对象并令其在屏幕上走动，之后投递下一个生成吸血鬼的请求。



- `int i`：将计数器的值拷贝至该变量，然后为计数器加 1。使用该变量的主要原因是后面的代码需要多次用到 `nVamp` 的值，用另一个名为 `i` 的变量代表它，可以减少打字量。
- `Scene scene`：因为本方法无法访问当前的 `Scene` 对象，故而代码通过 `Engine` 对象获取了它。
- `float startY`：使用随机数生成器计算出吸血鬼的产生地点。吸血鬼会从屏幕右侧的任意 `Y` 坐标处生成。
- `AnimatedSprite asprVamp[i]`：在计算出的位置生成吸血鬼精灵对象。在创建它时，使用了从 `mScrumTextureRegion` 对象中新复制出来的另一份拷贝。复制操作很重要，如果不使用复制之后的贴图对象的话，那么所有创建出来的吸血鬼将以相同的步伐移动，这显然不是预期的游戏效果。通过使用复制之后的 `TiledTextureRegion` 对象，每个吸血鬼都会播放自己的动画，互不干扰。
- `asprVamp[i].animate`：回忆一下图 6.6，`scrum_tiled.png` 文件中，所有瓦片贴图并未填满整个图片区域。所以要用这种版本的 `.animate()` 方法，它可以通过第 2 个参数 `pFirstTileIndex` 与第 3 个参数 `pLastTileIndex` 来指定第一张与最后一张瓦片的位置索引，该方法还需要通过 `pFrameDurations` 参数传入一个数组。于是代码先创建数组，并用相同的值将其所有元素填充。这个值即是每帧持续的毫秒数。最后一个参数取值为 `true` 的意思是，该动画将循环播放。
- 注册一些吸血鬼的修改器，使吸血鬼淡入并行走到屏幕左侧中部（Miss B 门前）
- 将吸血鬼精灵加入当前 `Scene` 对象中。
- 如果当前吸血鬼数量不足 10 个，则请求 Android 系统在 5 秒钟之后再次执行本任务以生成另一个吸血鬼。

## 6.6 动画制作的问题

想做好的动画是非常困难的，它是一门艺术。即使新的工具能够使生成动画的过程更加简单、快捷，要想做出流畅且逼真的动画效果，依然要花费大量的时间与精力。

因特网上可以找到大量的材料，教你如何做出好的动画，如何修改有问题的动画。如果读者做出的角色动作很不自然，达不到想要的效果，并且不知道如何做才对的话，不妨花几分钟时间研究一下动画制作的技巧与窍门，仔细修改每一帧动画。

如果读者有能力使用动画制作软件的话，这将极大地推进游戏制作的进度。将一副原始图画纳入游戏项目中需要很多步骤：渲染、帧捕捉、插入透明背景、缩放、载入

Eclipse 项目等等，在这个过程中应尽早发现并修复动画制作中存在的问题。

## 6.7 高级话题：从 3D 模型中制作 2D 动画

据说，由于 3D 游戏及 3D 插画的流行，3D 游戏美工的数量会超过 2D 美工。笔者不能肯定这一点，然而使用 3D 模型来生成 2D 动画效果确实有其优势。游戏中需要角色在不同方向上移动时的动画序列，有时角色可能从左向右移动，有时需要向远离屏幕的方向移动，有时又需要朝玩家走来。如果进行 2D 绘图的话，需要做出 3 个方向的动画序列（从右向左移动的序列可以通过反转从左向右的移动序列得到）。

如果使用 3D 模型的话，可以轻而易举地移动角色或摄像机的位置，这样就可以从任意角度生成 2D 动画序列了：只需要将每个方向的动画生成电影文件，再用 AnimGet 将动画帧捕捉下来即可。

尽管 3D 动画制作软件的学习曲线很陡，但是如果读者或其朋友会使用此类软件的话，则可以考虑用它来制作游戏角色的动画序列。

## 6.8 总结

本章讲述了制作 V3 游戏动画效果需要的所有知识。尽管本书不是关于动画制作艺术的专著，不过仍然讲述了几种用于制作动画序列、将其载入 TiledTextureRegion 以及调用 animate() 方法播放动画的方式。

现在简述一下制作动画精灵的过程：

- 将含有瓦片贴图的文件载入 TiledTextureRegion 对象中，并将其载入 Texture 对象。用 TextureManager 将 Texture 载入缓存中。
- 创建 AnimatedSprite 对象，如果要多个精灵具有各自不同的动画效果，注意创建时要使用复制过的 TiledTextureRegion 对象。
- 调用合适的 animate() 方法以播放动画。
- 将精灵加入到 Scene 对象中。

在学会了如何为游戏角色制作能够移动并具有动画效果的精灵之后，接下来看一看如何在游戏中绘制文本。

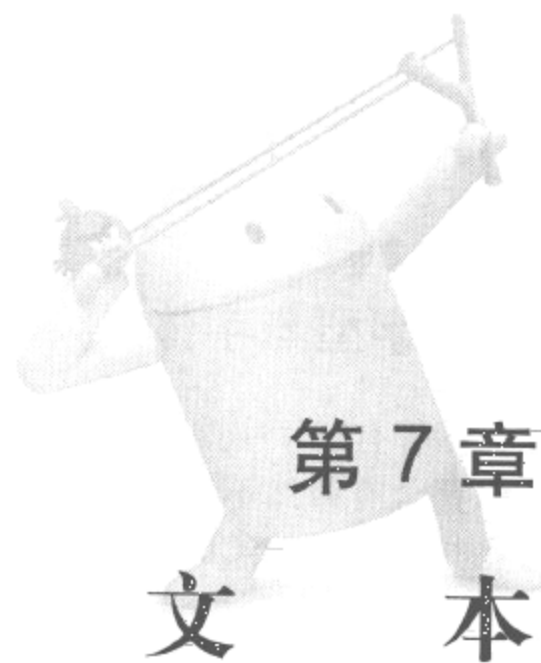
## 6.9 习题

1. 让 V3 游戏启动画面上的蝙蝠在墓碑前面来回飞行，而不要一直悬停在墓碑上面。如果要让它在墓碑后面飞行，又该如何修改？

2. 绘制某个角色走路的动画效果。如果您和笔者一样不太精通美术的话，用简笔画来代替也无所谓。关键是要学会如何制作动画帧，将其调整为适当的格式并在游戏中使用。

3. 使用动画制作软件重做习题 2。很多公司都提供限时免费的试用版软件，看看用它能够节省多少制作动画所需的时间。在纯手工绘制与使用工具制作过渡动画帧之间，能否找到平衡点？

4. 修改 `Level1Activity`，让吸血鬼到达 Miss B 的家门口后消失。



- 7.1 字型与字体
- 7.2 载入字型
- 7.3 AndEngine 中的文本
- 7.4 定制字型
- 7.5 将定制字型加入 V3
- 7.6 总结
- 7.7 习题

第3章创建文本菜单时用到了文本，本章将更加细致地讲述 AndEngine 中文本的使用，包括字体、使用文本的游戏元素以及如何创建并使用开发者自制的字体。

## 7.1 字型与字体

尽管有时“字型”(font)与“字体”(typeface)这两个词可以互换使用，但是它们毕竟含有截然不同的意义。字体类似 Arial 或 Droid Sans 这样，定义了某个字符集并将其映射到 ASCII 或 Unicode 码。字型则包含了字体及其大小（例如 16 点，1 点即 1/72 英寸）、风格（例如粗体或斜体）等属性。

有三种方式可以用来生成字符的物理表征：

- 位图 (bitmap)：位图字型中，每个字符都是以图片表示的。因为位图缩放时会失真，所以通常每种字体大小都要配一套图，这就导致了位图字型很大且很占内存。
- 轮廓 (outline) 或 矢量 (vector)：在矢量字型中，每个字符均以矢量信息表示，这样缩放时不会失真。仅保存矢量信息意味着矢量字型占用的空间比较小。
- 笔画 (stroke)：对于很多字型作者来说，它同矢量字型很相似（AndEngine 中不是这样）。此种字型将字符定义为一系列笔画的集合，所以特别适用于基于笔画的东亚语言，它可大大降低含有 5000 多字的中文字符所占的空间。

和图形类似，矢量字体可以很容易缩放到不同的大小（尤其是比较大时），而位图字型则可以很好地根据不同的大小进行精确的剪裁。AndEngine 支持矢量字型，不过大家看到的 StrokeFont 指的就是轮廓矢量字型，而非上述所说的笔画字型。开发者可以自己定义新类以支持位图及轮廓字型，不过这不在本书讨论范围内。

Android 系统渲染文本时均使用矢量字型，迄今为止，每台设备都预装了三种字体：Droid Sans、Droid Sans Mono 及 Droid Serif。“Sans”意为“sans serif”，即不含衬线。衬线指的是笔画末端装饰性的线条。“Mono”意为“monospaced”，即等宽字体。此种字体每个字符宽度相等，非常像打字机打出来的那样。图 7.1 展示了三种标准的 Android 系统字体。

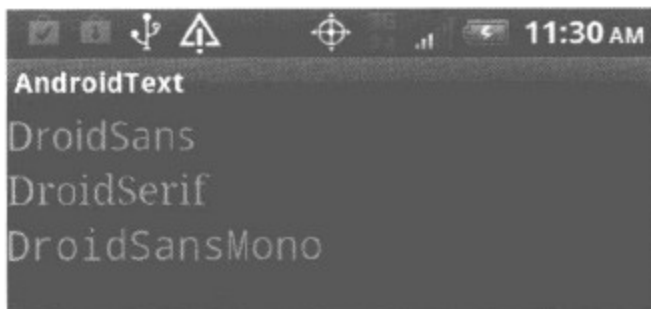


图 7.1 Android 系统标准字体

Android 系统也允许开发者载入定制的字体与字型，它支持 TrueType 及 OpenType 格式，不过有人说 OpenType 格式存在一些问题。AndEngine 频繁地使用 Typeface 类，本章稍后将介绍如何使用定制的 TrueType 字型。

## 7.2 载入字型

在创建基于文本的实体之前，需要先载入字型。因为 AndEngine 的所有绘制都使用 OpenGL，所以就算要使用 Android 系统的标准字型，也要在使用前将其载入 Texture 对象。AndEngine 使用 Font 类（与 Android 系统的同名类不同）与 FontManager 类来载入与管理字型。

### 7.2.1 Font 类

AndEngine 中，Font 类是 OpenGL 代码的一部分。对于开发者来说，唯一需要了解的就是其构造器：

```
Font(final Texture pTexture, final Typeface pTypeface,
      final float pSize, final boolean pAntiAlias, final int pColor)
```

参数意义如下：

- ❑ Texture pTexture：表示要将字型图片载入其中的 Texture 对象，下节会详述。
- ❑ Typeface pTypeface：指定字型图片来源的 Android 系统 Typeface 对象。
- ❑ float pSize：以像素为单位的字型高度，不是以点为单位的。
- ❑ boolean pAntiAlias：如果为 true，则在显示时运用抗锯齿效果。如果读者不熟悉抗锯齿的意思，就理解为使字型显示时更加清晰即可。
- ❑ int pColor：显示字型所用的颜色。注意，不能为轮廓线与填充色分别指定颜色，只能使用同一种颜色。

本章稍后的程序清单 7.1 将演示如何使用 Font 类来显示标准的 Android 系统字型。Font 类所显示的字体只有填充色而无轮廓线。

### 7.2.2 StrokeFont 类

AndEngine 提供了名为 StrokeFont 的 Font 子类。该类所绘字型具有轮廓线，同时可用另一种颜色进行填充，或者不填充。其构造器类似 Font 类，其中最完备的一个如下：

```
StrokeFont( final Texture pTexture, final Typeface pTypeface, final float pSize,
            final boolean pAntiAlias, final int pColor, final float pStrokeWidth,
            final int pStrokeColor, final boolean pStrokeOnly)
```

未出现在 Font 类构造器中的参数如下：

- ❑ float pStrokeWidth：该参数指定了笔画的宽度。
- ❑ int pStrokeColor：笔画颜色。
- ❑ boolean pStrokeOnly：为 true 意味着仅绘制轮廓线（笔画）而不填充。该参数是



可选的，其默认值为 false。

### 7.2.3 FontFactory 类

除了调用 Font 及 StrokeFont 的构造器外，还有一创建类的实例的方式，就是使用 FontFactory 类的 create 方法。其中用途最广的两个如下：

```
Font createFromAsset(final Texture pTexture, final Context pContext,
    final String pAssetPath, final float pSize, final boolean pAntiAlias,
    final int pColor)
StrokeFont createStrokeFromAsset(final Texture pTexture,
    final Context pContext, final String pAssetPath, final float pSize,
    final boolean pAntiAlias, final int pColor, final int pStrokeWidth,
    final int pStrokeColor, final boolean pStrokeOnly)
```

参数意义与上述构造器相同，不过可以另外指定 assets 文件夹下的资源以 Typeface 形式创建 Font 对象。这些方法可以用来从定制的 Typeface 对象中创建字型。

### 7.2.4 FontManager 类

AndEngine 提供了一个类似 TextureManager 的 FontManager 单例，可以用它管理游戏中的字型库。载入 Font 对象的代码范式如下：

```
this.mEngine.getFontManager().loadFont(this.mFont);
```

mFont 是需要载入的 Font 对象，程序清单 7.1 会演示 FontManager 的用法。

### 7.2.5 Typeface 类

游戏中可以通过 Android 系统的 Typeface 类来访问标准的 Android 字型。在 Android 开发者网站可以找到该类的详尽文档，不过我们还是快速浏览一下将会用到的方法与常量：

```
Typeface.create(Typeface family, int style)
```

当需要使用 Android 系统的默认字型时，可以用该方法创建。

□ family 参数可以取下列值之一：

- Typeface.DEFAULT
- Typeface.DEFAULT\_BOLD
- Typeface.MONOSPACE
- Typeface.SANS\_SERIF
- Typeface.SERIF

□ style 参数可以取下列值之一：

- ☐ Typeface.NORMAL
- ☐ Typeface.BOLD
- ☐ Typeface.ITALIC
- ☐ Typeface.BOLD\_ITALIC

## 7.3 AndEngine 中的文本

除了第3章介绍过的 MenuItem 之外，AndEngine 有三种使用文本的方式：

- ☐ Text：包含固定消息的标签。
- ☐ ChangeableText：包含可变消息的标签。
- ☐ TickerText：以逐个字母显示方式的固定内容标签。它不像电报纸条那样滚动，不过效果也很有趣。

当然也可以使用 Android 系统自带的 View 类显示文本，特别是 Toast 类，AndEngine 也经常用它来显示桌面提醒。稍后将会简单地提一下 Toast 类的用法。

### 7.3.1 AndEngine 中的文本 API

Text 对象可用来表示内容不变的文本标签，它有三个构造器，按照详细程度排列如下：

```
Text(final float pX, final float pY, final Font pFont, final String pText)
Text(final float pX, final float pY, final Font pFont, final String pText,
    final HorizontalAlign pHorizontalAlign)
Text(final float pX, final float pY, final Font pFont, final String pText,
    final HorizontalAlign pHorizontalAlign, final int pCharactersMaximum)
```

通用参数意义如下：

- ☐ float pX, pY：标签出现在屏幕上的位置。
- ☐ String pText：文本字符串，可以用“\n”作为换行符来表示多行文本。
- ☐ HorizontalAlign pHorizontalAlign：有三种水平对齐方式，默认是 LEFT 即左对齐：
  - ☐ HorizontalAlign.LEFT
  - ☐ HorizontalAlign.CENTER
  - ☐ HorizontalAlign.RIGHT
- ☐ int pCharactersMaximum：pText 所包含的字符数，不含换行符。前两个构造器将自动计算此参数值。

程序清单 7.1 是个简短的范例，演示了 Text 对象的创建与显示。

程序清单 7.1 演示文本使用的范例代码

```
public class TextExample extends BaseGameActivity {
```

```

// =====
// 常量
// =====

private static final int CAMERA_WIDTH = 720;
private static final int CAMERA_HEIGHT = 480;

// =====
// 字段
// =====

private Camera mCamera;
private Texture mFontTexture, mStrokeFontTexture;
private Font mFont;
private StrokeFont mStrokeFont;

@Override
public Engine onLoadEngine() {
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
}

@Override
public void onLoadResources() {
    this.mFontTexture = new Texture(256, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mStrokeFontTexture = new Texture(256, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mFont = new Font(this.mFontTexture,
        Typeface.create(Typeface.DEFAULT,
            Typeface.BOLD), 32, true, Color.BLACK);
    this.mStrokeFont = new StrokeFont(this.mStrokeFontTexture,
        Typeface.create(Typeface.DEFAULT,
            Typeface.BOLD), 32, true, Color.RED, 2.0f,
        Color.WHITE, true);
    this.mEngine.getTextureManager().loadTexture(

```

```

        this.mFontTexture);
    this.mEngine.getTextureManager().loadTexture(
        this.mStrokeFontTexture);
    this.mEngine.getFontManager().loadFont(this.mFont);
    this.mEngine.getFontManager().loadFont(this.mStrokeFont);
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());
    final Scene scene = new Scene(1);
    scene.setBackground(new ColorBackground(0.1f, 0.6f, 0.9f));
    final Text textCenter = new Text(100, 60, this.mFont,
        "Show this centered \n on two lines.",
        HorizontalAlign.CENTER);
    final Text textStroke = new Text(100, 160,
        this.mStrokeFont,
        "Stroke font example \n also on two lines.",
        HorizontalAlign.CENTER);
    scene.getLastChild().attachChild(textCenter);
    scene.getLastChild().attachChild(textStroke);

    return scene;
}

@Override
public void onLoadComplete() {
}
}

```

以上代码将会制作出图 7.2 所示效果。大部分代码都是不言自明的，只有少数几处需要说明如下：

- 容纳字型贴图的 `mFontTexture` 对象必须足够大，要容纳得下所有载入的字型。如果大小不足，不会报错，但是某些字符在显示时会消失。所需的大小取决于字体及字符大小。这里所使用的 256 像素 × 256 像素，对于 32 像素的 DEFAULT、SANS 及 MONOSPACE 字体来说足够了。然而如果要使用 64 像素的 SERIF 字体，则需要 256 像素 × 512 像素的 Texture 对象（注意 Texture 的长宽必须是 2 的整数幂）。必须按照字型的大小来指定对应的 Texture 对象大小，而且要注意，有些 Android 手机最大只能支持 512 像素 × 512 像素的 Texture。

- ❑ 尽管使用相同的字体与字符大小，但是 Font 与 StrokeFont 对象需要使用不同的 Texture 对象来创建。
- ❑ 创建 mFont 对象时，使用的参数是默认字型（通常是 Droid Sans）、32 像素字符、抗锯齿效果以及黑色填充色。创建 mStrokeFont 时用的参数也是这样，另外指定 pStrokeOnly 为 true，表示只绘制外框线。
- ❑ 先用 TextureManager 载入 Texture 对象，然后用 FontManager 载入 Font 对象。
- ❑ onLoadScene() 方法的代码将文本显示在坐标 (100,60) 处，并使用水平居中对齐。该参数的意思是所有的文本行在水平方向上都放置于文本块的中央，而不是屏幕的中央，如图 7.2 所示。

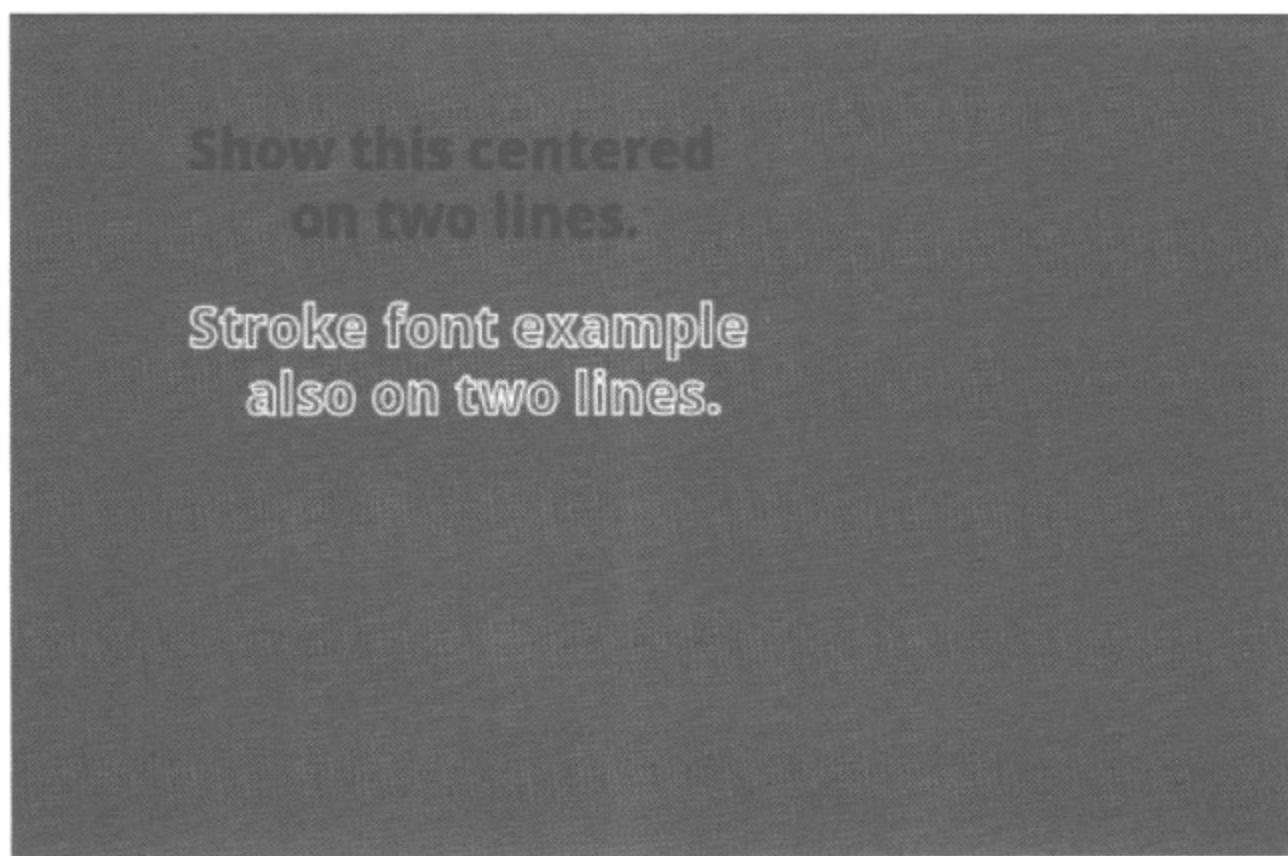


图 7.2 文本显示范例

### 7.3.2 桌面通知

Toast 类是标准的 Android 系统饰件类。这里提到它，主要是因为可以借此迅速地向游戏玩家通知消息。建立一条桌面通知的代码范式是：

```
Toast.makeText(context, text, duration).show();
```

参数意义如下：

- ❑ Context context: 当前的应用程序环境变量（例如 StartActivity.this）。
- ❑ String text: 要显示的文本。

❑ int duration: 该选项可以取以下两个值之一:

❑ Toast.LENGTH\_SHORT (默认值)

❑ Toast.LENGTH\_LONG

如果在上述文本范例代码的 onLoadComplete() 方法中加入桌面通知, 则效果如图 7.3 所示, 通知文本会持续数秒钟。

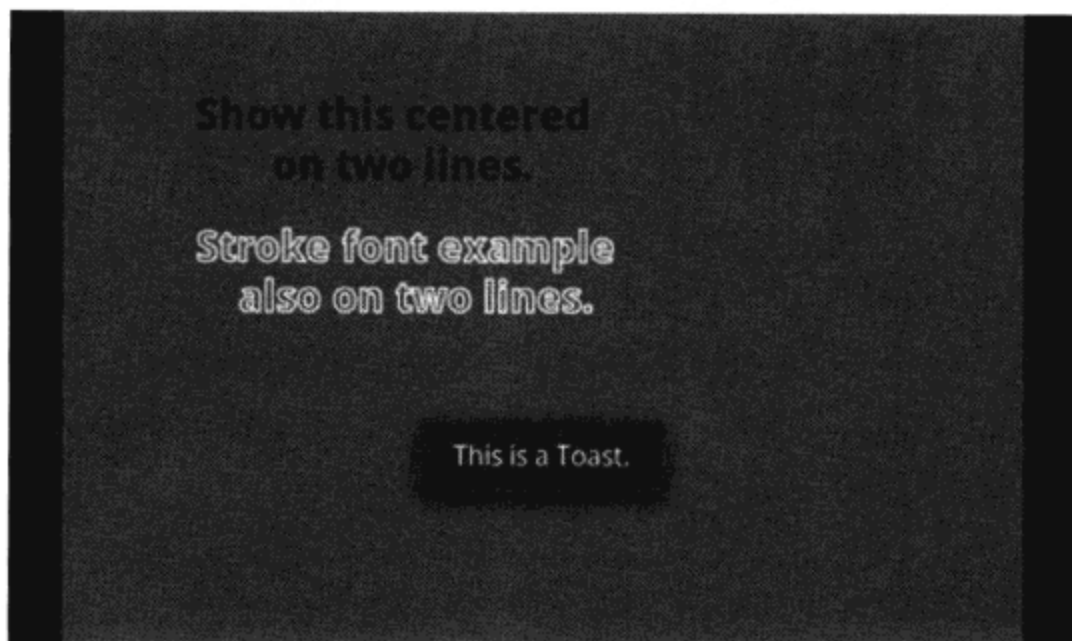


图 7.3 带有桌面通知效果的文本显示范例

## 7.4 定制字型

前文已经讲到, Android 系统自带了几款 TrueType 字型, AndEngine 的范例中也包含了一些, 不过开发者依然可以使用网上找到的甚至自己制作的字型。和其他具有知识产权的物品一样, 如果要在游戏中使用下载的字型, 务必确认已经得到允许使用的授权。如果计划出售游戏, 则要更加小心, 很多免费字体是没有商用授权的。

### 创建自己的 TrueType 字型

V3 游戏中使用了从 [www.1001fonts.com](http://www.1001fonts.com) 网站上下载的 Flubber 字型, 它的授权非常宽松: “您可以任意使用该字型。” 作者 Ben McGehee 仅仅要求将含有其名字的授权文件包含在产品中即可。

该字型已经不需要再进行编辑了, 不过如果需要的话, 有很多编辑器可以编辑 TrueType 字型, 包括一些昂贵的专业字型制作工具, 以及免费或便宜的基本字型编辑工具。

如果需要从头开始制作字型, 而不是修改已有的字型, 那么可以使用第 2 章中提到的 FontStruct 工具 ([www.fontstruct.com](http://www.fontstruct.com)), 它是由 [fontshop.com](http://fontshop.com) 所提供的免费服务。先在网站注册免费账号, 然后就可以用预定义好的构建素材来创建字型了。也可以在授权



许可下，在已有的字型基础上进行编辑，这样产生的字型可以通过创作共用协议授权给他人使用，该授权问题在网站上有详述。图 7.4 中，笔者复制了一款由 Mike Lee 所做的 SwiftedStrokes 字型。

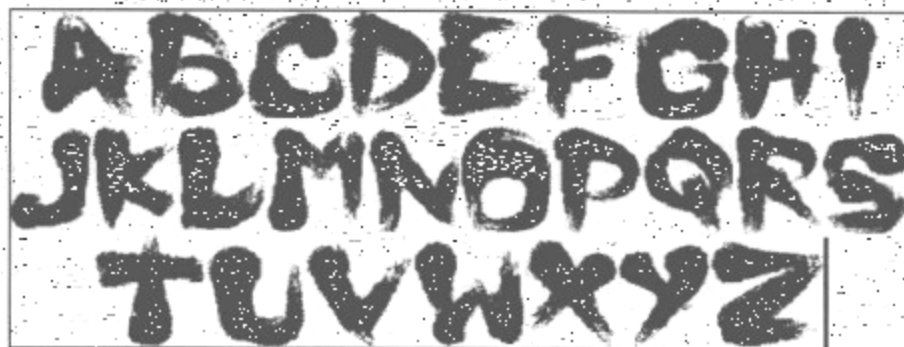


图 7.4 TrueType 字型 SwiftedStrokes 的复制图样

笔者使用 FontStruct 将该字体复制下来，并用此软件自由地编辑它。如图 7.5 所示，正在修改“A”字符，它的每个部分都可以修改，其余的字符也一样。编辑完之后可以将字型文件保存下来自己使用或者分享给他人。

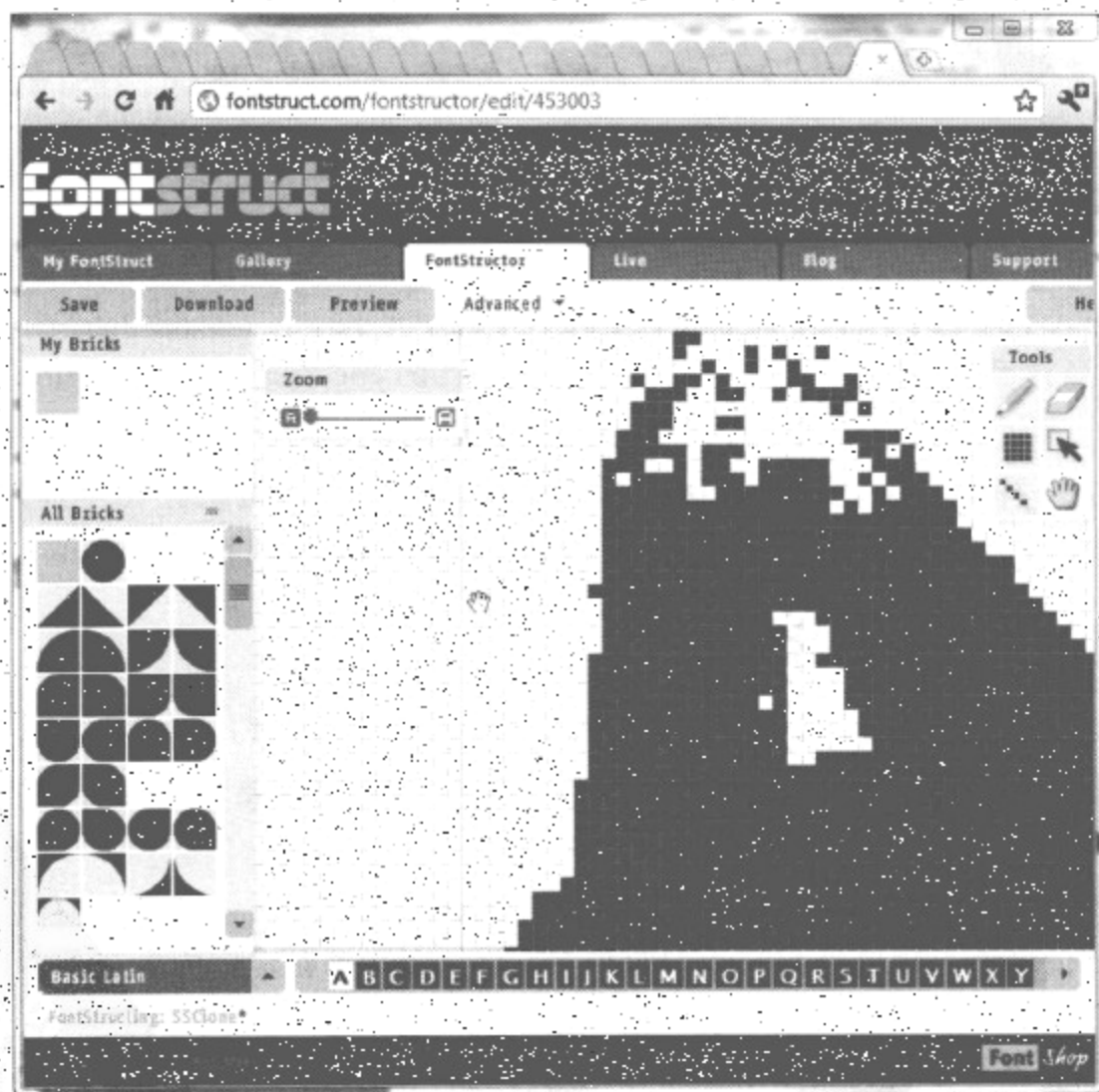


图 7.5 正在编辑 TrueType 字型 SwiftedStrokes 的 FontStruct 软件界面

## 7.5 将定制字型加入 V3

现在的 V3 代码已经将文本标签放置到各种图形之上, 包括背景图以及武器库中的武器图片。还有一个屏幕没做, 那就是选项画面, 它含有一些执行如下任务的菜单项:

- 打开与关闭音乐
- 打开与关闭音效

为了和整个游戏保持一致, 选项画面所显示的文字也要使用 Flubber 字型。菜单项的文本要随着开/关状态的变化而改变, 因为此时游戏中尚未有声音(第 11 章将加入声音), 所以此处的代码仅需要切换某个 Boolean 变量的值并修改菜单项文本即可。选项画面的效果如图 7.6 所示。

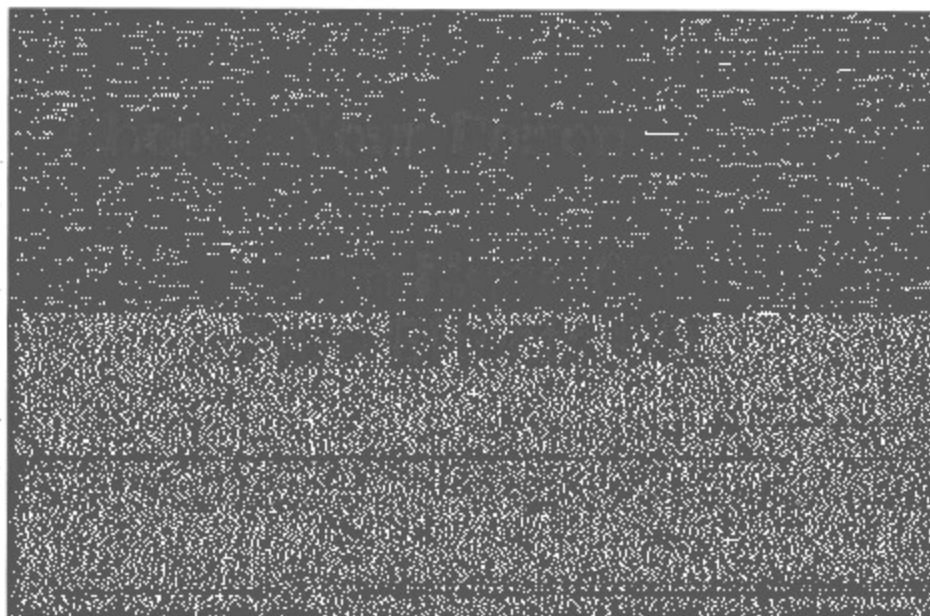


图 7.6 选项画面

要实现选项画面, 需要对已有游戏程序做修改。首先修改 MainMenuActivity.java, 使菜单项被选中时切换到 OptionsMenuActivity, 而不是像原来的代码那样弹出桌面通知。程序清单 7.2 是修改后的程序。

程序清单 7.2 增加 OptionsActivity 之后的 MainMenuActivity.java 文件

```
@Override
public boolean onOptionsItemSelected(final MenuScene pMenuScene,
    final IMenuItem pMenuItem, final float pMenuItemLocalX,
    final float pMenuItemLocalY) {
    switch(pMenuItem.getID()) {

        case MENU_OPTIONS:
            mMainScene.registerEntityModifier(
                new ScaleModifier(1.0f, 1.0f, 0.0f));
            mStaticMenuScene.registerEntityModifier(
```

```

        new ScaleModifier(1.0f, 1.0f, 0.0f));
        mHandler.postDelayed(mLaunchOptionsTask,
            1000);
        return true;
    }

    private Runnable mLaunchOptionsTask = new Runnable() {
        public void run() {
            Intent myIntent = new Intent(MainMenuActivity.this,
                OptionsActivity.class);
            MainMenuActivity.this.startActivity(myIntent);
        }
    }

```

同时需要如程序清单 7.3 所示，将 OptionsActivity 加入到 manifest 文件中。

程序清单 7.3 加入 OptionsActivity 之后的新版 AndroidManifest.xml 文件

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.pearson.lagp.v3"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".StartActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MainMenuActivity"></activity>
        <activity android:name=".LevellActivity"></activity>
        <activity android:name=".OptionsActivity"></activity>
    </application>
    <uses-sdk android:minSdkVersion="4" />
    <uses-permission android:name="android.permission.WAKE_LOCK">
    </uses-permission>
</manifest>

```

现在循例新建 BaseGameActivity 的子类 OptionsActivity 类，将其加入到游戏项目中。新类的代码如程序清单 7.4 所示。

程序清单 7.4 OptionsActivity.java

```

package com.pearson.lagp.v3;

```

```

...
// 多条 imports 指令
...

public class OptionsActivity extends BaseGameActivity implements
    IOnMenuItemClickListener {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    protected static final int MENU_MUSIC = 0;
    protected static final int MENU_EFFECTS = MENU_MUSIC + 1;

    // =====
    // 字段
    // =====

    protected Camera mCamera;

    protected Scene mMainScene;
    protected Handler mHandler;

    private Texture mMenuBackTexture;
    private TextureRegion mMenuBackTextureRegion;

    protected MenuScene mOptionsMenuScene;
    private TextMenuItem mTurnMusicOff, mTurnMusicOn;
    private TextMenuItem mTurnEffectsOff, mTurnEffectsOn;
    private IMenuItem musicMenuItem;
    private IMenuItem effectsMenuItem;

    private Texture mFontTexture;
    private Font mFont;

    public boolean isMusicOn = true;
    public boolean isEffectsOn = true;

    // =====
    // 构造器
    // =====

```

```

// =====
// Getter 和 Setter 方法
// =====

// =====
// 覆写超类及接口中的方法
// =====

@Override
public Engine onLoadEngine() {
    mHandler = new Handler();
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
}

@Override
public void onLoadResources() {
    /* 载入 Font 及 Textures 对象。 */
    this.mFontTexture = new Texture(256, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);

    FontFactory.setAssetBasePath("font/");
    this.mFont = FontFactory.createFromAsset(this.mFontTexture,
        this, "Flubber.ttf", 32, true, Color.WHITE);
    this.mEngine.getTextureManager().loadTexture(
        this.mFontTexture);
    this.mEngine.getFontManager().loadFont(this.mFont);

    this.mMenuBackTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mMenuBackTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mMenuBackTexture, this,
            "gfx/OptionsMenu/OptionsMenuBk.png", 0, 0);
    this.mEngine.getTextureManager().loadTexture(
        this.mMenuBackTexture);

    mTurnMusicOn = new TextMenuItem(MENU_MUSIC, mFont,
        "Turn Music On");
}

```

```

mTurnMusicOff = new TextMenuItem(MENU_MUSIC, mFont,
    "Turn Music Off");
mTurnEffectsOn = new TextMenuItem(MENU_EFFECTS, mFont,
    "Turn Effects On");
mTurnEffectsOff = new TextMenuItem(MENU_EFFECTS, mFont,
    "Turn Effects Off");
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    this.createOptionsMenuScene(true, true);

    /* 将摄像头置于背景中央。 */
    final int centerX = (CAMERA_WIDTH -
        this.mMenuBackTextureRegion.getWidth()) / 2;
    final int centerY = (CAMERA_HEIGHT -
        this.mMenuBackTextureRegion.getHeight()) / 2;

    this.mMainScene = new Scene(1);
    /* 增加背景与静态菜单。 */
    final Sprite menuBack = new Sprite(centerX, centerY,
        this.mMenuBackTextureRegion);
    mMainScene.getLastChild().attachChild(menuBack);
    mMainScene.setChildScene(mOptionsMenuScene);

    return this.mMainScene;
}

@Override
public void onLoadComplete() {
}

@Override
public boolean onMenuItemClicked(final MenuScene pMenuScene,
    final IMenuItem pMenuItem, final float pMenuItemLocalX,
    final float pMenuItemLocalY) {
    switch(pMenuItem.getID()) {
        case MENU_MUSIC:
            if (isMusicOn) {
                isMusicOn = false;
            } else {
                isMusicOn = true;
            }
    }
}

```



```

        createOptionsMenuScene();
        mMainScene.clearChildScene();
        mMainScene.setChildScene(mOptionsMenuScene);
        return true;
    case MENU_EFFECTS:
        if (isEffectsOn) {
            false);
            isEffectsOn = false;
        } else {
            true);
            isEffectsOn = true;
        }
        createOptionsMenuScene();
        mMainScene.clearChildScene();
        mMainScene.setChildScene(mOptionsMenuScene);
        return true;
    default:
        return false;
}

// =====
// 方法
// =====

protected void createOptionsMenuScene() {
    this.mOptionsMenuScene = new MenuScene(this.mCamera);

    if (isMusicOn) {
        musicMenuItem = new ColorMenuItemDecorator(
            mTurnMusicOff, 0.5f, 0.5f, 0.5f, 1.0f,
            0.0f, 0.0f);
    } else {
        musicMenuItem = new ColorMenuItemDecorator(
            mTurnMusicOn, 0.5f, 0.5f, 0.5f, 1.0f,
            0.0f, 0.0f);
    }
    musicMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mOptionsMenuScene.addMenuItem(musicMenuItem);

    if (isEffectsOn) {
        effectsMenuItem = new ColorMenuItemDecorator(
            mTurnEffectsOff, 0.5f, 0.5f, 0.5f,

```

```

        1.0f, 0.0f, 0.0f);
    } else {
        effectsMenuItem = new ColorMenuItemDecorator(
            mTurnEffectsOn, 0.5f, 0.5f, 0.5f,
            1.0f, 0.0f, 0.0f);
    }
    effectsMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mOptionsMenuScene.addMenuItem(effectsMenuItem);
    this.mOptionsMenuScene.buildAnimations();
    this.mOptionsMenuScene.setBackgroundEnabled(false);
    this.mOptionsMenuScene.setOnMenuItemClickListener(this);
}
}
. . .

```

这段代码与第3章的 MainMenuActivity.java 很像，通过比较它们的异同可以了解此段代码的工作原理：

- ❑ onLoadEngine() 方法依然不变。
- ❑ onLoadResources() 方法的代码载入了字体资源文件 Flubber.ttf。
  - ❑ Texture 对象是 256 像素 × 256 像素大小，在前面的文本显示范例中，已经通过尝试证明它足以容纳 32 像素大小的字符了。
  - ❑ 字型文件查找路径设置为 “/font”，因为 assets 目录下的不同类型文件是分类存放的，同时 Flubber.ttf 已经被导入该目录。
  - ❑ 创建 Font 对象，并分别通过 TextureManager 与 FontManager 将 Texture 对象与 Font 对象载入内存。
  - ❑ Scene 对象所使用的 Texture 对象依然用原来的方法载入。
  - ❑ 定义了选项屏幕所使用的 4 个 TextMenuItem 对象，使用其中的 2 个分别显示 isMusicOn() 和 isEffectsOn() 方法的 Boolean 型返回值。很抱歉，这两个方法不符合“动词+宾语”的命名规范，但是返回 Boolean 值的方法确实需要以 “is” 开头。
- ❑ onLoadScene() 方法调用 createOptionsMenuScene() 方法创建菜单，该方法也用来在 TextMenuItem 对象的文本需要改变时重建菜单。前面提到过，TextMenuItem 对象一旦创建，则不可改变其文本，所以要想改变，必须重建。
- ❑ onMenuItemClicked() 方法的代码捕捉菜单项中的触摸事件，并切换 boolean 字段的值，然后调用 createOptionsMenuScene() 方法重建菜单，最后将旧的菜单对象删除并将新的菜单对象加入到 Scene 对象中。
- ❑ createOptionsMenuScene() 方法使用 Boolean 值来设定菜单项的文本，同时通过

应用 `ColorMenuItemDecorator` 对象使得菜单项在未按下时显示为红色，在被按下时变为灰色。

运行游戏并在主菜单画面选择 `Options` 菜单项，将会切换到选项菜单，可以在此打开或关闭游戏的音乐与音效。尽管现在还没有声音，但是第 11 章加入声音效果时将会用到这里设置的 `Boolean` 值。

## 7.6 总结

本章讲述了 `AndEngine` 中有关文本使用的重要内容：可以在游戏中使用选定的样式制作含有文本的标签，使用定制的矢量字体，以及发布桌面通知。

现将用 `AndEngine` 创建 `OpenGL` 渲染对象的步骤总结如下：

- 创建能够容纳可显示实体所需图像的 `Texture` 对象。
  - 将实体所用的图像文件载入 `Texture` 对象中。图像文件可以使用 `Android` 对象（例如 `Typeface`）、`asset` 资源（例如定制的 `TrueType` 字型文件）以及 `Resource` 资源（如第 5 章讲 `Sprite` 的创建时所述）。
  - 使用 `TextureManager` 单例将 `Texture` 对象载入缓存。
  - 有时也需要使用某种实体管理器（例如 `FontManager`）来将 `Entity` 对象载入缓存。
  - 创建可显示的对象（例如 `Sprite` 对象、`Text` 对象）并将其加至 `Scene` 对象。
- 此步骤已经重复了很多次，在接下来的游戏制作中仍要大量用到它。

## 7.7 习题

1. 修改 `TextExample.java` 中的 `Boolean` 值，使得屏幕显示如图 7.7 所示效果。



图 7.7 带有填充色的 `StrokeFont` 对象

2. 在原有的 `TextExample.java` 文件上进行如下修改：
  - 将 `mStrokeFont` 的大小修改为 64 像素。

- ❑ 将 textStroke 的位置设置为 (60,160)。
- ❑ 将两条文本所使用的字体修改为 Typeface.SERIF。
- ❑ 运行程序后, 应该显示如图 7.8 所示效果。其中的“i”字符为何没有显示出来?
- ❑ 修改程序, 使“i”字符能够正确地显示。

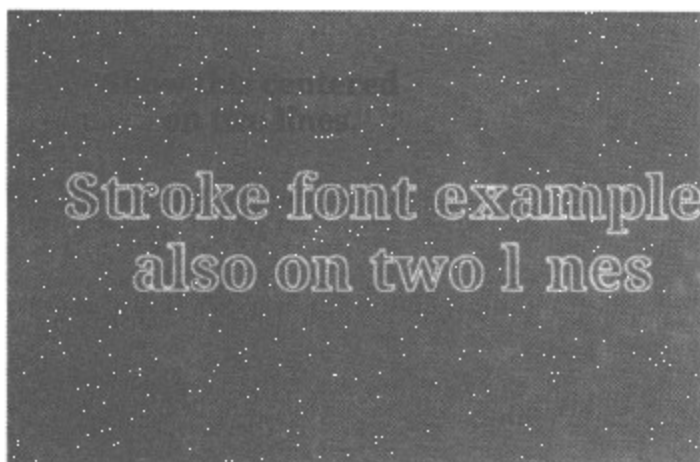


图 7.8 修改之后的 TextExample

3. 在选项菜单上增加“Help”菜单项, 以便在此直接切换到帮助屏幕 (这里仅仅显示桌面通知即可), 而不需先切换到主菜单。

4. 修改代码, 在 OptionsMenuActivity 中代表音乐及音效开关的 Boolean 值改变时, 弹出桌面通知。



## 第 8 章 用 户 输 入

- 8.1 Android 与 AndEngine 的输入方式
- 8.2 将用户输入加入 V3
- 8.3 总结
- 8.4 习题

如果电脑游戏没有用户输入，将会变得毫无意义。Android 系统搭配上 AndEngine 之后，就构成了一套功能丰富的输入处理系统，开发者可利用它使游戏更有趣。本章先讲解每个用户输入的功能，然后讲解 AndEngine 如何利用底层的 Android 系统功能来处理输入。有些功能 AndEngine 尚未支持，用户可能需要直接使用 Android 系统所提供的对应功能，如 EditText 等 View 对象，这些内容本章也会讲到。

## 8.1 Android 与 AndEngine 的输入方式

Android 系统拥有非常灵活的用户界面，能够应对多种风格的用户输入。这款游戏将会在很多不同的手机上运行，所以要尽可能多地支持各种输入方式。迄今为止，Android 手机支持如下的输入方式（以后还会更多）：

- ☐ 硬 QWERTY 键盘<sup>①</sup>（多种语言布局）
- ☐ 软 QWERTY 键盘（多种语言布局）
- ☐ 方向键盘（上 / 下 / 左 / 右 / 选择）
- ☐ 袖珍（小）键盘（具备专用按键及触摸点）
  - ☐ 发送键（Send）
  - ☐ 主桌面键（Home）
  - ☐ 返回键（Back）
  - ☐ 结束键（End）
  - ☐ 音量大 / 小键（Vol+/-）
  - ☐ 搜索键（Search）
  - ☐ 菜单键（Menu）
- ☐ 轨迹球
- ☐ 单点触控屏幕
- ☐ 单点触控键盘
- ☐ 多点触摸屏幕
- ☐ 语音识别
- ☐ 加速度计
- ☐ 定位系统（GPS 或 AGPS）
- ☐ 方向识别（罗盘航向）

读者可能并未意识到后三种也算用户输入，不过游戏中确实会用它们获取输入数据。

<sup>①</sup> QWERTY 是各国使用的基于拉丁字母的标准的打字机和计算机键盘。QWERTY 是键盘第一行的前六个字母。——译者注



### 8.1.1 字母键盘与袖珍键盘

Android 系统将不同类型与布局的键盘所产生的按键事件统一起来，事件数据中除了包含诸如按下、弹起、持续按键等事件类型外，还包括指示相关按键的键值码。使用 AndEngine 开发游戏时也可以访问这些事件。

现有的键值码超过了 200 个，包括在字母键盘上可能会出现的多媒体控制键、照相机键及 TV 键等。在 Android 开发者站点可以找到下列描述键值码的文档：

<http://developer.android.com/reference/android/view/KeyEvent.html>

包括 AndEngine 游戏在内的所有 Android 系统 Activity 对象，均可以将自己注册为按键事件监听器，这样当有按键事件发生时，就会收到回调通知。可以通过从事件中获取的旗标来判断此事件的产生是通过硬按键产生，还是软按键，或者是以编程的方式（并未实际按下任何键）。

在涉及 Android 的标准 View 时，系统会自行处理相关细节。例如某应用程序创建了一个可以编辑的 TextView，那么当用户正在其中编辑字符串时，Android 系统会负责收集这些按键，并在用户完成编辑后显示编辑好的字符串，期间的按键数据均不会以按键事件的形式传递给监听器。

程序清单 8.1 是一段典型的捕捉并处理按键事件的代码片段。

程序清单 8.1 捕捉按键事件

---

```
@Override
public boolean onKeyDown(final int pKeyCode,
    final KeyEvent pEvent) {
    if(pKeyCode == KeyEvent.<insert keycode> &&
        pEvent.getAction() == KeyEvent.ACTION_DOWN) {
        // 执行按键按下后的逻辑。
        return true;
    } else {
        return super.onKeyDown(pKeyCode, pEvent);
    }
}
```

---

onKeyDown() 方法覆写了 Activity 类的默认实现，当某个按键被按下，且没有被任何 View 对象处理的时候，该方法就会被调用。如果返回 true，表明该方法已经处理了这个按键按下事件，它不应该再被传播到按键处理链中的其他处理程序中去了。AndEngine 与其他 Android 应用一样，也沿用这套 API。

### 8.1.2 触摸

Android 系统一开始仅仅支持单点触控接口，目前游戏中使用的这套就是。然而，很快 Android 就开始支持多点触摸了：一开始是为了缩放手势，然后提供了一套更加通

用的接口。AndEngine 将 Android 系统的基本触摸功能与管理触摸事件的代码包裹起来，使游戏中处理碰撞的代码更容易编写。

### 单点触控模式

在单点触控模式下，当用户按下屏幕上的某个点，监听器就会收到 MotionEvent 事件。可以通过标准的 Android API 向 View 注册 onTouchListener() 方法。当 View 对象被触摸时，相应的 MotionEvent 事件（按下、拖动、弹起）会被推送至回调方法。程序清单 8.2 示范了标准的 Android 应用程序如何捕捉及处理触摸事件。

程序清单 8.2 Android 触摸事件的捕捉与处理

---

```
package com.pearson.lagp.example;
...
public class TouchExample extends Activity implements OnTouchListener {
    private LinearLayout linear;
    private ImageView image;
    ...
    @Override
    public boolean onTouch(View v, MotionEvent e) {
        if (e.getAction() == MotionEvent.ACTION_DOWN) {
            Toast.makeText(this, "Down",
                Toast.LENGTH_SHORT).show();
        }
        return false;
    }
}
```

---

该方法对于标准的 Android 系统 View 对象来说，很合适，然而 AndEngine 所开发的游戏只含有一个 View，所有的 OpenGL 相关绘制都是由作为桌布的 GLSurfaceView 对象来执行的。如果想用此方法处理触摸事件，那么需要将屏幕上的每一个坐标点都映射到对应的 Entity 对象上去。

好在 AndEngine 已经为我们做好了这些运算，并扩充了 Android 处理触摸事件的范式。只要是从 Shape 继承下来的类对象，包括 Sprite，都可以在其上注册触摸事件监听器。开发者也可以为任何 Scene 对象注册触摸监听器。相关方法如下所示：

```
void Scene.setOnSceneTouchListener(final IOnSceneTouchListener
    pOnSceneTouchListener)
boolean Shape.onAreaTouched(final TouchEvent pSceneTouchEvent,
    final float pTouchAreaLocalX, final float pTouchAreaLocalY)
```

参数的意义一看即知。onAreaTouched() 方法中的 TouchEvent 类是 AndEngine 自创的，它和 Android 的 MotionEvent 类很像，不过为了尽力避免垃圾回收，AndEngine 用了缓冲池技术来管理 TouchEvent 对象。

程序清单 8.3 示范了如何捕捉 Scene 与 Sprite 对象的触摸事件。

程序清单 8.3 AndEngine 触摸事件的捕捉与处理

```
public class AndEngineTouchExample extends BaseGameActivity {
    . . .

    // =====
    // 覆写超类及接口中的方法
    // =====
    . . .

    @Override
    public void onLoadResources() {
        mIconTexture = new BuildableTexture(128, 128,
            TextureOptions.BILINEAR_PREMULTIPLYALPHA);
        mIconTextureRegion =
            TextureRegionFactory.createFromAsset(
                this.mIconTexture, this, "icon.png");
        try {
            mIconTexture.build(
                new BlackPawnTextureBuilder(2));
        } catch (final TextureSourcePackingException e) {
            Log.d(tag, "Sprites won't fit in mIconTexture");
        }
        this.mEngine.getTextureManager().loadTexture(
            this.mIconTexture);
    }

    @Override
    public Scene onLoadScene() {
        final Scene scene = new Scene(1);
        scene.setBackground(new ColorBackground(0.1f, 0.6f, 0.9f));
        scene.setOnSceneTouchListener(new IOnSceneTouchListener() {
            @Override
            public boolean onSceneTouchEvent(
                final Scene pScene,
                final TouchEvent pSceneTouchEvent) {
                switch(pSceneTouchEvent.getAction()) {
                    case TouchEvent.ACTION_DOWN:
                        Toast.makeText(
                            AndEngineTouchExample.this,
                            "Scene touch DOWN",
                            Toast.LENGTH_SHORT).show();
                        break;
                    case TouchEvent.ACTION_UP:
                        Toast.makeText(
                            AndEngineTouchExample.this,
                            "Scene touch UP",
```

```

        Toast.LENGTH_SHORT).show();
        break;
    }
    return true;
}
});

mIcon = new Sprite(100, 100, this.mIconTextureRegion) {
    @Override
    public boolean onAreaTouched(
        final TouchEvent pAreaTouchEvent,
        final float pTouchAreaLocalX,
        final float pTouchAreaLocalY) {
        switch(pAreaTouchEvent.getAction()) {
            case TouchEvent.ACTION_DOWN:
                Toast.makeText(
                    AndEngineTouchExample.this,
                    "Sprite touch DOWN",
                    Toast.LENGTH_SHORT).show();
                break;
            case TouchEvent.ACTION_UP:
                Toast.makeText(
                    AndEngineTouchExample.this,
                    "Sprite touch UP",
                    Toast.LENGTH_SHORT).show();
                break;
        }
        return true;
    }
};

scene.getLastChild().attachChild(mIcon);
scene.registerTouchArea(mIcon);
scene.setTouchAreaBindingEnabled(true);

return scene;
}

@Override
public void onLoadComplete() {
}

}

```

在 Android 手机上运行该程序后，屏幕会显示一幅浅蓝色背景与一个 Android 图标。如果触摸图标，则会弹出一条桌面通知，指示精灵对象上发生了触摸事件；如果在别处触摸，则会通知屏幕上面发生了触摸事件。

注意代码最后的 `setTouchAreaBindingEnabled()` 方法，它可以使 AndEngine 将其他区域的触摸事件（例如拖动、松开）绑定到同样的 Sprite 对象上。该方法有时很有用，

比如玩家要通过拖动来重新定位某 Sprite 对象,有时由于延迟的关系,会导致相关触摸事件绑定到此 Sprite 之外的对象上。通过设置该方法,可以让 AndEngine 将下一个触摸按下事件之前发生的所有触摸事件都绑定到此 Sprite 对象上。为了演示该方法的效果,请按住图标(这时会产生一个“Sprite touch DOWN”的桌面通知),拖动,然后松开,尽管此时触摸松开事件已经发生在此 Sprite 对象之外,但是产生的桌面通知包含的信息仍是“Sprite touch UP”,而非“Scene touch UP”,这正是开发者所期望的效果。

### 多点触摸模式

处理多点触摸时,第一个触摸动作仍会产生普通的触摸事件,其后其他点的触摸动作将会产生 MotionEvent.ACTION\_POINTER 类型的触摸事件。AndEngine 处理多点触摸时仍然沿用 Android 系统的多点触摸 API。如果需要在游戏中使用这些 API,可以在 Android 网站上查阅详细文档。

### 手势

除了简单的触摸事件,有时需要根据触摸动作序列生成手势事件。单一的触摸手势自从 API Level<sup>①</sup> 4 版本(即 Android 1.6)就开始支持了。有限度的多点触摸手势是从 API Level 8(即 Android 2.2)版本开始支持的,并且仅限于两个手指所做的缩放手势。

在本书写作时,AndEngineExample 中尚未包含关于手势的例子。程序清单 8.4 演示了使用基础 Android API 来捕获每一种支持的手势,并弹出桌面通知。

程序清单 8.4 使用 Android API 来捕捉与处理手势事件的范例代码

```
public class GestureExample extends BaseGameActivity {
    @Override
    public Engine onLoadEngine() {
        mGestureDetector = new GestureDetector(this,
            new ExampleGestureListener());
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        return new Engine(new EngineOptions(true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
    }
}
```

① API Level 是 Android 开发包版本的标识方式,每种 Level 都对应于某个 Android 操作系统的版本,详细列表可查阅 <http://developer.android.com/guide/appendix/api-levels.html>。——译者注

```

@Override
public boolean onTouchEvent(MotionEvent event) {
    if (mGestureDetector.onTouchEvent(event))
        return true;
    else
        return false;
}

class ExampleGestureListener extends
    GestureDetector.SimpleOnGestureListener{
    @Override
    public boolean onSingleTapUp(MotionEvent ev) {
        Toast.makeText(GestureExample.this,
            "Single tap up.", Toast.LENGTH_SHORT).show();
        return true;
    }

    @Override
    public void onShowPress(MotionEvent ev) {
        Toast.makeText(GestureExample.this,
            "Show press.", Toast.LENGTH_SHORT).show();
        return true;
    }

    @Override
    public void onLongPress(MotionEvent ev) {
        Toast.makeText(GestureExample.this,
            "Long press.", Toast.LENGTH_SHORT).show();
        return true;
    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2,
        float distanceX, float distanceY) {
        Toast.makeText(GestureExample.this, "Scroll.",
            Toast.LENGTH_SHORT).show();
        return true;
    }

    @Override
    public boolean onDown(MotionEvent ev) {
        Toast.makeText(GestureExample.this, "Down.",
            Toast.LENGTH_SHORT).show();
        return true;
    }

    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2,
        float velocityX, float velocityY) {
        Toast.makeText(GestureExample.this,

```



```
        "Fling.", Toast.LENGTH_SHORT).show();  
        return true;  
    }  
}
```

运行此程序（最好在手机而非模拟器上），就会看到当做出手势时，会弹出相应的桌面通知。Android 的手势处理接口非常灵活，在本书写作时，AndEngine 的开发者正持续地致力于研究如何简化游戏中的手势处理。读者可以到 AndEngine 论坛上查阅相关信息：

<http://www.andengine.org/forums>

### 8.1.3 自定义手势

Android SDK 包含了名为手势构建器（Gesture Builder）的应用程序，可以用它来创建并记录单点触摸的手势。尽管也可以在模拟器里运行，但是创建手势最简单的办法还是在 Android 手机上运行，这样产生的手势文件将会以 gestures 为文件名，存放在手机的 SD 卡上。将手机通过 USB 线连上电脑，就可以通过如下 adb 命令获取该文件了：

```
adb -d pull /sdcard/gestures
```

执行上述命令后，手势文件将会存入当前工作目录，然后需要将其导入游戏项目的 assets 文件夹下。尽管 V3 游戏中并不使用它，不过开发者可以通过使用自定义手势而使自己制作的游戏更加有趣。如果需要了解自定义手势的用法，请参阅以下网址：

<http://developer.android.com/resources/articles/gestures.html>

### 8.1.4 屏幕游戏手柄

AndEngine 有个非常实用的功能，就是可以在游戏屏幕上显示模拟的游戏控制器，它们通常以一定的透明度浮现于游戏画面之上，并且可以如真实的游戏手柄那般响应触摸事件。有两种方式的屏幕游戏手柄可选：

- ❑ 类比手柄（Analog Controller），其按钮偏离中心点的距离值将用来按比例计算游戏中的距离值，以便控制游戏操作。
- ❑ 方向键手柄（Digital Controller），它更像方向键盘，按钮只有上、下、左、右键。

图 8.1 是类比手柄效果图，该范例程序的代码在 AndEngineExample 项目的 AnalogOnScreenControlsExample.java 文件中。

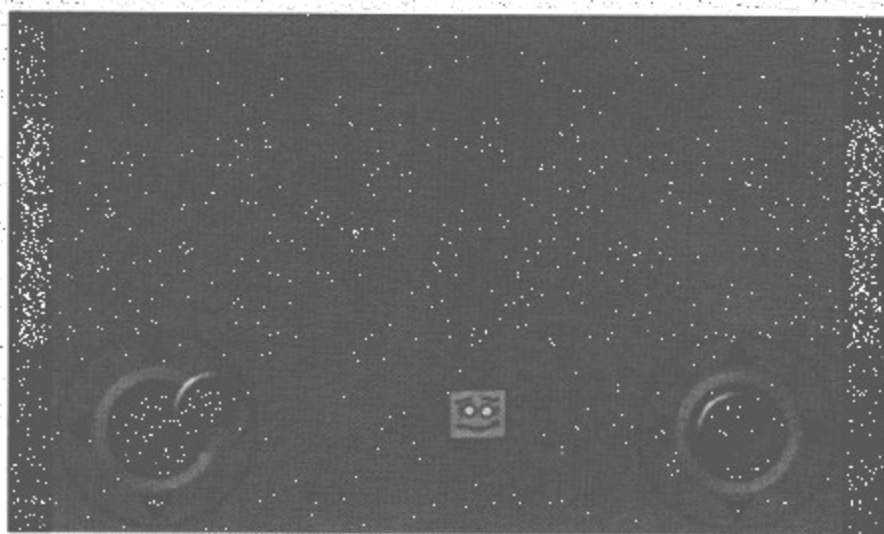


图 8.1 AndEngineExample 所演示的屏幕手柄效果

该手柄看起来很真实，不过尽管它们是透明的，但是仍然会占据一定的屏幕空间。V3 游戏中不会使用屏幕手柄，如果开发者对其感兴趣，想在自己的游戏中使用的话，可以看看 AndEngineExample 项目中的相关范例或参考附录中对习题 1 的解答。

### 8.1.5 加速计

大部分 Android 手机都有加速计，用 AndEngine 开发的游戏也可以使用它。作为用户输入方式，它主要用来侦测手机的倾斜程度。通常的用法是在屏幕上显示某个可滚动的物体，玩家通过倾斜手机而让其滚动至目的地，同时避开障碍物。

加速计功能通常大都与物理学模块一起使用，本书第 12 章将详细讨论这个话题。现在读者可以在习题 3 与附录中对该题的解答中简单地了解一下如何使用加速计。

### 8.1.6 位置和方向

大多数 Android 手机都包含定位感应器（通常是 GPS 或 AGPS）以及方向感应器（用于计算罗盘方向的磁力计），它们在应用程序中通常都不会用作用户输入，不过在游戏中却有可能。AndEngine 包裹了基本的 Android API，并增加了一些类，使位置与方向的处理更加简单。

要使用方向相关的 API，需要首先让 Activity 类实现 `IOrientationListener` 接口，并覆写如下方法：

```
void onOrientationChanged(final OrientationData pOrientationData)
```

回调方法中的 `OrientationData` 包含了很多数据，包含最常用的 roll、pitch 及 yaw 信息<sup>①</sup>，也就是三个坐标轴方向上的转动角度，其值从 0.0 到 360.0，而且有对应的 getter

① 此三数值用于表示矩阵旋转中 x、y、z 三个坐标轴方向的旋转角度，中文叫做滚动、俯仰、偏航。——译者注

方法用于查询。

AndEngine 所定义的 ILocationListener 包含更多需要覆写的方法:

```
void onLocationChanged(final Location pLocation)
void onLocationLost()
void onLocationProviderDisabled()
onLocationProviderStatusChanged(final LocationProviderStatus
    pLocationProviderStatus, final Bundle pBundle)
```

这些方法中, onLocationChanged() 最常用, 通过它可以找出手机现在的新位置。与其他 Android 应用程序一样, 要使用定位 API, 需要请求相关的权限, 并通过创建新的 Location 对象 (参见程序清单 8.5 之后的注释) 来访问某个定位信息提供者。程序清单 8.5 展示了如何使用 AndEngine 所提供的定位与方向功能。

程序清单 8.5 使用定位与方向功能的范例代码

---

```
package com.pearson.lagp.example;

import org.anddev.andengine.engine.Engine;
import org.anddev.andengine.engine.camera.Camera;
import org.anddev.andengine.engine.options.EngineOptions;
import org.anddev.andengine.engine.options.EngineOptions
    .ScreenOrientation;
import org.anddev.andengine.engine.options.resolutionpolicy
    .RatioResolutionPolicy;
import org.anddev.andengine.entity.scene.Scene;
import org.anddev.andengine.entity.scene.background.ColorBackground;
import org.anddev.andengine.entity.sprite.Sprite;
import org.anddev.andengine.opengl.texture.BuildableTexture;
import org.anddev.andengine.opengl.texture.TextureOptions;
import org.anddev.andengine.opengl.texture.builder
    .BlackPawnTextureBuilder;
import org.anddev.andengine.opengl.texture.builder.ITextureBuilder
    .TextureSourcePackingException;
import org.anddev.andengine.opengl.texture.region.TextureRegion;
import org.anddev.andengine.opengl.texture.region.TextureRegionFactory;
import org.anddev.andengine.sensor.location.ILocationListener;
import org.anddev.andengine.sensor.location.LocationProviderStatus;
import org.anddev.andengine.sensor.location.LocationSensorOptions;
import org.anddev.andengine.sensor.orientation.IOrientationListener;
import org.anddev.andengine.sensor.orientation.OrientationData;
import org.anddev.andengine.ui.activity.BaseGameActivity;
import org.anddev.andengine.util.Debug;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;
```

```

public class AndEngineSensorExample extends BaseGameActivity implements
IOrientationListener, ILocationListener {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;
    private String tag = "AndEngineSensorExample";

    // =====
    // 字段
    // =====

    protected Camera mCamera;
    private Location mUserLocation;

    protected Scene mMainScene;
    protected Sprite mIcon;

    private BuildableTexture mIconTexture;
    private TextureRegion mIconTextureRegion;
    private Location mLocation;

    // =====
    // 构造器
    // =====
    // Getter 和 Setter 方法
    // =====

    // =====
    // 覆写超类及接口中的方法
    // =====

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.mLocation = new Location(
            LocationManager.GPS_PROVIDER);
    }

    @Override
    public Engine onLoadEngine() {
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);

        return new Engine(new EngineOptions(true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,

```

```

        CAMERA_HEIGHT), this.mCamera));
    }

    @Override
    public void onLoadResources() {
        mIconTexture = new BuildableTexture(128, 128,
            TextureOptions.BILINEAR_PREMULTIPLYALPHA);
        mIconTextureRegion =
            TextureRegionFactory.createFromAsset(
                this.mIconTexture, this, "icon.png");
        try {
            mIconTexture.build(new BlackPawnTextureBuilder(2));
        } catch (final TextureSourcePackingException e) {
            Log.d(tag, "Sprites won't fit in mIconTexture");
        }
        this.mEngine.getTextureManager().loadTexture(
            this.mIconTexture);
    }

    @Override
    public Scene onLoadScene() {
        final Scene scene = new Scene(1);
        scene.setBackground(new ColorBackground(0.1f, 0.6f, 0.9f));
        mIcon = new Sprite(100, 100, this.mIconTextureRegion);
        scene.getLastChild().attachChild(mIcon);
        return scene;
    }

    @Override
    public void onLoadComplete() {
    }

    @Override
    protected void onResume() {
        super.onResume();

        this.enableOrientationSensor(AndEngineSensorExample.this);

        final LocationSensorOptions locationSensorOptions =
            new LocationSensorOptions();
        locationSensorOptions.setAccuracy(
            Criteria.ACCURACY_COARSE);
        locationSensorOptions.setMinimumTriggerTime(0);
        locationSensorOptions.setMinimumTriggerDistance(0);
        this.enableLocationSensor(AndEngineSensorExample.this,
            locationSensorOptions);
    }

    @Override

```

```

protected void onPause() {
    super.onPause();
    this.mEngine.disableOrientationSensor(this);
    this.mEngine.disableLocationSensor(this);
}

@Override
public void onOrientationChanged(
    final OrientationData pOrientationData) {
    float yaw = pOrientationData.getYaw() / 360.0f;
    mIcon.setPosition( CAMERA_WIDTH/2, yaw * CAMERA_HEIGHT);
}

@Override
public void onLocationChanged(final Location pLocation) {
    String tst = "Lat: " + pLocation.getLatitude()+
        " Lng: " + pLocation.getLongitude();
    Toast.makeText(AndEngineSensorExample.this, tst,
        Toast.LENGTH_LONG).show();
}

@Override
public void onLocationLost() {
}

@Override
public void onLocationProviderDisabled() {
}

@Override
public void onLocationProviderEnabled() {
}

@Override
public void onLocationProviderStatusChanged(
    final LocationProviderStatus pLocationProviderStatus,
    final Bundle pBundle) {
}
}

```

该段代码的要点如下：

- ❑ 要访问位置信息，应用程序必须具有适当的权限，对于这段范例代码（请求 GPS 位置提供器），其 manifest 包含了如下声明：

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION">
```

- ❑ 需要覆写 onCreate() 方法，而不是使用 BaseGameActivity 中定义的版本，因为需要在 onResume() 方法被调用之前先激活位置提供器。
- ❑ 也需要覆写 onPause() 与 onResume() 方法，而不能使用 BaseGameActivity 中定



义的版本，因为需要在其中正确地关闭与打开对方向及位置的监听，Android 系统可以在应用程序不使用这些数据时关闭相应的感应器以减少耗电。

□ `ILocationListener` 接口所定义的方法很多，然而本类在许多方法的实现中都没有写任何代码，因为不需要使用这些回调方法。

### 8.1.7 语音

Android 系统提供了通用的语音识别 API，并利用运行在 Google 服务器上的程序来返回结果。因为原始的语音信息必须往返于本地程序与远程服务器之间，所以如果要在游戏中使用语音，应该将延迟因素考虑在内。另一个需要注意的地方是，默认的语音识别用户界面通常会出现在游戏屏幕上，干扰游戏的进行。程序清单 8.6 中简单地演示了如何使用语音识别功能。

程序清单 8.6 Android 的语音识别功能演示

---

```
package com.pearson.lagp.example;

import java.util.ArrayList;

import org.anddev.andengine.engine.Engine;
import org.anddev.andengine.engine.camera.Camera;
import org.anddev.andengine.engine.options.EngineOptions;
import org.anddev.andengine.engine.options.EngineOptions
    .ScreenOrientation;
import org.anddev.andengine.engine.options.resolutionpolicy
    .RatioResolutionPolicy;
import org.anddev.andengine.entity.scene.Scene;
import org.anddev.andengine.entity.scene.background.ColorBackground;
import org.anddev.andengine.entity.sprite.Sprite;
import org.anddev.andengine.input.touch.TouchEvent;
import org.anddev.andengine.opengl.texture.BuildableTexture;
import org.anddev.andengine.opengl.texture.TextureOptions;
import org.anddev.andengine.opengl.texture.builder
    .BlackPawnTextureBuilder;
import org.anddev.andengine.opengl.texture.builder.ITextureBuilder
    .TextureSourcePackingException;
import org.anddev.andengine.opengl.texture.region.TextureRegion;
import org.anddev.andengine.opengl.texture.region.TextureRegionFactory;
import org.anddev.andengine.ui.activity.BaseGameActivity;

import android.content.Intent;
import android.speech.RecognizerIntent;
import android.util.Log;
import android.widget.Toast;

public class VoiceRecExample extends BaseGameActivity {
    // =====
```

```

// 常量
// =====

private static final int CAMERA_WIDTH = 480;
private static final int CAMERA_HEIGHT = 320;
private String tag = "VoiceRecExample";
    private static final int VOICE_RECOGNITION_REQUEST_CODE = 1234;

// =====
// 字段
// =====

protected Camera mCamera;

protected Scene mMainScene;
protected Sprite mIcon;

private BuildableTexture mIconTexture;
private TextureRegion mIconTextureRegion;

// =====
// 构造器
// =====

// =====
// Getter 和 Setter 方法
// =====
// =====
// 覆写超类及接口中的方法
// =====

@Override
public Engine onLoadEngine() {
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
}

@Override
public void onLoadResources() {
    mIconTexture = new BuildableTexture(128, 128,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mIconTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mIconTexture, this, "icon.png");
    try {
        mIconTexture.build(

```

```

        new BlackPawnTextureBuilder(2));
    } catch (final TextureSourcePackingException e) {
        Log.d(tag, "Sprites won't fit in mIconTexture");
    }
    this.mEngine.getTextureManager().loadTexture(this.mIconTexture);
}

@Override
public Scene onLoadScene() {
    final Scene scene = new Scene(1);
    scene.setBackground(new ColorBackground(0.1f, 0.6f, 0.9f));

    mIcon = new Sprite(CAMERA_WIDTH/2, CAMERA_HEIGHT/2,
        this.mIconTextureRegion) {
        @Override
        public boolean onAreaTouched(
            final TouchEvent pAreaTouchEvent,
            final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            switch(pAreaTouchEvent.getAction()) {
                case TouchEvent.ACTION_DOWN:
                    startVoiceRecognitionActivity();
                    break;
                case TouchEvent.ACTION_UP:
                    break;
            }
            return true;
        }
    };

    scene.getLastChild().attachChild(mIcon);
    scene.registerTouchArea(mIcon);
    scene.setTouchAreaBindingEnabled(true);
    Toast.makeText(VoiceRecExample.this,
        "Touch icon and say move left/right/up/down",
        Toast.LENGTH_SHORT).show();
    return scene;
}

@Override
public void onLoadComplete() {
}

// =====
// 方法
// =====

private void startVoiceRecognitionActivity() {
    Intent intent = new Intent(RecognizerIntent
        .ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
        RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT,

```

```

        "Speech recognition demo");
        startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
        if (requestCode == VOICE_RECOGNITION_REQUEST_CODE &&
            resultCode == RESULT_OK) {
            ArrayList<String> matches = data.getStringArrayListExtra(
                RecognizerIntent.EXTRA_RESULTS);
            for (String match : matches) {
                if (match.equalsIgnoreCase("move left")) {
                    mIcon.setPosition(mIcon.getX() - 10.0f,
                        mIcon.getY());
                }
                if (match.equalsIgnoreCase("move right")) {
                    mIcon.setPosition(mIcon.getX() + 10.0f,
                        mIcon.getY());
                }
                if (match.equalsIgnoreCase("move up")) {
                    mIcon.setPosition(mIcon.getX(),
                        mIcon.getY() - 10.0f);
                }
                if (match.equalsIgnoreCase("move down")) {
                    mIcon.setPosition(mIcon.getX(),
                        mIcon.getY() + 10.0f);
                }
            }
        }

        super.onActivityResult(requestCode, resultCode, data);
    }

    // =====
    // 内部类与匿名类
    // =====
}

```

将 Voice Recognizer 程序安装至支持语音搜索的 Android 手机上，运行后触摸图标，就会弹出语音识别界面。用户说出“Move left”、“Move up”等指令后，图标就会根据语音指令移动，然后语音识别界面就消失了。

## 8.2 将用户输入加入 V3

现在可以给正在开发的范例游戏加入用户输入功能了。回想一下第 1 关的场景：有一群正在走向 Miss B 房间的吸血鬼，屏幕左下角还有武器库（如图 8.2 所

示)。现在需要使玩家能够触摸代表武器的 Sprite 对象, 将其拖放至墓地上, 用以阻止吸血鬼的攻势。这里还不需要实现武器的效果, 只要能让玩家将武器拖至游戏场地即可。

修改后的 Level1Activity.java 实现了此功能, 如程序清单 8.7 所示。

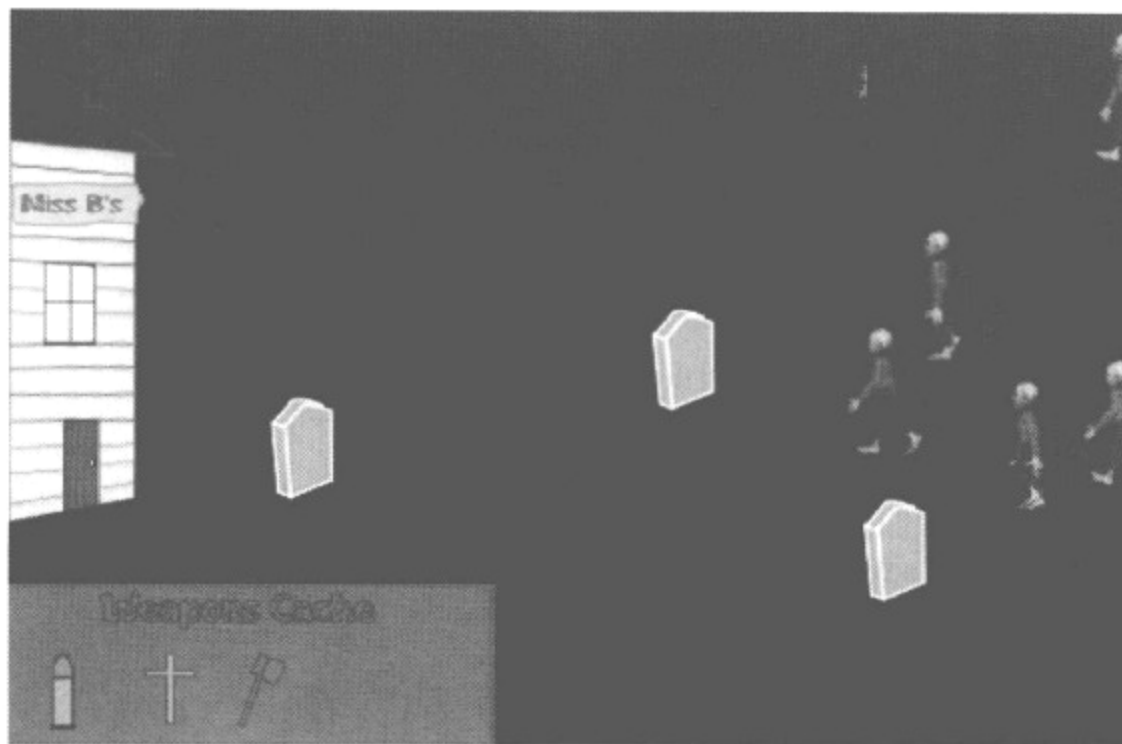


图 8.2 V3 游戏第 1 关的场景

程序清单 8.7 修改后的 Level1Activity.java 代码

```
package com.pearson.lagp.v3;
...
@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());
    final Scene scene = new Scene(1);

    /* 将摄像头置中。 */
    final int centerX = (CAMERA_WIDTH -
        mLevel1BackTextureRegion.getWidth()) / 2;
    final int centerY = (CAMERA_HEIGHT -
        mLevel1BackTextureRegion.getHeight()) / 2;

    /* 创建 Sprite 对象并将其加入 Scene 对象中。 */
    final Sprite background = new Sprite(centerX, centerY,
        mLevel1BackTextureRegion);
    scene.getLastChild().attachChild(background);
    final Sprite obstacleBox = new Sprite(0.0f, CAMERA_HEIGHT -
        mBoxTextureRegion.getHeight(), mBoxTextureRegion);
    scene.getLastChild().attachChild(obstacleBox);
    final Sprite bullet = new Sprite(20.0f, CAMERA_HEIGHT -
```

```

40.0f, mBulletTextureRegion){
@Override
public boolean onAreaTouched(
    final TouchEvent pAreaTouchEvent,
    final float pTouchAreaLocalX,
    final float pTouchAreaLocalY) {
    switch(pAreaTouchEvent.getAction()) {
    case TouchEvent.ACTION_DOWN:
        Toast.makeText(Level1Activity.this,
            "Sprite touch DOWN",
            Toast.LENGTH_SHORT).show();
        break;
    case TouchEvent.ACTION_UP:
        Toast.makeText(Level1Activity.this,
            "Sprite touch UP",
            Toast.LENGTH_SHORT).show();
        break;
    case TouchEvent.ACTION_MOVE:
        this.setPosition(pAreaTouchEvent.getX() -
            this.getWidth() / 2,
            pAreaTouchEvent.getY() -
            this.getHeight() / 2);
        break;
    }
    return true;
}
};
bullet.registerEntityModifier(
    new SequenceEntityModifier(
        new ParallelEntityModifier(
            new MoveYModifier(3, 0.0f,
                CAMERA_HEIGHT - 40.0f,
                EaseQuadOut.getInstance() ),
            new AlphaModifier(3, 0.0f, 1.0f),
            new ScaleModifier(3, 0.5f, 1.0f)
        ),
        new RotationModifier(3, 0, 360)
    )
);
scene.registerTouchArea(bullet);
scene.setTouchAreaBindingEnabled(true);
scene.getLastChild().attachChild(bullet);

final Sprite cross = new Sprite(bullet.getInitialX() +
    40.0f, CAMERA_HEIGHT - 40.0f, mCrossTextureRegion){
@Override
public boolean onAreaTouched(
    final TouchEvent pAreaTouchEvent,
    final float pTouchAreaLocalX,
    final float pTouchAreaLocalY) {
    switch(pAreaTouchEvent.getAction()) {

```



```

        case TouchEvent.ACTION_DOWN:
            Toast.makeText(LevellActivity.this,
                "Sprite touch DOWN",
                Toast.LENGTH_SHORT).show();
            break;
        case TouchEvent.ACTION_UP:
            Toast.makeText(LevellActivity.this,
                "Sprite touch UP",
                Toast.LENGTH_SHORT).show();
            break;
        case TouchEvent.ACTION_MOVE:
            this.setPosition(pAreaTouchEvent.getX() -
                this.getWidth() / 2,
                pAreaTouchEvent.getY() -
                this.getHeight() / 2);
            break;
    }
    return true;
}
};

cross.registerEntityModifier(
    new SequenceEntityModifier(
        new ParallelEntityModifier(
            new MoveYModifier(4, 0.0f,
                CAMERA_HEIGHT - 40.0f,
                EaseQuadOut.getInstance() ),
            new AlphaModifier(4, 0.0f, 1.0f),
            new ScaleModifier(4, 0.5f, 1.0f)
        ),
        new RotationModifier(2, 0, 360)
    )
);

cross.registerEntityModifier(
    new AlphaModifier(10.0f, 0.0f, 1.0f));
scene.registerTouchArea(cross);
scene.getLastChild().attachChild(cross);

final Sprite hatchet = new Sprite(cross.getInitialX() +
    40.0f, CAMERA_HEIGHT - 40.0f, mHatchetTextureRegion){
    @Override
    public boolean onAreaTouched(
        final TouchEvent pAreaTouchEvent,
        final float pTouchAreaLocalX,
        final float pTouchAreaLocalY) {
        switch(pAreaTouchEvent.getAction()) {
            case TouchEvent.ACTION_DOWN:
                Toast.makeText(LevellActivity.this,
                    "Sprite touch DOWN",
                    Toast.LENGTH_SHORT).show();
                break;
            case TouchEvent.ACTION_UP:

```

```

        Toast.makeText(Level1Activity.this,
            "Sprite touch UP",
            Toast.LENGTH_SHORT).show();
        break;
    case TouchEvent.ACTION_MOVE:
        this.setPosition(pAreaTouchEvent.getX() -
            this.getWidth() / 2,
            pAreaTouchEvent.getY() -
            this.getHeight() / 2);
        break;
    }
    return true;
}
};
hatchet.registerEntityModifier(
    new SequenceEntityModifier(
        new ParallelEntityModifier(
            new MoveYModifier(5, 0.0f,
                CAMERA_HEIGHT - 40.0f,
                EaseQuadOut.getInstance() ),
            new AlphaModifier(5, 0.0f, 1.0f),
            new ScaleModifier(5, 0.5f, 1.0f)
        ),
        new RotationModifier(2, 0, 360)
    )
);
hatchet.registerEntityModifier(
    new AlphaModifier(15.0f, 0.0f, 1.0f));
scene.getLastChild().attachChild(hatchet);
scene.registerTouchArea(hatchet);
scene.registerEntityModifier(
    new AlphaModifier(10, 0.0f, 1.0f));

    // 加入第一个吸血鬼 (其他的也会随之加入)
    nVamp = 0;
    mHandler.postDelayed(mStartVamp, 5000);
    return scene;
}
...
}

```

onLoadScene() 方法的修改主要有两处:

- 在创建表示武器的 Sprite 对象时, 通过匿名内部类的方式, 覆写其 onAreaTouched() 回调方法以捕捉触摸事件。现在对于触摸事件的处理与程序清单 8.3 一样, 都是简单地弹出桌面通知, 显示玩家是按下屏幕还是松开屏幕。随着游戏制作的推进, 这些桌面通知会被逐步去除。此处的三个 onAreaTouched() 方法完全相同, 可以将其重构提取出来, 但是这里选择分开写, 为的是方便稍后修改其中的代码, 使之分别完成不同的事情。在每个方法的 switch 语句内, 都有一个

ACTION\_MOVE 分支, 其中的代码会在拖动 Sprite 对象时改变其位置。

- ❑ 对于每一个表示武器的 Sprite 对象, 代码都将其作为参数, 调用 Scene 对象的 registerTouchArea() 方法, 将 Sprite 对象所在区域注册为触摸区域, 然后再通过呼叫 setTouchAreaBindingEnabled(), 将触摸区域绑定到精灵对象上, 这样在按下精灵之后产生的拖动事件将投递给对应的 Sprite 对象。

## 8.3 总结

通过本章可以看到, Android 系统与 AndEngine 提供了非常丰富的用户输入处理功能, 这种多样性有时显得太复杂了。其中大多数输入方式是基于触摸的, 这也使得跟踪触摸事件的传播与捕捉触摸事件变得颇具挑战性。

作为游戏设计者, 首要任务是将玩家输入映射为游戏中的操作, 因此, 需要提供最为友好的用户界面。现在简要地重述一下 Android 系统与 AndEngine 支持的输入方式:

- ❑ 触摸
  - ❑ 场景触摸
  - ❑ 区域 (Sprite 对象) 触摸
- ❑ 多点触摸
- ❑ 手势
- ❑ 屏幕手柄
  - ❑ 类比手柄
  - ❑ 方向键手柄
- ❑ 语音识别
- ❑ 加速计
- ❑ 定位
- ❑ 方向

对于上述的大部分触摸方式, AndEngine 不仅提供了封装之后更加易用的接口, 而且通过对象池技术管理了资源消耗, 使得触摸动作不会产生大量的触摸事件对象, 从而避免了频繁的垃圾回收。这些封装之后的接口使得开发者能够迅速做出优秀的游戏来。

## 8.4 习题

1. 写一个简单的范例游戏, 使玩家通过单点触摸式手柄来控制 Sprite 对象在屏幕上移动。

2. 写一个范例游戏，让玩家通过类比手柄来移动屏幕上的 Sprite 对象。当对象移出屏幕边界后，使其从另一端重新进入屏幕。

3. 写个简单的游戏程序，根据手机是处于平躺状态还是直立状态（使用手机时通常都是将其竖着拿的）来改变屏幕的背景色。若手机平躺，则显示绿色背景；如果是直立，则显示红色。



## 第 9 章

# 瓦片地图

- 9.1 为何使用瓦片地图
- 9.2 瓦片地图的类型
- 9.3 瓦片地图的结构
- 9.4 AndEngine 中的瓦片地图
- 9.5 瓦片编辑器: Tiled
- 9.6 TMX 文件
- 9.7 正交瓦片地图游戏:《打吸血鬼》
- 9.8 等距投影瓦片地图
- 9.9 总结
- 9.10 习题

不管是否注意过，读者一定曾在玩过的游戏中看到过瓦片地图<sup>①</sup>。许多早期的街机游戏以及电子游戏机上的游戏，都是基于瓦片的，因为它可以适应许多种游戏的需求。

本章将讲述如何用 AndEngine 来管理瓦片，并介绍一款搭配 AndEngine 使用的瓦片地图编辑器，它可使开发者用别处获得的瓦片集来制作自己的瓦片。V3 游戏中将会加入一个使用瓦片地图来实现的小游戏。

## 9.1 为何使用瓦片地图

瓦片地图对于早期的电脑游戏来说，具有两大无可取代的优势，且对于当前的游戏开发来说，这两大优势依然存在：

- 许多游戏的设计者都需要足够大且可以扩展的游戏地图。相对于创建一张巨大的位图来说，瓦片可以被多次使用，拼接成大图，从而可以在游戏运行时根据需要即刻生成大小不限的地图。瓦片地图既节省了内存，又节省了时间，因为相同的瓦片会在很多地方重复绘制，可以提高绘图效率。这恰恰是很多非游戏领域的地图绘制等程序中使用瓦片的原因，该技术可以不需要在内存中一次性地载入整张大地图。
- 瓦片地图可以简化对碰撞的检测。现在的游戏需要用到很多类型的碰撞检测，如果使用瓦片地图的话，检测的执行只需耗费很少资源。

## 9.2 瓦片地图的类型

瓦片地图是由若干反复出现的多边形小图片以矩阵的方式组合而成的。对于简单的瓦片地图来说，多边形通常是正方形。因此整张地图可以看做一张表格，其中每个单元格对应于瓦片集中的某个瓦片，这样所有单元格合起来就构成了整幅地图图像。图 9.1 演示了使用本章稍后将会介绍的编辑器来编辑瓦片地图，该地图是一张俯视的沙漠场景。

整幅地图都是用图 9.2 所示瓦片集中的瓦片拼合而成的。

使用这 48 块瓦片，游戏设计者就可以按照需要拼合成一张具有各种风貌及障碍物的沙漠大地图来。游戏仅需保存此瓦片集图像，并附上描述每个单元格所用瓦片索引的地图文件即可。

<sup>①</sup> 原文为 tile map，中文一般称为“瓦片地图”或“切片地图”。——译者注



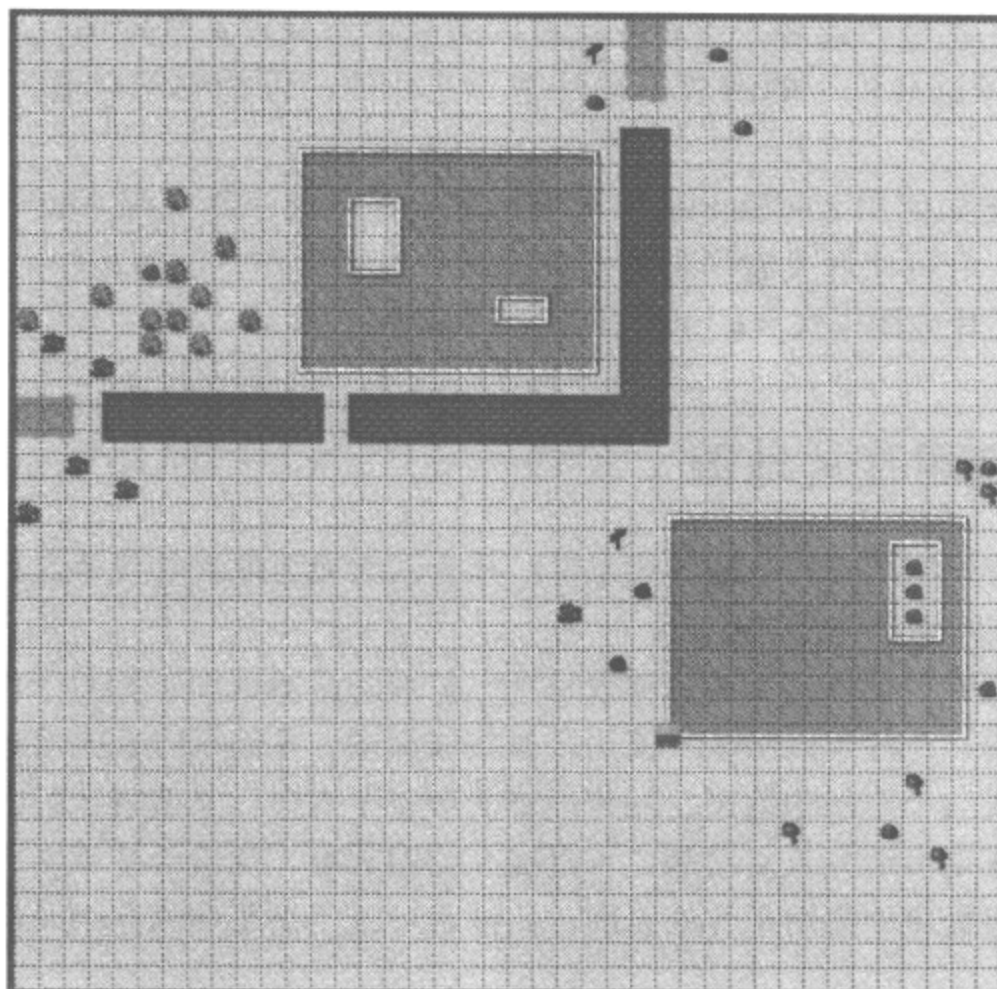


图 9.1 使用 Tiled 编辑沙漠场景的瓦片地图

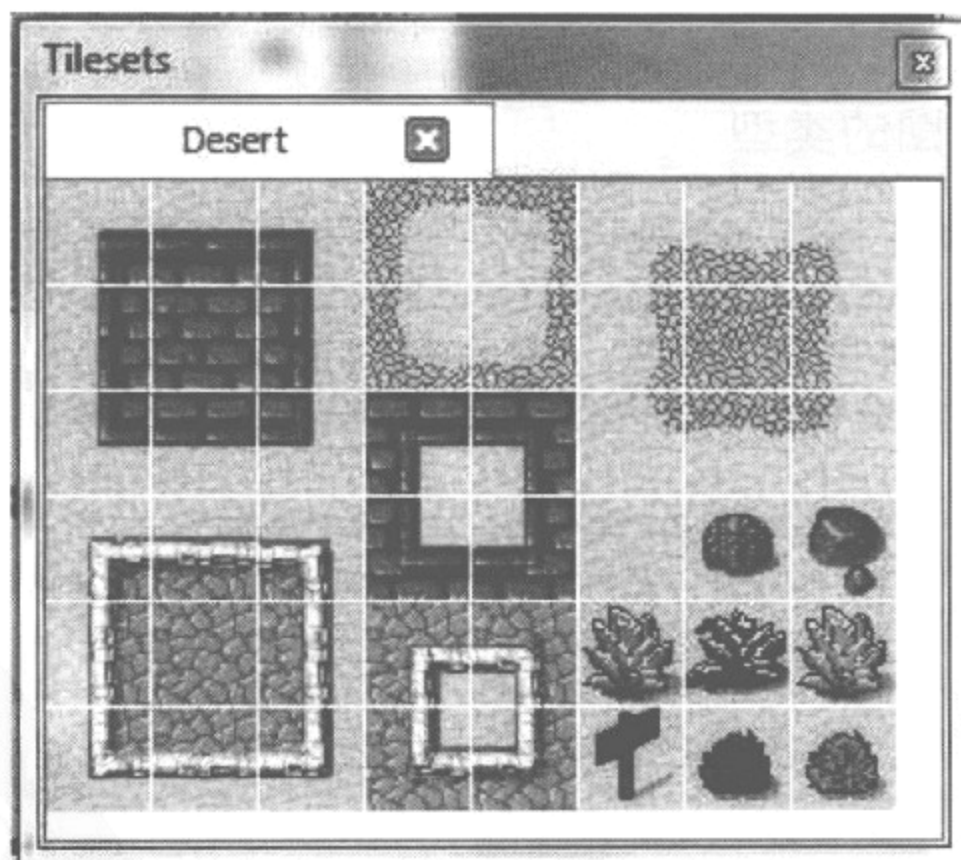


图 9.2 沙漠地图所使用的瓦片集

### 9.2.1 正交瓦片地图

上一节讲到的瓦片地图是正交（Orthogonal）瓦片地图，视角是在地图的正上方，而且瓦片基本是正方形的，偶尔也会遇到由普通矩形拼接的瓦片地图。此种瓦片地图是最先被使用的瓦片地图，并且用于大量游戏中，同时它也是最简单的瓦片地图类型，因为其瓦片图像绝不会重叠。在制作“V3 游戏中的小游戏”时，将会用到正交瓦片地图。

### 9.2.2 等距投影瓦片地图<sup>①</sup>

另一种流行的地图类型使用等距投影瓦片，这种地图的基本原理是通过移动取景摄像头的位置来达成伪 3D 效果。此时的摄像头不是在正上方，而是面对取景物体沿 y 坐标轴方向旋转 45 度，再沿 x 轴向上旋转约 35.264 度<sup>②</sup>。图 9.3 展示了一幅正在用瓦片地图编辑器进行编辑的森林场景地图。

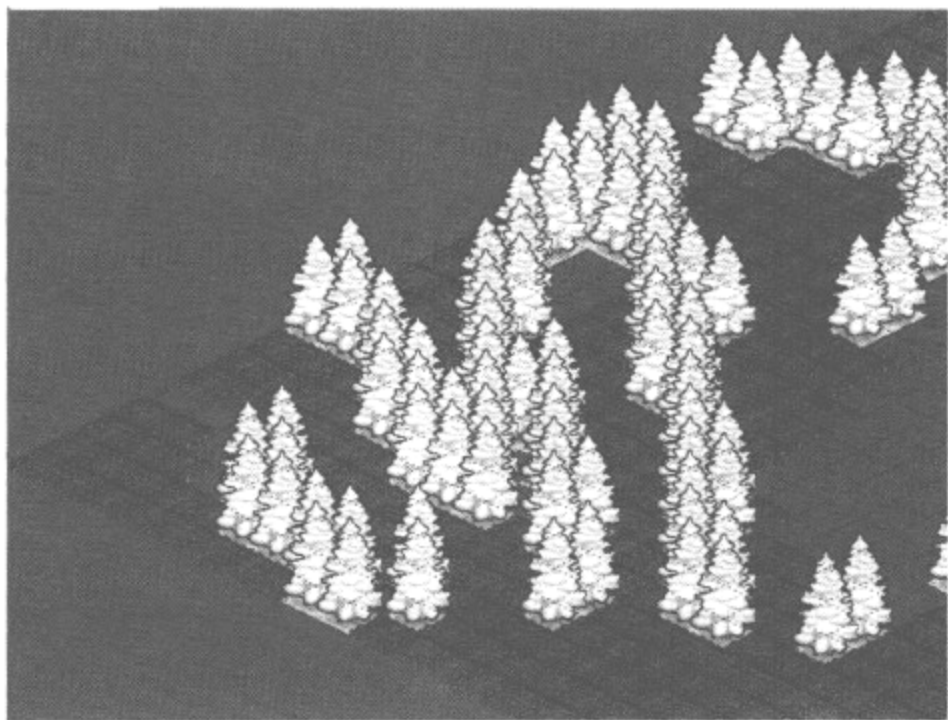


图 9.3 等距投影瓦片地图

视角的转换引发了两个效果：

- 正交视角下的瓦片由正方形变为菱形（钻石形）。
- 近处的瓦片上的物体会叠加在稍远的瓦片之上，在图 9.3 的范例中，近处的树覆盖了远处瓦片上的树。

① 原文为 isometric tile map，中文为“等距投影瓦片地图”，俗称“斜 45 度角地图”、“120 度角地图”、“伪 3D 地图”、“2.5D 地图”等等，意为被绘制的立方体其投影为每条边距离均相等的正六边形，具体可参见：[http://en.wikipedia.org/wiki/Isometric\\_projection](http://en.wikipedia.org/wiki/Isometric_projection)。——译者注

② 此处所说的坐标轴和角度都是针对 3D 空间的。具体细节仍参见前一个注释中的网址。——译者注

等距投影瓦片所制造的 3D 效果当然无法与使用 OpenGL 等手段制作的真 3D 游戏相比，不过考虑到此方法耗费资源很小，故而也算一种很高效的解决方案。尽管在 V3 范例游戏中不会使用这种地图，不过读者应当明白，正交瓦片地图所使用的一些技巧，对于等距投影瓦片地图同样适用。

## 9.3 瓦片地图的结构

瓦片地图由三部分组成：地图本身、瓦片集，以及用于创建瓦片贴图的图形文件。地图是以分层的方式创建的：

- 每个瓦片都有其属性，包含位置 ID（在地图中的行列位置）以及全局 ID（在瓦片集中的贴图位置）。
- 瓦片被拼装为图层。每个图层都是一张完整的地图横截面，其上覆盖了这片 2D 区域的所有瓦片。
- 所有图层合起来构成了整幅地图。

在瓦片地图中也允许构建对象组，它是一组连续瓦片，开发者可基于某种原因，将其当做单一的实体对待。对象组可以实现为地图上的某个矩形区域，地图、图层、对象组都有其各自的属性，这些属性都是键 - 值对，可以通过其后将要介绍的 Tiled Qt 编辑器来编辑，并在游戏运行时查询。

## 9.4 AndEngine 中的瓦片地图

在游戏中使用瓦片地图已经有很长一段时间了，在这个过程中，形成了一套标准的有关瓦片集与瓦片地图的文件格式。有很多免费的瓦片地图编辑器可用于构建及编辑基于该标准的地图。

### 9.4.1 TMX 与 TSX 文件

有两种非常流行的基于 XML 的文件格式，分别是用于瓦片地图的 TMX 格式以及用于瓦片集的 TSX 格式。这两种格式是搭配使用的，大部分 TMX 文件都会引用 TSX 文件来获取瓦片集信息。本章稍后讲解 Tiled 瓦片地图编辑器时，将深入研究这两种文件格式。

TMX 瓦片地图很容易同 AndEngine 相结合，在稍后的几节中将会看到，AndEngine 所提供的用于加载、操作及渲染瓦片地图的 API 非常直观易用。

### 9.4.2 TMXLoader 类

AndEngine 中的 TMXLoader 类知道如何将 TMX 与 TSX 文件载入 TMXTiledMap 对

象中去。开发者可以使用构造器来创建该类对象，其中最完整的构造器如下：

```
TMXLoader(final Context pContext, final TextureManager pTextureManager,
          final TextureOptions pTextureOptions,
          final ITMXTilePropertiesListener pTMXTilePropertyListener)
```

其中的 Context 与 TextureManager 参数是必需的，TextureOptions 与 ITMXTilePropertiesListener 参数是可选的。TextureOptions 的含义与 Sprite 类构造器一样，指示 OpenGL 如何绘制瓦片贴图，如果不指定其他值的话，默认为 NEAREST\_PREMULTIPLYALPHA。

ITMXTilePropertiesListener 参数所指的监听器会在瓦片被载入时回调，在稍后介绍 Tiled Qt 编辑器时，将会看到这些属性只不过是简单的键 - 值对而已。使用该监听器的范例代码也将在稍后提供。

创建好 TMXLoader 对象之后，便可使用如下载入方法通过它来加载瓦片地图了：

```
TMXTiledMap loadFromAsset(final Context pContext, final String pAssetPath)
TMXTiledMap load(final InputStream pInputStream)
```

使用第一个方法，即 loadFromAsset()，会更加方便，除非开发者基于某种原因，需要打开 .tmx 文件另作他用。在解析与加载过程中如果出错，两个方法都会抛出 TMXLoaderException 异常。方法返回的 TMXTiledMap 是访问所有瓦片地图信息所使用的根节点。

### 9.4.3 TMXTiledMap 类

TMXTiledMap 类是 AndEngine 用于表示整个瓦片地图的类。它包含所有图层、瓦片及其相关属性。通常不需要直接调用构造器去构建整个地图，而是通过调用上述加载地图方法而获得。编程中，经常使用如下 getter 方法来获取瓦片地图的内容：

```
int getTileColumns()
int getTileRows()
int getTileWidth()
int getTileHeight()
```

以上方法会分别返回对应的长宽信息以供使用，其中 width 与 height 都是以像素为单位的，代表单个瓦片的大小。

```
ArrayList<TMXTileSet> getTMXTileSets()
ArrayList<TMXLayer> getTMXLayers()
ArrayList<TMXObjectGroup> getTMXObjectGroups()
```

以上方法分别返回包含该瓦片地图瓦片集、图层、对象组的列表。这些对象都有相应的方法来获取更加具体的地图相关信息。

```
TMXProperties<TMXTileProperty> getTMXTilePropertiesByGlobalTileID(
```

```

    final int pGlobalTileID)
    TMXProperties<TMXTiledMapProperty> getTMXTiledMapProperties()
    TMXProperties<TMXTileProperty> getTMXTileProperties(
        final int pGlobalTileID)

```

以上方法可以获取到地图对象中地图、图层或瓦片的属性列表。

```
TextureRegion getTextureRegionFromGlobalTileID(final int pGlobalTileID)
```

以上方法可以获取载入瓦片地图中某块瓦片所用的 TextureRegion 对象。

## 9.4.4 TMXLayer 类

瓦片地图是由图层组成的，每个图层都有自己的瓦片。前面讲到，开发者可以从 TMXTiledMap 对象上获取地图对象的图层列表。可以在图层对象上调用相关方法来获取或修改该图层的相关信息：

```

String getName()
int getTileColumns()
int getTileRows()

```

以上方法可以获取图层对象的基本信息，其名称可在 Tiled Qt 编辑器中设置。

```

TMXTile[][] getTMXTiles()
TMXTile getTMXTile(final int pTileColumn, final int pTileRow)
TMXTile getTMXTileAt(final float pX, final float pY)

```

以上三个方法都可用来获取瓦片。第一个将所有的瓦片放入 TMXTile 类型的二维数组中，第二个根据行列坐标来获取瓦片，最后一个根据屏幕的点坐标来获取其下对应的瓦片。

```
TMXProperties<TMXLayerProperty> getTMXLayerProperties()
```

此方法提供本图层的属性列表。

## 9.4.5 TMXTile 类

有了对 TMXTile 对象的引用，就可在其上调用相关 getter/setter 方法，以获取感兴趣的信息：

```

int getTileRow()
int getTileColumn()
int getTileX()
int getTileY()
int getTileWidth()
int getTileHeight()
TextureRegion getTextureRegion()

```

以上方法都很简明，无须解释。读者可能认为，应该使用 getTextureRegion() 方法



来判断并改变被关联至此瓦片对象的图像，不过，这儿有更好的办法，那就是通过全局 ID 来查询：

```
int getGlobalTileID()
void setGlobalTileID(final TMXTiledMap pTMXTiledMap, final int pGlobalTileID)
```

全局 ID 是由 Tiled Qt 编辑器（下一节会讲到）赋予瓦片地图中的瓦片的。尽管开发者可以指定 Tiled Qt 编辑器从哪个序号开始编列瓦片，但是一般来说，都是从默认值 0 开始的。瓦片集中的每个瓦片图像都有其独特的全局 ID。当使用 setGlobalTileID() 来修改全局 ID 时，对应的瓦片贴图也会修改，可以通过此方法在运行时改变某个瓦片对象的图像。

## 9.5 瓦片编辑器：Tiled

AndEngine 知道如何解析用 Tiled Qt 编辑器创建的瓦片地图文件。在第 2 章中曾简单地提了一下它，现在来详细介绍它的用法，并用其构建 V3 游戏中所用的瓦片地图。Tiled 支持正交瓦片地图与等距投影瓦片地图，这里只关注在 V3 范例游戏中所使用的正交瓦片地图。

如果读者还没有安装 Tiled 的话，请从 [www.mapeditor.org](http://www.mapeditor.org) 下载并安装。请确认安装的是“Tiled Qt”版本，此版本以 C++ 语言编写，并用了 Nokia 公司的 Qt 图形库。读者可能还会碰到稍早的 Java 版本，不过此版本已经不再更新了。本书的范例均是使用 Tiled Qt 0.6.0 版本来制作的。

运行 Tiled 并打开位于 examples 子目录下的 desert.tmx 文件后，如图 9.4 所示的软件工作区就会出现。

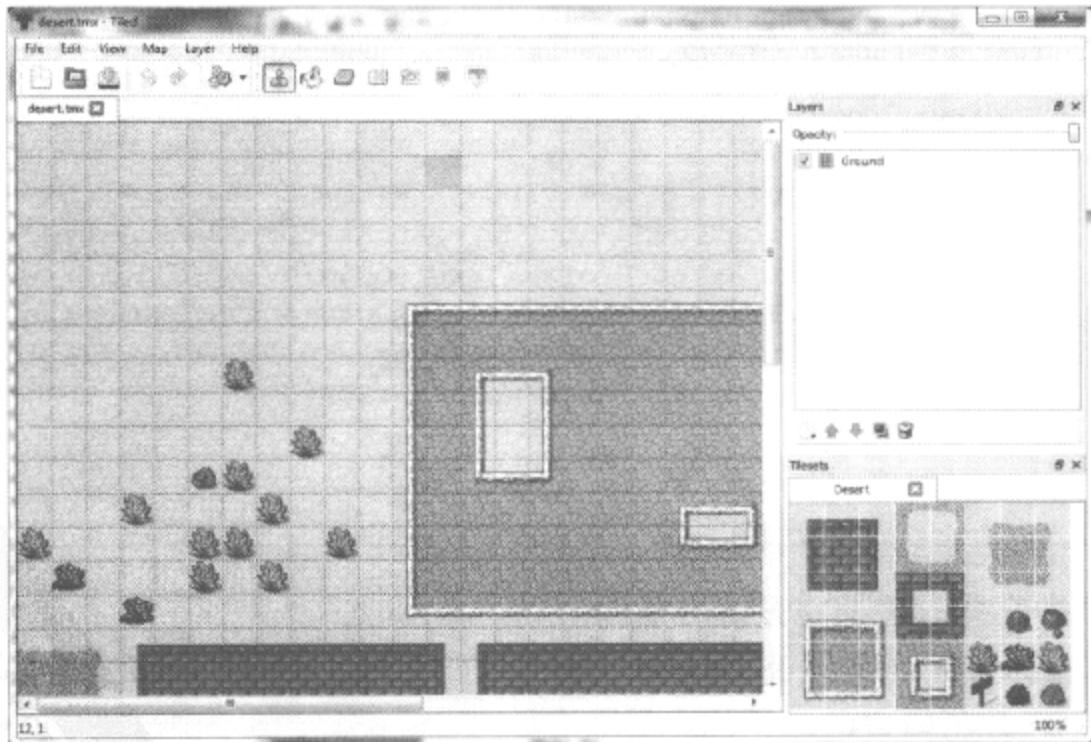


图 9.4 Tiled 软件的工作区



默认的编辑工具是“图章”(stamp)。从右下角的瓦片集中选择瓦片,然后单击主工作区中的某一点,就会将选定的瓦片贴至该处。需要保存文件时,可以执行 File 菜单下的“Save”或“Save As...”菜单项,将当前工作保存为 .tmx 文件,这样 AndEngine 就可以用它来绘制瓦片地图了。右击瓦片集将会弹出菜单,在其中可以编辑瓦片集,或将其保存为 .tsx 文件。

Tiled 工作区右上角的 Layers 面板列出了当前地图的所有图层。图中所编辑的这份地图,仅有一个名为 Ground 的图层,它即是当前正在显示的图层。

## 9.6 TMX 文件

Tiled 及 AndEngine 所使用的文件格式都是 TMX,所有的贴图地图文件均以 .tmx 为扩展名。它所使用的瓦片集图像将以 png 格式随之一同存放。TMX 文件是基于 XML 的,其中包含了地图上每个位置所对应的瓦片信息,为了减少数据量,这些信息是经过压缩的。图 9.4 所编辑的沙漠场景仅用了 845 个字节。程序清单 9.1 提供了完整的文件内容。

程序清单 9.1 desert.tmx

```
<?xml version="1.0" encoding="UTF-8"?>
<map version="1.0" orientation="orthogonal" width="40" height="40"
tilewidth="32" tileheight="32">
<tilesheet firstgid="1" name="Desert" tilewidth="32" tileheight="32"
spacing="1" margin="1">
<image source="tmw_desert_spacing.png" width="265" height="199"/>
</tilesheet>
<layer name="Ground" width="40" height="40">
<data encoding="base64" compression="zlib">
eJztmNkKwjAQRaN9cAPrAq5Yq3Xf6v9/nSM2VibQJjEZR+nDwQZScrwztoORECLySBcIgz7nc2
y4KfyWDLx+Jb9nViNgDEwY+KioAXUgQN4+zpoCMwPmQAtOAx2CLFbA2oDEo9+hwG8DnIDtF/
2K8ks086Tw2zH0uyMv7HcRr/6/EvvhnsPrsrwxX7rwU/0ODig/eV3mh3N1ld8eraWPax6+
64s9McesfrqcHfglMpoifxcVEWjukyW+9AtFPl/I71pER3Of6j4bv7HI54s+MChhqLlPdZ/
P3qMmFuo5h5NnTOhjM5tReN2yT5ln5/v7J3F0vi46fk+ne7ax0i9l6If7mpufTX3f5wsqv9
TAD2fJLT9VrTn7UeZnM5tR+v0LMQOHXwFnxe2/warGFRWf8QDjOLfP
</data>
</layer>
</map>
```

文件内容中有些有趣的地方值得强调:

- width 和 height 都是以瓦片为单位计算的。所以这个沙漠场景地图是由 40 行、40 列的瓦片所组成的。
- <tilesheet> 元素描述了瓦片集, <image source="tmw\_desert\_spacing.png"> 这部分

描述了瓦片集所用的图片文件，此文件中是 `tmw_desert_spacing.png`。瓦片集也可以用单独的 XML 文件描述，其后缀名为 TSX。在稍后的 V3 游戏制作中，将会使用 TSX 文件的瓦片集。

- `tileWidth` 与 `tileHeight` 均是以像素为单位的，所以沙漠地图的瓦片大小是  $32 \times 32$  像素。
- 此文件仅包含一个 `<layer>` 元素，这表明沙漠地图中只有一个图层，其名字为 `Ground`。
- `Ground` 图层所使用的瓦片数据以 `zlib` 算法压缩 (`compression="zlib"`)。

需要修改其中一些属性，使之符合游戏所需的地图。按照如下数据修改之后，按 OK 按钮继续：

- ☐ Orientation: Orthogonal
- ☐ Map size: width: 15; height: 10
- ☐ Tile size: width: 32; height: 32

现在 Tiled 软件的工作区如图 9.6 所示。

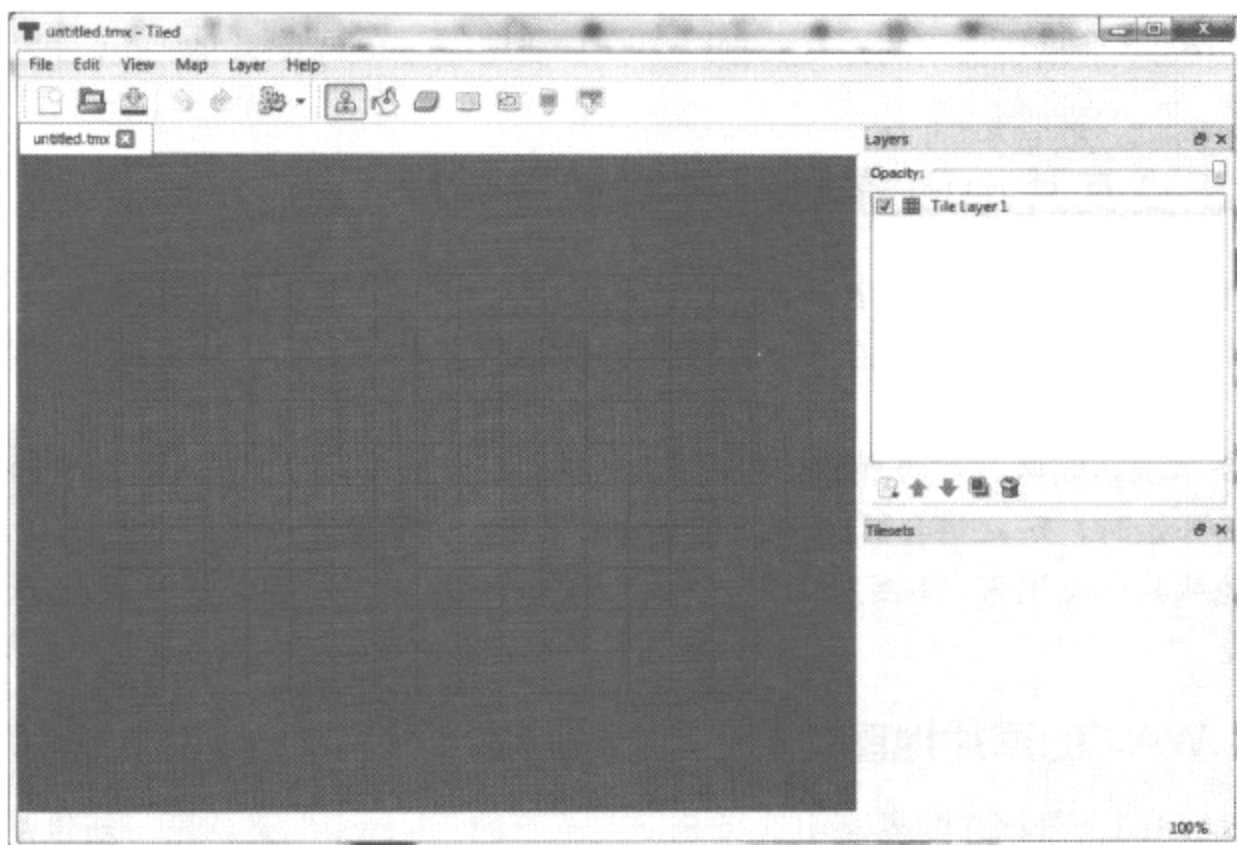


图 9.6 Tiled 软件的空白工作区

### 9.7.2 创建 WAV 的瓦片集

读者可以使用任何所喜好的图形编辑器来制作游戏中用到的瓦片。对于 WAV 游戏，笔者使用 Inkscape 制作了 6 张 32 像素 × 32 像素的瓦片。Tiled Qt 所需的瓦片集图像为 .png 格式，所有的瓦片需要像精灵表那样排列到一张大图上。Zwoptex ([www.zwoptex.com](http://www.zwoptex.com)) 软件对于排列这种小的精灵表很适用，当然，读者也可以使用自己喜好的图形编辑器来将瓦片拼合成一幅大的 .png 文件。WAV 游戏的瓦片集贴图 (.png) 如图 9.7 所示。



图 9.7 WAV 瓦片集图像

### 9.7.3 创建 WAV 的瓦片地图

选择 Map 菜单下的“New Tileset...”菜单项，将弹出如图 9.8 所示对话框。

在该对话框中，通过“Browse...”按钮来指定该瓦片集所使用的图片，然后软件会依此在 Name 文本框中填写瓦片集的名字。要确保瓦片的大小、图像边距（Margin）、瓦片间距（Spacing）等属性与瓦片集图像相符。开发者可以根据需要创建多个瓦片集，不过 WAV 游戏仅需要一个。

在 Tile Sets 面板中，找到表示闭合棺材的那张瓦片（右起第二张），右击它，在弹出菜单中选择“Tile Properties...”菜单项，将会弹出如图 9.9 所示对话框。

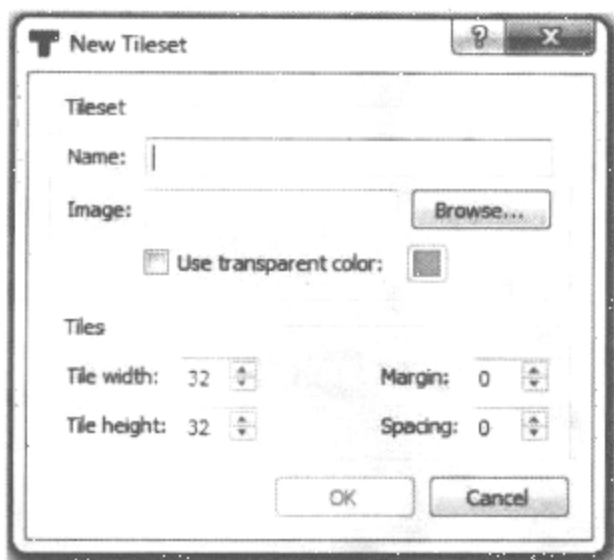


图 9.8 Tiled 软件的 New Tileset 对话框

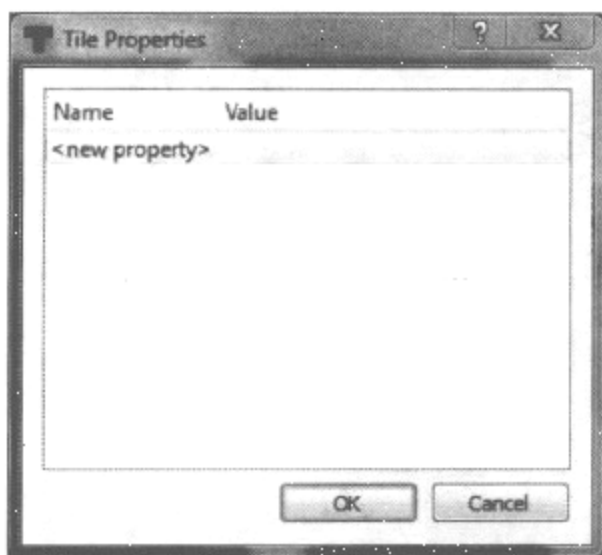


图 9.9 Tiled 软件的 Tile Properties 对话框

现在需要为此瓦片添加新的属性。点击 <new property>，在“Name”和“Value”栏中分别填入 coffin 与 true。游戏载入地图后，将依据此属性来判断某瓦片是否代表棺材。填好后点击 OK 按钮。

在 Tilesets 面板任意处右击鼠标，于弹出式菜单中选择“Export Tileset As...”菜单项，将会弹出对话框，询问导出 TSX 文件的文件名。对于 WAV 游戏，可将其命名为 WAVTileset.tsx。

有了瓦片集，就可以用 Tiled 软件提供的编辑工具来拼接地图了。最终的地图如图 9.10 所示。

拼接地图时，首先使用填充工具（油漆桶）将整张地图用黑色瓦片填充。然后选择图章工具拼出长满草的小路来（路两旁也许有篱笆，但是从这个角度看不到）。可以在地图上通过单击并拖动的方法，将一连串的单元格用同一张选定的瓦片填充。接下来，使用图章工具随机地在地图上安插一些棺材、墓穴以及墓碑。现在不需要加入打开的棺材，它们将在游戏运行时动态地出现。

尽管 WAV 游戏中并不需要对象组，不过如果有需要，也可以在地图上增添对象



组。Tiled 软件的对象组是一片在图层之中划定的不可见矩形区域，可以在游戏运行时用其辨识图层对象。要在图层中创建对象组，可以点击 Layer 菜单下的“Add Object Layer...”命令。

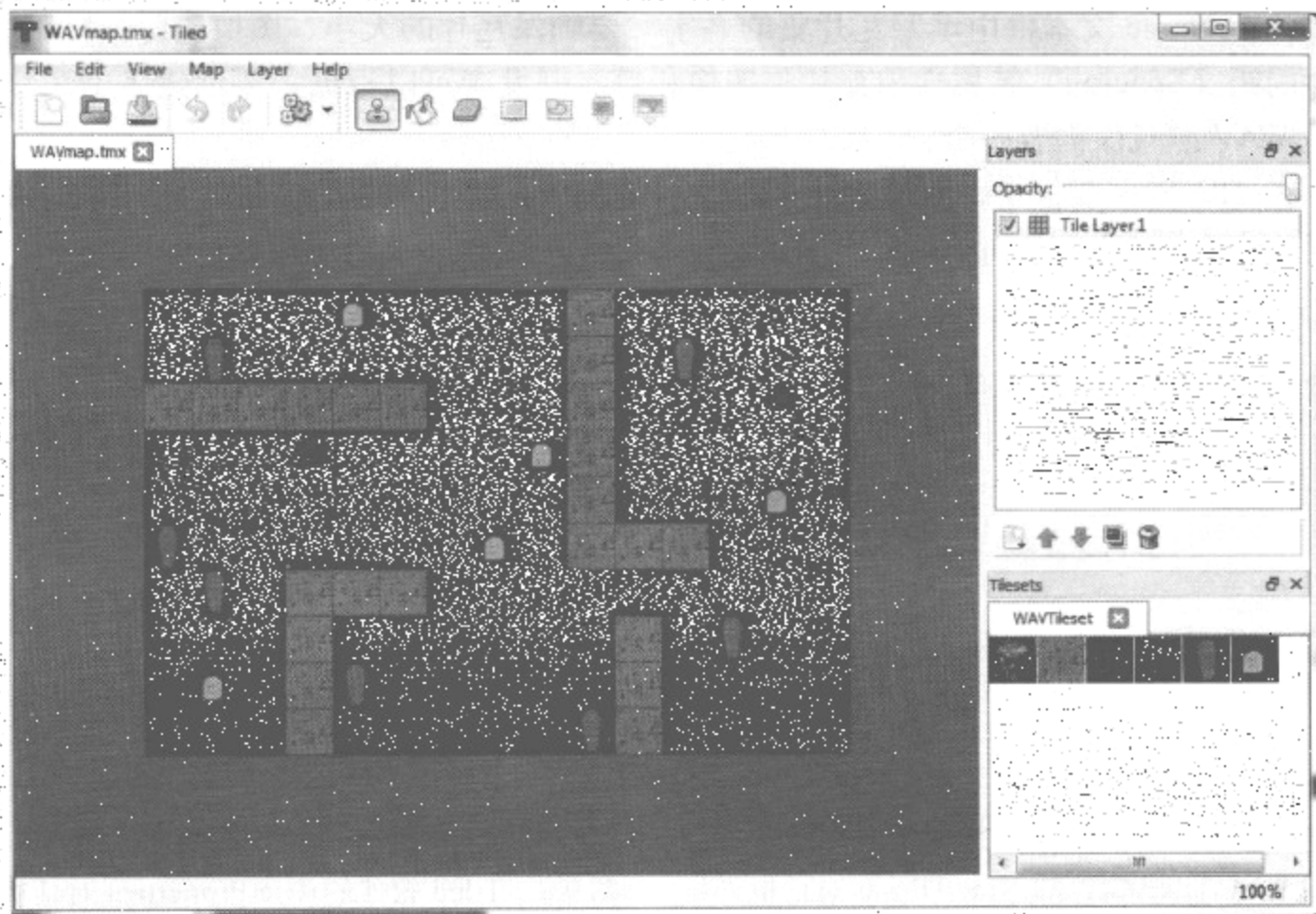


图 9.10 用 Tiled 软件拼接完成的 WAV 游戏瓦片地图

当然，属性的设置也不仅仅局限于瓦片。TMX 文件中的任何组件（包括地图、图层、瓦片、对象组）都可以有关联属性。这些属性都是以键-值对的形式存在的。例如，如果点击 Layer 菜单下的“Layer Properties...”菜单项，则会弹出如图 9.11 所示的对话框。

有时候，需要在保存瓦片地图之前，先以 gzip 压缩其数据。可以点击 Edit 菜单下的“Preferences...”菜单项，这时将会弹出如图 9.12 所示对话框。

开发者需要确认“Store tile layer data as:”选项的值被设定为“Base64(gzip compressed)”。该选项不是默认值，至少在笔者所使用的这个版本中不是。现在可以通过 New 菜单下的“Save”菜单项来保存此地图，这个动作会创建 WhackAVampire.tmx 文件，此文件需要导入游戏项目的 assets 文件夹中。



图 9.11 Tiled 软件的 Layer Properties 对话框

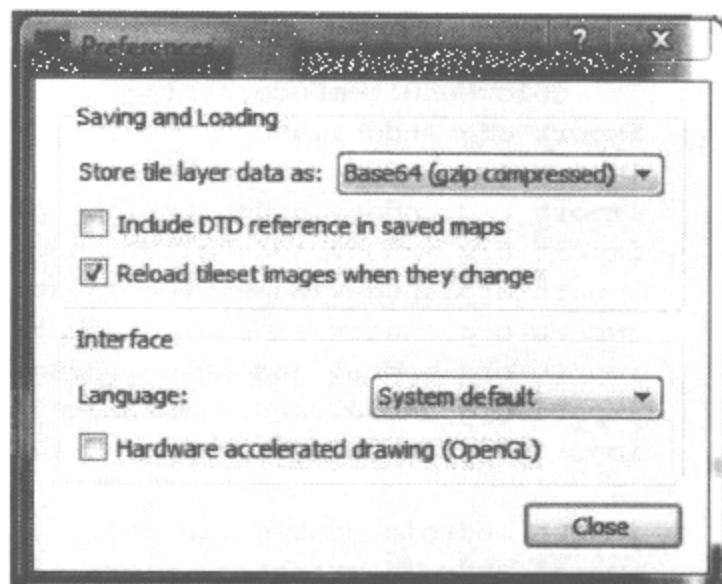


图 9.12 Tiled 软件的 Preferences 对话框

#### 9.7.4 《打吸血鬼》游戏的代码

由于现在还不能在 V3 游戏中完整地玩完第 1 关，所以需要以某种方式直接切换到 WAV 游戏中。此时可以先在游戏的选项菜单里增加一个菜单项，当做小游戏的进入点，等到 V3 游戏的关卡机制彻底做好之后，再将其插入到两个关卡之间。

##### 将 WAV 游戏添加至 OptionsActivity

程序清单 9.2 是新版的 OptionsActivity 类，其中改动之处都以粗体表示。在掌握了第 3 章中菜单制作的方法之后，这些代码修改大多不需要再解释了。

程序清单 9.2 增加 WAV 游戏之后的 OptionsActivity 类

```
package com.pearson.lagp.v3;

import javax.microedition.khronos.opengles.GL10;

import org.anddev.andengine.engine.Engine;
import org.anddev.andengine.engine.camera.Camera;
import org.anddev.andengine.engine.options.EngineOptions;
import org.anddev.andengine.engine.options.EngineOptions
    .ScreenOrientation;
import org.anddev.andengine.engine.options.resolutionpolicy
    .RatioResolutionPolicy;
import org.anddev.andengine.entity.modifier.ScaleModifier;
import org.anddev.andengine.entity.scene.Scene;
import org.anddev.andengine.entity.scene.menu.MenuScene;
import org.anddev.andengine.entity.scene.menu.MenuScene
    .IOnMenuItemClickListener;
import org.anddev.andengine.entity.scene.menu.item.IMenuItem;
```



```

import org.anddev.andengine.entity.scene.menu.item.TextMenuItem;
import org.anddev.andengine.entity.scene.menu.item.decorator
    .ColorMenuItemDecorator;
import org.anddev.andengine.entity.sprite.Sprite;
import org.anddev.andengine.entity.util.FPSLogger;
import org.anddev.andengine.opengl.font.Font;
import org.anddev.andengine.opengl.font.FontFactory;
import org.anddev.andengine.opengl.texture.Texture;
import org.anddev.andengine.opengl.texture.TextureOptions;
import org.anddev.andengine.opengl.texture.region.TextureRegion;
import org.anddev.andengine.opengl.texture.region.TextureRegionFactory;
import org.anddev.andengine.ui.activity.BaseGameActivity;

import android.content.Intent;
import android.graphics.Color;
import android.os.Handler;

public class OptionsActivity extends BaseGameActivity implements
    IOnMenuItemClickListener {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    protected static final int MENU_MUSIC = 0;
    protected static final int MENU_EFFECTS = MENU_MUSIC + 1;
    protected static final int MENU_WAV = MENU_EFFECTS + 1;

    // =====
    // 字段
    // =====

    protected Camera mCamera;

    protected Scene mMainScene;
    protected Handler mHandler;

    private Texture mMenuBackTexture;
    private TextureRegion mMenuBackTextureRegion;

    protected MenuScene mOptionsMenuScene;
    private TextMenuItem mTurnMusicOff, mTurnMusicOn;
    private TextMenuItem mTurnEffectsOff, mTurnEffectsOn;
    private TextMenuItem mWAV;
    private IMenuItem musicMenuItem;
    private IMenuItem effectsMenuItem;
    private IMenuItem WAVMenuItem;

    private Texture mFontTexture;

```

```

private Font mFont;

public boolean isMusicOn = true;
public boolean isEffectsOn = true;

// =====
// 构造器
// =====
// =====
// Getter 和 Setter 方法
// =====

// =====
// 覆写超类及接口中的方法
// =====

@Override
public Engine onLoadEngine() {
    mHandler = new Handler();
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH, CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
}

@Override
public void onLoadResources() {
    /* 载入 Font 与 Textures 对象。 */
    this.mFontTexture = new Texture(256, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);

    FontFactory.setAssetBasePath("font/");
    this.mFont = FontFactory.createFromAsset(
        this.mFontTexture, this, "Flubber.ttf", 32,
        true, Color.WHITE);
    this.mEngine.getTextureManager().loadTexture(this.mFontTexture);
    this.mEngine.getFontManager().loadFont(this.mFont);

    this.mMenuBackTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mMenuBackTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mMenuBackTexture, this,
            "gfx/OptionsMenu/OptionsMenuBk.png", 0, 0);
    this.mEngine.getTextureManager().loadTexture(this.mMenuBackTexture);

    mTurnMusicOn = new TextMenuItem(MENU_MUSIC, mFont,
        "Turn Music On");
    mTurnMusicOff = new TextMenuItem(MENU_MUSIC, mFont,

```

```

        "Turn Music Off");
mTurnEffectsOn = new TextMenuItem(MENU_EFFECTS, mFont,
    "Turn Effects On");
mTurnEffectsOff = new TextMenuItem(MENU_EFFECTS, mFont,
    "Turn Effects Off");

mWAV = new TextMenuItem(MENU_WAV, mFont,
    "Whack A Vampire");
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    this.createOptionsMenuScene();
    /* 将摄像头放在背景中央。 */
    final int centerX = (CAMERA_WIDTH -
        this.mMenuBackTextureRegion.getWidth()) / 2;
    final int centerY = (CAMERA_HEIGHT -
        this.mMenuBackTextureRegion.getHeight()) / 2;

    this.mMainScene = new Scene(1);
    /* 增加背景与静态菜单。 */
    final Sprite menuBack = new Sprite(centerX, centerY,
        this.mMenuBackTextureRegion);
    mMainScene.getLastChild().attachChild(menuBack);
    mMainScene.setChildScene(mOptionsMenuScene);

    return this.mMainScene;
}

@Override
public void onLoadComplete() {
}

@Override
public boolean onMenuItemClicked(final MenuScene pMenuScene,
    final IMenuItem pMenuItem, final float pMenuItemLocalX,
    final float pMenuItemLocalY) {
    switch(pMenuItem.getID()) {
        case MENU_MUSIC:
            if (isMusicOn) {
                isMusicOn = false;
            } else {
                isMusicOn = true;
            }
            createOptionsMenuScene();
            mMainScene.clearChildScene();
            mMainScene.setChildScene(mOptionsMenuScene);
            return true;
        case MENU_EFFECTS:

```

```

        if (isEffectsOn) {
            isEffectsOn = false;
        } else {
            isEffectsOn = true;
        }
        createOptionsMenuScene();
        mMainScene.clearChildScene();
        mMainScene.setChildScene(mOptionsMenuScene);
        return true;
    case MENU_WAV:
        mMainScene.registerEntityModifier(
            new ScaleModifier(1.0f, 1.0f, 0.0f));
        mOptionsMenuScene.registerEntityModifier(
            new ScaleModifier(1.0f, 1.0f, 0.0f));
        mHandler.postDelayed(mLaunchWAVTask, 1000);
        return true;

    default:
        return false;
    }
}

// =====
// 方法
// =====

protected void createOptionsMenuScene() {
    this.mOptionsMenuScene = new MenuScene(this.mCamera);

    if (isMusicOn) {
        musicMenuItem = new ColorMenuItemDecorator(
            mTurnMusicOff, 0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    } else {
        musicMenuItem = new ColorMenuItemDecorator(
            mTurnMusicOn, 0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    }
    musicMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mOptionsMenuScene.addMenuItem(musicMenuItem);

    if (isEffectsOn) {
        effectsMenuItem = new ColorMenuItemDecorator(
            mTurnEffectsOff, 0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    } else {
        effectsMenuItem = new ColorMenuItemDecorator(
            mTurnEffectsOn, 0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    }
    effectsMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mOptionsMenuScene.addMenuItem(effectsMenuItem);
}

```

```

        WAVMenuItem = new ColorMenuItemDecorator(mWAV, 0.5f, 0.5f,
            0.5f, 1.0f, 0.0f, 0.0f);
        WAVMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
            GL10.GL_ONE_MINUS_SRC_ALPHA);
        this.mOptionsMenuScene.addMenuItem(WAVMenuItem);

        this.mOptionsMenuScene.buildAnimations();
        this.mOptionsMenuScene.setBackgroundEnabled(false);
        this.mOptionsMenuScene.setOnMenuItemClickListener(this);
    }

    private Runnable mLaunchWAVTask = new Runnable() {
        public void run() {
            Intent myIntent = new Intent(OptionsActivity.this,
                WAVActivity.class);
            OptionsActivity.this.startActivity(myIntent);
        }
    };

    // =====
    // 内部类与匿名类
    // =====
}

```

### 创建《打吸血鬼》游戏的 Activity 类

循例创建 WAVActivity.java 文件，记得将其加入 manifest 文件，这样 Android 系统就会将其识别为新的 Activity 了。程序清单 9.3 列出了该类的代码，其后附有详细的解释。

程序清单 9.3 WAVActivity.java

```

package com.pearson.lagp.v3;

+imports

public class WAVActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;
}

```

```

private String tag = "WAVActivity";

// =====
// 字段
// =====

private Handler mHandler;

protected Camera mCamera;

protected Scene mMainScene;

private TMXTiledMap mWAVTMXMap;
private TMXLayer tmxLayer;
private TMXTile tmxTile;

private int[] coffins = new int[50];
private int coffinPtr = 0;
private int mCoffinGID = -1;
private int mOpenCoffinGID = 1;

private Random gen;

// =====
// 构造器
// =====

// =====
// Getter 和 Setter 方法
// =====

// =====
// 覆写超类及接口中的方法
// =====

@Override
public Engine onLoadEngine() {
    mHandler = new Handler();
    gen = new Random();
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
}

```



```

    }

    @Override
    public void onLoadResources() {
    }

    @Override
    public Scene onLoadScene() {
        this.mEngine.registerUpdateHandler(new FPSLogger());

        final Scene scene = new Scene(1);
        try {
            final TMXLoader tmxLoader = new TMXLoader(
                this, this.mEngine.getTextureManager(),
                TextureOptions.BILINEAR_PREMULTIPLYALPHA,
                new ITMXTilePropertiesListener() {
                    @Override
                    public void onTMXTileWithPropertiesCreated(
                        final TMXTiledMap pTMXTiledMap,
                        final TMXLayer pTMXLayer,
                        final TMXTile pTMXTile,
                        final TMXProperties<TMXTileProperty>
                            pTMXTileProperties) {
                        if (pTMXTileProperties.containsTMXProperty("coffin", "true")) {
                            coffins[coffinPtr++] =
                                pTMXTile.getTileRow() * 15 +
                                pTMXTile.getTileColumn();
                            if (mCoffinGID < 0) {
                                mCoffinGID =
                                    pTMXTile.getGlobalTileID();
                            }
                        }
                    }
                });
            this.mWAVTMXMap = tmxLoader.loadFromAsset(this,
                "gfx/WAV/WAVmap.tmx");
        } catch (final TMXLoadException tmxle) {
            Debug.e(tmxle);
        }

        tmxLayer = this.mWAVTMXMap.getTMXLayers().get(0);
        scene.getFirstChild().attachChild(tmxLayer);
        scene.setOnSceneTouchListener(new IOnSceneTouchListener() {
            @Override

```

```

public boolean onSceneTouchEvent(
    final Scene pScene,
    final TouchEvent pSceneTouchEvent) {
    switch(pSceneTouchEvent.getAction()) {
        case TouchEvent.ACTION_DOWN:
            /* 获取被触摸的瓦片。 */
            tmxTile = tmxLayer.getTMXTileAt(
                pSceneTouchEvent.getX(),
                pSceneTouchEvent.getY());
            if((tmxTile != null) &&
                (tmxTile.getGlobalTileID() == mOpenCoffinGID)) {
                tmxTile.setGlobalTileID(mWAVTMXMap, mCoffinGID);
            }
            break;
    }
    return true;
}

mHandler.postDelayed(openCoffin, gen.nextInt(2000));
return scene;
}

@Override
public void onLoadComplete() {
}

private Runnable openCoffin = new Runnable() {
    public void run() {
        int openThis = gen.nextInt(coffinPtr);
        int tileRow = coffins[openThis]/15;
        int tileCol = coffins[openThis] % 15;
        tmxTile = tmxLayer.getTMXTileAt(tileCol*32 + 16,
            tileRow*32 + 16);
        tmxTile.setGlobalTileID(mWAVTMXMap, mOpenCoffinGID);
        mHandler.postDelayed(openCoffin, gen.nextInt(4000));
    }
};
}

```

在通常的 imports 语句及变量声明之后，进入了 Engine 设置阶段。本代码与其他 Activity 的重要区别首先出现在 onLoadScene() 方法中。代码先创建新的 TMXLoader 对象，再通过它从 assets 文件夹下加载相关图片以创建瓦片地图并载入 AndEngine。在创建 TMXLoader 对象时，以匿名内部类的手段覆写了 onTMXTileWithPropertiesCreated()

方法。如果地图被填入 coffin 属性值为 true 的瓦片，该方法就会被回调。该方法的代码在发现含有棺材的瓦片时，就将其位置记录入 int coffins[] 数组。

AndEngine 会在 assets 目录下查找相关的 .tmx 文件。有些时候可以通过在 onLoadResources() 方法中调用 setAssetBasePath() 而修改 assets 子目录查找路径。也可以像本范例游戏这样，通过手工编辑 Tiled 软件所生成的 .tmx 和 .tsx 文件，将完整的 assets 路径加入其中。例如，程序清单 9.4 与程序清单 9.5 分别展示了编辑之后的 WAVmap.tmx 与 WAVTileset.tsx 文件。修改后的文件中，source 属性值填入了完整的 assets 路径。这里的文件名有点微妙，尤其是当在另一目录中创建 TMOX 和 TSX 文件时更是如此。

程序清单 9.4 填入完整 assets 子目录路径后的 WAVmap.tmx 文件

---

```
<?xml version="1.0" encoding="UTF-8"?>
<map version="1.0" orientation="orthogonal" width="15" height="10"
    tilewidth="32" tileheight="32">
  <tileset firstgid="1" source="gfx/WAV/WAVTileset.tsx"/>
  <layer name="Tile Layer 1" width="15" height="10">
    <data encoding="base64" compression="gzip">

H4sIAAAAAAAC5VQQQ4AMAQTG/9/8rKDpBF1O/RAlDYWkQ3wVF9o0QtYw6E2zylBd3cV+6ueD5
4nZG38pMvqoK2yoL6b+fX28mv0xjJMXhhvhDtug+XEWAIAAA==

    </data>
  </layer>
</map>
```

---

程序清单 9.5 填入完整 assets 子目录路径后的 WAVTileset.tsx 文件

---

```
<?xml version="1.0" encoding="UTF-8"?>
<tileset name="WAVTileset" tilewidth="32" tileheight="32" spacing="2"
    margin="2">
  <image source="gfx/WAV/WAVTileSet.png" width="256" height="64"/>
  <tile id="4">
    <properties>
      <property name="coffin" value="true"/>
    </properties>
  </tile>
</tileset>
```

---

载入地图后，获取序号为 0 的 Layer 对象（也就是该地图的唯一图层），将其作为子对象加入当前 Scene 对象中。现在调用 setOnSceneTouchListener() 方法在场景对象上注册监听器以捕获触摸事件。当类型为 ACTION\_DOWN 的触摸事件发生时，回调方法通过在 Layer 对象上调用 getTileAt() 方法来获取被触摸的瓦片，然后检查该瓦片的全局 ID 以判断此瓦片是否为打开的棺材，如果是，就用表示封闭棺材的瓦片取代它。

做完如上设置之后,就可以通过投递名为 openCoffin 的任务来安排游戏逻辑的执行。openCoffin 任务将从 coffins 数组中随机选取一个包含封闭棺材的瓦片,将其图像替换为打开的棺材。最后将自己作为下一个任务投递给线程,使其在 4 秒钟内的某个随机时刻再次运行。

运行此 Activity 后,会看到经过某段随机的时间后,屏幕上将会有棺材从封闭状态变为打开状态,如果此时点击打开的棺材,它将重新合上。现在的这个 WAV 算不上一个精彩的小游戏,不过稍后将为其增加声音,并增加一些干扰因素,让玩家更难发现打开的棺材。

## 9.8 等距投影瓦片地图

这种地图可以使游戏看起来具有 3D 效果,从而变得更加有趣,同时又免去了制作 3D 模型与动画的开销,不过这将给游戏的制作和调试带来困难。头一个难题就是如何以正确的透视关系绘制相关的瓦片。其次还有 z 轴的绘制顺序问题,如果处理不好,就会破坏 3D 效果。有些美工喜欢用虚拟摄像机以合适的角度将 3D 模型拍摄渲染为 2D 图像序列,再依此创建等距投影瓦片。这种技术同先前讨论的用 3D 模型创建动画序列没什么不同。

尽管 V3 游戏中不曾使用等距投影瓦片地图,但是 AndEngine 对其支持度很好。开发者可以使用喜欢的图形编辑器创建等距投影瓦片,再用 Zwoptex 将其做成瓦片集,最后用 Tiled Qt 制作瓦片地图。AndEngine 可以解析生成的 .tmx 和 .tsx 文件,开发者依然可以通过 TMXLayer 来访问单个瓦片。

## 9.9 总结

瓦片地图是游戏编程的重要组成部分。正如本章所述,使用 AndEngine 提供的功能,可以将瓦片集与瓦片地图轻松地集成进游戏中来。制作这些瓦片集与瓦片地图所使用的工具都是格式统一且方便易用的(对了,笔者说过它们都是免费的吗? )。

构建瓦片地图的步骤如下所述:

1. 使用喜好的图形编辑器制作所需瓦片。对于正交瓦片地图来说,最后的结果应当是一系列大小相同的 .png 文件。
2. 通过 Zwoptex 等专业工具,或者开发者喜爱的图形编辑器,将小的瓦片图像拼合成瓦片集。构建图像时,可以使用最小尺寸的画布(但要确保长宽均为 2 的整数幂),同时设定好合适的瓦片图像间距(笔者使用 2 像素的间距),最后将图形保存为瓦片集所用的图片文件。

3. 使用 Tiled Qt 创建游戏所需大小的瓦片地图，并将其瓦片尺寸设置为瓦片集中的瓦片大小。将第 2 步所做瓦片集图片导入到软件中，并用其中的瓦片来拼接地图。使用 Tiled Qt 将该瓦片集保存为 .tsx 文件。

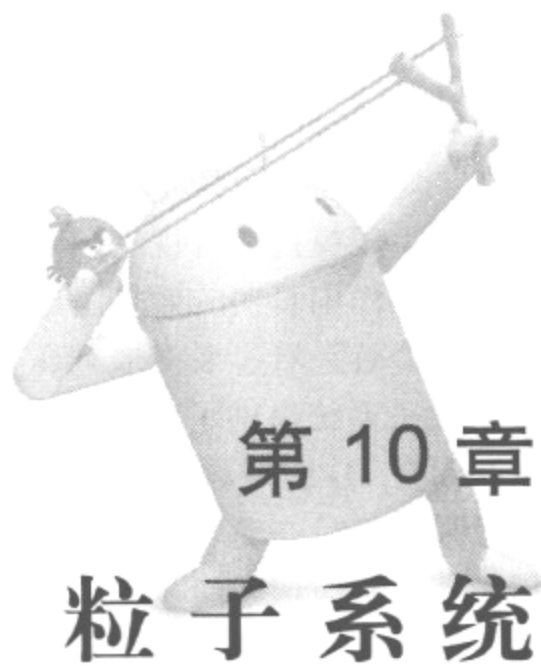
4. 使用 Tiled Qt 将地图存为 TMX 文件，其中的瓦片数据使用 gzip 算法压缩。将 .tmx、.tsx 及 .png 文件导入 Android 项目的 assets 文件夹中。

5. 地图已经准备好了，可以开始使用了。将瓦片地图载入并使用 TMX... 系列方法来访问单个的瓦片，以获取或修改其属性。

## 9.10 习题

1. 使用 Tiled Qt 软件，为 WAVTileset.tmx 中的墓碑瓦片增加名为 “tombstone” 的属性，并将其值设为 “true”。

2. 利用习题 1 中增加的属性，修改 WAVActivity 类，使用户触摸墓碑时，它被一块写有 “Boo!” 字样的瓦片所取代。这块瓦片需要另行制作，同时还要做一份包含它的瓦片集，并使用 Tiled Qt 软件将新的瓦片加入到地图中。



## 第 10 章 粒子系统

- 10.1 粒子发射器是什么
- 10.2 粒子系统如何运作
- 10.3 AndEngine 的粒子系统
- 10.4 创建粒子系统
- 10.5 将粒子发射器加入 V3 游戏中
- 10.6 总结
- 10.7 习题



粒子 (particle) 系统是电脑游戏的另一个组成部分, 就算有时没有明显察觉其存在, 玩家还是会经常看到它。粒子系统解决了一个重要问题, 那就是如何将一整束小粒子的生成模式进行视觉呈现。思考一下火焰、烟雾、下雨等场景, 这些都需要以许多粒子来制作出真实的效果。这些粒子会根据特定的规则进行运动, 同时也要加入随机因素, 使得效果看起来更加逼真。

粒子系统提供了创建粒子发射器的环境, 发射器用来指定单独的粒子效果。粒子系统经常在 3D 游戏环境中使用, 不过 AndEngine 也在范例程序中演示了一些粒子发射器的效果, 同时开发者还可自行创建发射器。本章将会研究范例发射器与定制发射器的使用方法, 并演示如何在 V3 游戏程序中加入粒子发射器。

图 10.1 展示了粒子系统的效果, 这张屏幕截图描述的是加入粒子效果之后的 V3 游戏, 展示了本章将要介绍的粒子强化效果。

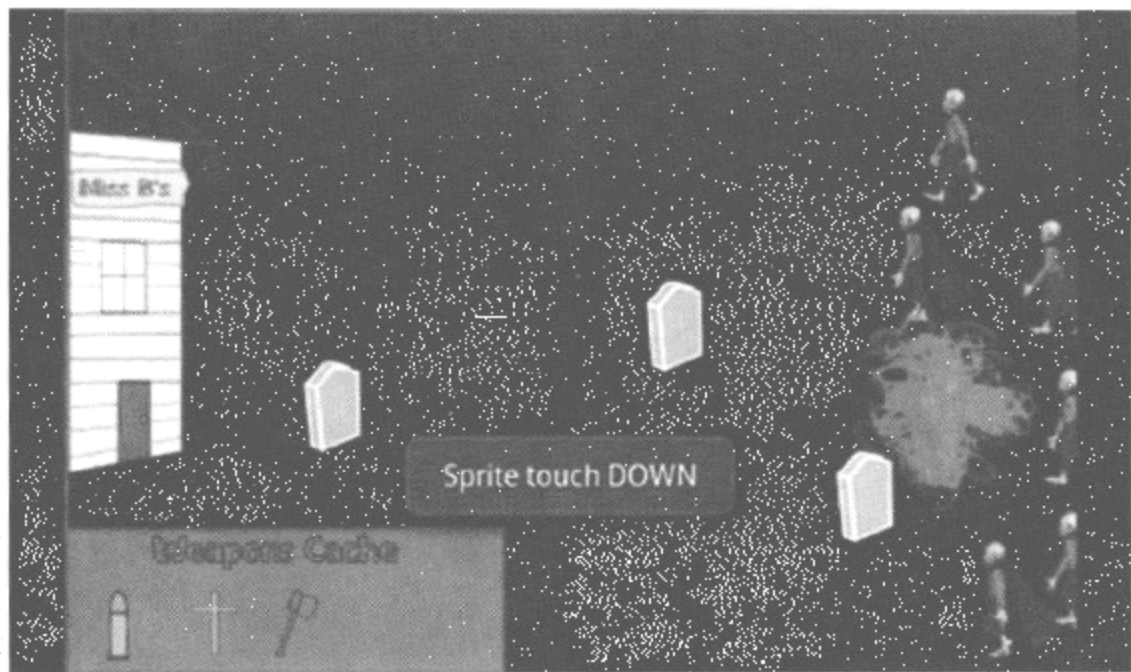


图 10.1 爆炸粒子效果

## 10.1 粒子发射器是什么

AndEngine 的粒子子系统提供了创建与运行粒子发射器的环境。发射器以某种模式创建粒子, 并将其展示在屏幕的某个位置上。

试图将这些粒子作为精灵来创建的话, 对于通常的游戏引擎来说, 会导致太多的运行开销, AndEngine 也不例外。实际需要的是一个特殊的子系统, 含有用于绘制的贴图, 并且能根据时间来修改它, 模仿粒子的物理效果, 从而产生逼真的动画。

创建一个效果真实的粒子发射器, 是一门学问, 也是一门艺术。产生粒子效果所需的代码并不复杂, 然而要制作出非常逼真的粒子效果, 需要反复试验, 甚至某

个微小的更改都会产生完全不同的结果。更麻烦的是, Android 模拟器不能很好地展示粒子发射器所生成的动画, 因为模拟器毕竟没有利用宿主操作系统平台的硬件绘图机制, 而是通过软件方式以模拟的指令集来运行的。仅使用模拟器很难调试粒子生成器。

## 10.2 粒子系统如何运作

粒子系统的艺术源于其建模的本质。大家都知道火的样子, 然而真实的火都具有随机性。比如蜡烛的火焰, 通常由颜色较深的内焰以及稍浅的外焰组成, 其颜色由蓝色逐级过渡为亮黄色。火焰的热量导致蜡烛芯中的蜡颗粒开始融化并逐渐燃烧, 从而产生些许随机的火苗, 这些火苗又影响到周围其余已经燃烧的蜡颗粒所产生的火焰, 这就是通常看到的烛光。

当上述行为被建模为粒子发射器时, 有些因素是固定的, 而另一些则希望它随机发生。例如, 所有的蜡烛颗粒都是从烛芯离开的, 通常来说是向上移动, 但是它们离开烛芯的角度, 却需要是个随机值。而且, 蜡烛颗粒的颜色会随着时间而改变。刚开始离开烛芯时, 是灰暗无光的(实际绘制中使用透明色即可), 随后燃烧时发出蓝光, 在最亮时呈现亮黄色, 然后变暗为红色, 最后化为黑烟。并不是所有的颗粒都以一模一样的方式变色, 所以在变颜色这个行为上, 也需要随机性。最后, 在颗粒移动的路径上, 再一次需要随机性, 因为受热越多的颗粒运动得越快。

粒子系统可以产生成千上万的颗粒。构建粒子时可指定一些基本参数, 并在其上运用变化效果以产生随机性。这些粒子也可以对重力做出反应, 进行重力或反重力加速运动, 它们也可具有相关的生命期、速度、方向、加速度等属性, 其大小、颜色、旋转方式等属性也可以改变。此外, 开发者还可以选择或替换粒子绘制到背景上时所用的渲染方法。

## 10.3 AndEngine 的粒子系统

AndEngine 的粒子系统是由如下组件所构成的体系:

- ParticleSystem (粒子系统): 该组件包含整个粒子系统, 包括 ParticleEmitter、一个小的 OpenGL 贴图以及一些用于控制粒子生成率与生成质量的参数。
- ParticleEmitter (粒子发射器): 该组件产生并绘制粒子, 而且会在其生命期中按照需要修改粒子的属性。ParticleEmitter 的类型 (Circle、CircleOutline、Point、Rectangle、RectangleOutline) 定义了粒子生成区域的形状。对于每个粒子来说, 它都会在 ParticleInitializer 所限定的范围内随机地初始化其位置、速度、加速

度、旋转角度、重力加速度、颜色及透明度等属性。

- Particle (粒子): Particle 是由 ParticleEmitter 所产生并绘制的单个粒子元素。某个效果所产生的所有 Particle 对象都将共享同一个 OpenGL 贴图, 该贴图可以是任意内容, 但通常都很小。同时, 每个 Particle 都有自己的生命期和参数集, 如颜色、透明度、旋转角度和缩放, 这些参数在生命期内可用 ParticleModifier 修改。
- ParticleInitializer (粒子初始化器): 该组件用于设定某效果所产生粒子的初始属性。它以最小值和最大值的方式限定若干属性的取值区间, 包括速度、加速度、旋转角度、重力加速度、颜色以及透明度。ParticleInitializer 会在生成 Particle 对象时在给定范围内随机选取数值填入。
- ParticleModifier (粒子修改器): 该组件将告知 ParticleSystem 如何在粒子生命期内改变其属性。很多属性都可以使用 Modifier 对象进行修改, 包括生命期、颜色、透明度、旋转角度、缩放。Modifier 定义了修改效果的起止时间, 以及最小值与最大值。在每次更新时, ParticleEmitter 将调整每个 Particle 对象, 使其符合 ParticleModifier 所指定的变更方式。数值的变换是渐进式的, 也就是说, 如果某粒子的 ColorModifier 效果在第 0 秒开始, 第 1 秒结束, 并将粒子的颜色由红变蓝, 那么在这 1 秒钟内, 所有 Particle 对象将会各自逐渐地改变其颜色, 而不是瞬间完成颜色更改。

### 10.3.1 ParticleSystem 类

ParticleSystem 类只有一个构造器, 需要经由它传入所需参数:

```
ParticleSystem(final IParticleEmitter pParticleEmitter, final float pMinRate,
                final float pMaxRate, final int pMaxParticles,
                final TextureRegion pTextureRegion)
```

该方法的参数都很直白。rate 参数用于指定每秒所生成的粒子数量, ParticleSystem 会在指定的范围内随机选取数值并以该频率生成粒子。pTextureRegion 所指定的贴图将会用于所有该系统生成的粒子之上。很显然, 在调用本构造器之前, 需要先行将 TextureRegion 对象载入。载入它的方式和之前创建 Sprite 与其他 AndEngine 组件时所使用的是一样的。

AndEngine 本身没有附带粒子贴图, 不过 AndEngineExample 范例程序中有很多贴图, 同时本书游戏项目的 assets/gfx/particles 目录下也加入了许多贴图。对于大多数效果来说, 会使用具有某种颜色及透明度的特定粒子形状, 在其基础上进行修改, 以创建所需的效果粒子, 不过也可以随意使用一张小的贴图。正如本章最后习题所讲的那样, 开发者可以轻松地创建大量使用特效的 Android 程序。

### 10.3.2 ParticleEmitter 类

如果要创建 ParticleSystem 对象, 则需要有 ParticleEmitter 对象。根据效果所需的图形, 开发者可以选择使用不同类型的 ParticleEmitter。发射器将会在其图形范围内的随机位置生成粒子。AndEngine 使用 Java 的 Random 工具类进行随机计算, 并且会在发射器的图形范围内均匀地分布粒子。

#### CircleParticleEmitter 类

该类本来应该被命名为 EllipseParticleEmitter 的, 不过这不影响使用。此种发射器用来在屏幕上的椭圆或圆形区域内发射粒子。

```
CircleParticleEmitter(final float pCenterX, final float pCenterY,
    final float pRadius)
CircleParticleEmitter(final float pCenterX, final float pCenterY,
    final float pRadiusX, final float pRadiusY)
```

第一个构造器使用圆心位于 (pCenterX, pCenterY) 的圆形作为发射区域, 第二个使用椭圆。粒子将从圆或椭圆内部的任何位置发射出来。椭圆可以通过 RotationInitializer 进行旋转。

#### CircleOutlineParticleEmitter 类

此发射器也是从圆形或椭圆区域中发射粒子的, 不过发射位置仅限于图形的边缘。

```
CircleOutlineParticleEmitter(final float pCenterX, final float pCenterY,
    final float pRadius)
CircleOutlineParticleEmitter(final float pCenterX, final float pCenterY,
    final float pRadius)
```

参数的意义与 CircleParticleEmitter 类的构造器相同。

#### PointParticleEmitter 类

此发射器将从某个单一的点发射粒子, 其构造器如下:

```
PointParticleEmitter(final float pCenterX, final float pCenterY)
```

#### RectangleParticleEmitter 类

此发射器的发射区域是矩形, 很明显, 当长或宽为 0 时, 就变成了一条线段。

```
RectangleParticleEmitter(final float pCenterX, final float pCenterY,
    final float pWidth, final float pHeight)
```

像 CircleParticleEmitter 一样, 其初始角度可以通过 RotationInitializer 来调整。

### RectangleOutlineEmitter 类

像 CircleOutlineParticleEmitter 一样, 此发射器在矩形的边缘生成粒子, 如果其形状是一条线段的话 (pWidth 或 pHeight 等于 0), 将会和 RectangleParticleEmitter 等效。其构造器如下:

```
RectangleOutlineParticleEmitter(final float pCenterX, final float pCenterY,  
    final float pWidth, final float pHeight)
```

### 10.3.3 ParticleInitializer 类

在创建 ParticleSystem 时, 可使用 ParticleInitializer 对象来控制粒子发射器的初始设定。有多种属性可通过初始化器来控制, 包括加速度、透明度、颜色、重力加速度、旋转角度、速度等。所有这些初始化器既可设定某属性的固定值, 也可设定其取值区间, 以便 ParticleEmitter 从中随机取值。

#### AccelerationInitializer 类

该类对象用于设定 ParticleSystem 所生成粒子的初始加速度:

```
AccelerationInitializer(final float pAcceleration)  
AccelerationInitializer(final float pAccelerationX, final float pAccelerationY)  
AccelerationInitializer(final float pMinAccelerationX,  
    final float pMaxAccelerationX, final float pMinAccelerationY,  
    final float pMaxAccelerationY)
```

前两个构造器使用固定值而非随机值来创建初始化器对象。第三个构造器更加灵活, 允许指定加速度值的范围。构造器中的 X 与 Y 方向都是相对于 ParticleEmitter 来说的。

#### AlphaInitializer 类

该类对象用于设定粒子的初始透明度:

```
AlphaInitializer(final float pAlpha)  
AlphaInitializer(final float pMinAlpha, final float pMaxAlpha)
```

与其他初始化器一样, 也可以选择用固定值还是随机数取值范围来构造对象。

#### ColorInitializer 类

此类对象用于设定粒子的初始颜色:

```
ColorInitializer(final float pRed, final float pGreen, final float pBlue)  
ColorInitializer(final float pMinRed, final float pMaxRed, final float pMinGreen,  
    final float pMaxGreen, final float pMinBlue, final float pMaxBlue)
```



第一个构造器使用指定的色调初始化粒子的颜色，其最终颜色值是由指定的色调同粒子贴图的颜色值相乘后得到的。第二个构造器使用取值区间来初始化粒子颜色。

#### GravityInitializer 类

此类对象仅用于将粒子的重力效果打开，也就是将其加速度设定为地球表面的重力加速度。

#### RotationInitializer 类

该类对象设定粒子的初始旋转角度，取值可从 0.0 度至 360.0 度。

```
RotationInitializer(final float pRotation)
```

```
RotationInitializer(final float pMinRotation, final float pMaxRotation)
```

#### VelocityInitializer 类

该类对象用于设定粒子的初始速度：

```
VelocityInitializer(final float pVelocity)
```

```
VelocityInitializer(final float pVelocityX, final float pVelocityY)
```

```
VelocityInitializer(final float pMinVelocityX, final float pMaxVelocityX, final float  
pMinVelocityY, final float pMaxVelocityY)
```

VelocityInitializer 与 AccelerationInitializer 非常相似，只不过这里设定的是初始速度而非加速度。其 X 与 Y 方向也是相对于 ParticleEmitter 来说的。

### 10.3.4 ParticleModifier 类

既然已经初始化了 ParticleSystem 对象，接下来，就需要告知它如何在粒子的生命期内修改其属性了。此事是由 ParticleModifier 对象来完成的。不巧的是，这一系列修改器的类名和用于 Entity 的修改器类名称刚好是一样的，尽管这不是大问题，而且便于记忆，但是当在某个类中需要同时引用 Entity 修改器与 Particle 修改器的同名类时，就需要使用完全修饰的类名了。这一点将在本章稍后的 V3 范例游戏代码中看到。

AndEngine 提供了用于修改粒子的透明度、颜色、生命期、旋转角度、缩放因子等属性的修改器。大部分的修改器都包含有起止时间，这两个时间都是相对于粒子被创建的时间来说的。

#### AlphaModifier 类

此类修改器将在指定的粒子生命期内将其透明度由 pFromAlpha 修改为 pToAlpha：

```
AlphaModifier(final float pFromAlpha, final float pToAlpha,  
final float pFromTime, final float pToTime)
```



### ColorModifier 类

此类修改器会在指定时间段内将指定的色调应用于粒子的贴图之上：

```
ColorModifier(final float pFromRed, final float pToRed, final float pFromGreen,
               final float pToGreen, final float pFromBlue, final float pToBlue,
               final float pFromTime, final float pToTime)
```

### ExpireModifier 类

该修改器定义了粒子的生命期。这是唯一不需要指定起止时间的修改器。

```
ExpireModifier(final float pLifeTime)
ExpireModifier(final float pMinLifeTime, final float pMaxLifeTime)
```

第一个构造器所创建的修改器会使所有粒子具有相同的生命期，第二个则提供了生命期的范围，每个粒子的生命期将是此范围内的随机值。

### RotationModifier 类

此类对象会在给定的时间段内修改粒子贴图的旋转角度：

```
RotationModifier(final float pFromRotation, final float pToRotation,
                  final float pFromTime, final float pToTime)
```

### ScaleModifier 类

此类对象将在给定的时间段内逐渐改变粒子的缩放倍数。

```
ScaleModifier(final float pFromScale, final float pToScale,
               final float pFromTime, final float pToTime)
ScaleModifier(final float pFromScaleX, final float pToScaleX,
               final float pFromScaleY, final float pToScaleY, final float pFromTime,
               final float pToTime)
```

第一个构造器所创建的修改器将会统一修改 X 与 Y 方向的缩放倍数，即不会改变纵横比，第二个则可以分别修改两个方向上的缩放倍数。

## 10.3.5 有用的 ParticleSystem 类方法

ParticleSystem 类包含一些有用的方法，可以在代码中使用它们来产生某些粒子效果。开发者可以启动或停止某个 ParticleSystem，也可以设置它与其他贴图混合渲染的方式。

### 启动与停止粒子效果

ParticleSystem 类包含用于启动或停止粒子生成的方法：

```
void setParticlesSpawnEnabled(final boolean pParticlesSpawnEnabled)
```

将参数设定为 true 即开始生成粒子, false 即停止生成。也可以用下列方法测试当前是否正在生成粒子:

```
boolean isParticlesSpawnEnabled()
```

### 设定 OpenGL 渲染方式的方法

前面讲 Sprite 类时曾经接触过此方法。其中 OpenGL 的部分仍然超出了本书讨论的范围, 读者只需知道该方法设定粒子贴图同屏幕上其他贴图的混合渲染方式即可:

```
void setBlendFunction(final int pSourceBlendFunction,  
                      final int pDestinationBlendFunction)
```

该方法可取的参数值定义在下列文件中:

```
javax.microedition.khronos.opengles.GL10. ⊖
```

## 10.4 创建粒子系统

写作本书时, AndEngine 要求游戏开发者必须使用刚才介绍的初始化器与修改器从头开始创建 ParticleSystem 系统。然而, 这种方法存在如下一些问题:

- ❑ 通过组合贴图、初始化器、修改器等对象来创建粒子效果, 显得很直观。
- ❑ 很难在不同的项目中复用粒子效果, 因为必须通过复制与粘贴的方式来使用这些未独立打包的代码。
- ❑ 粒子效果的制作很枯燥: 即使做微小的参数改动, 也需要重新编译项目。

针对以上某些问题, 笔者的解决办法是通过导入 XML 格式的文件来制作 AndEngine 中的粒子效果。在读者读到这里时, 也许这项功能已经作为扩展包集成入 AndEngine 之中了。这些 XML 扩展文件都以 .px 为扩展名, 所需的代码都已包含在本书的范例代码中。在为 V3 游戏加入粒子效果时, 将会看到这两种创建粒子效果的方式, 读者可选择自己所喜好的那一种。

### 10.4.1 以传统方式创建粒子系统

以传统方式创建 AndEngine 的粒子系统需遵循如下步骤:

1. 创建包含粒子系统所需 Texture 对象的 TextureRegion 对象, 并使用 TextureManager 将其载入。此过程与其他创建 TextureRegion 对象的过程相同。
2. 根据所需形状, 调用对应发射器类的构造器, 以创建 ParticleEmitter 对象。
3. 创建 ParticleSystem 对象, 通过构造器传入 ParticleEmitter 对象、Texture 对象、

<sup>⊖</sup> 该接口文档可在此网址查阅: <http://developer.android.com/reference/javax/microedition/khronos/egl/EGL10.html>。——译者注

最小生成频率、最大生成频率以及最大粒子数。

4. 选择性地为 ParticleSystem 对象设定 OpenGL 渲染方式。
5. 为 ParticleSystem 对象增加初始化器。
6. 为 ParticleSystem 对象增加修改器。
7. 将 ParticleSystem 对象作为子对象加入当前 Scene 对象中。

### 10.4.2 以 XML 文件创建粒子系统

使用 XML 文件创建粒子系统的步骤原则上与传统方式相同，但是 PX 相关类会替开发者完成其中的许多步骤。下述过程与先前讲到的载入 TMX 瓦片地图的过程相似：

1. 创建 PXLoader 对象，下一节将会讲述创建该类。
2. 使用 PXLoader 从描述粒子系统的 PX 文件中创建 ParticleSystem 对象。
3. 选择性地为 ParticleSystem 对象设定 OpenGL 渲染方式。
4. 将 ParticleSystem 对象作为子对象加入 Scene 对象中。

本书不会讲解所有 PXLoader 及其相关类的代码，完整的源代码包含在可供下载的本章代码包中，也许读者阅读本书时，此部分代码已纳入 AndEngineExtensions 扩展包中了。这里需要讲讲制作游戏时会用到的接口。

#### PX 文件

用于描述粒子系统的 XML 文件相当简明，每个组件，诸如初始化器、修改器等，均以 XML 元素表示。完整的 XML 元素列表可在范例代码的 PXConstants.java 文件里查找。程序清单 10.1 中所示的 PX 范例文件也会在 V3 游戏中用到。

程序清单 10.1 PX 文件 explo.px

```
<ParticleConfig>
  <emitter
    shape="circle"
    center_x="0.0"
    center_y="0.0"
    radius_x="40.0"
    radius_y="40.0">
  </emitter>
  <system
    texture="particle_fire.png"
    min_rate="100"
    max_rate="100"
    max_particles="500">
    <init_color
      min_red="1"
      max_red="1"
      min_green="0"
      max_green="0"
```

```

    min_blue="0"
    max_blue="0">
</init_color>
<init_alpha
    min_alpha="0"
    max_alpha="0">
</init_alpha>
<init_velocity
    min_velocity_x="-2"
    max_velocity_x="2"
    min_velocity_y="-2"
    max_velocity_y="2">
</init_velocity>
<init_rotation
    min_rotation="0.0"
    max_rotation="360.0">
</init_rotation>
<mod_scale
    from_scale_x="1.0"
    to_scale_x="2.0"
    from_scale_y="1.0"
    to_scale_y="2.0"
    from_time="0"
    to_time="5">
</mod_scale>
<mod_color
    from_red="1"
    to_red="1"
    from_green="0"
    to_green="0.5"
    from_blue="0"
    to_blue="0"
    from_time="0"
    to_time="2">
</mod_color>
<mod_color
    from_red="1"
    to_red="1"
    from_green="0.5"
    to_green="1"
    from_blue="0"
    to_blue="1"
    from_time="2"
    to_time="4">
</mod_color>
<mod_alpha
    from_alpha="0"
    to_alpha="1"
    from_time="0"
    to_time="1">
</mod_alpha>

```



```

<mod_alpha
  from_alpha="1.0"
  to_alpha="0"
  from_time="3"
  to_time="4">
</mod_alpha>
<mod_expire
  min_lifetime="2"
  max_lifetime="4">
</mod_expire>
</system>
</ParticleConfig>

```

文件内容非常简单，不需要参考别的资料就可以读懂。读者甚至不用浏览 PXConstants.java 文件就能很容易地猜到其他元素的名字。

### PX 文件编辑器：PXEditor

在可供下载的代码中，包含了一个编辑器程序，开发者可以用它来很容易地编辑 PX 文件。在将来（也许是读者阅读本书时），编辑器将会增加粒子效果查看器，开发者可通过它在编辑后立即查看实际的粒子效果。开发者现在可使用这一版本的编辑器来创建与编辑粒子系统，从而不再需要用传统方式通过手工编码去创建初始化器与修改器。

图 10.2 是 PXEditor 软件截图，可以看到，该软件非常简单易用。开发者可通过“File”菜单来创建新的 PX 文件，也可载入并编辑现有的文件。各种微调按钮可用于调整参数的值，界面中显示的图像即是粒子系统所使用的贴图。

### ParticleSystem

在本章的下载代码中，含有粒子效果查看程序，它会加载并显示 PX 文件所描述的粒子。如果需要查看其他 PX 文件的效果，则需修改 ParticlePlayer.java 文件中的一行。该行位于 onLoadScene() 方法中：

```
String pxFileName = "gfx/particles/explo.px";
```

将 PX 文件放在项目的 assets 文件夹下并编辑该字符串，这样 ParticlePlayer 就能找到它了。粒子所用的贴图图像需要导入项目的 assets/gfx 文件夹中。

运行 ParticlePlayer 之后，触摸屏幕上任意一点，粒子效果就会在触摸点上显示出来。如果在加载或显示的过程中发生错误，可在 LogCat 控制台中找到相关的错误信息。如果 PX 文件格式错误，LogCat 会指出包含错误的那一行文字，以及解析错误的类型。

图 10.3 是 explo.px 文件所定义的粒子系统在不同阶段的效果截图。



图 10.2 PXEditor

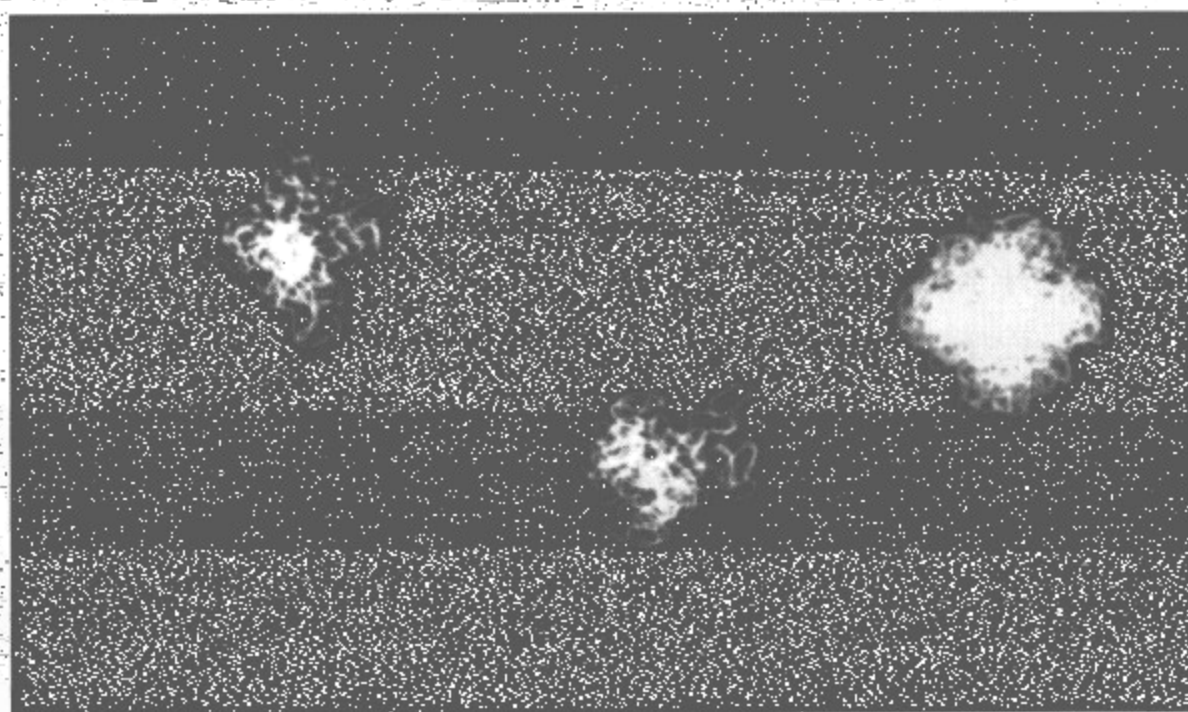


图 10.3 ParticlePlayer 程序正在展示 explo.px 所定义的粒子系统



## PXLoader

PXLoader 类对象知道如何加载描述粒子系统的 .px 文件，其创建与创建加载瓦片地图的 TMXLoader 对象相似：

```
PXLoader(final Context pContext, final TextureManager pTextureManager)
PXLoader(final Context pContext, final TextureManager pTextureManager,
        final TextureOptions pTextureOptions)
```

如果不指定 TextureOptions 参数的话，将会使用 TextureOptions.DEFAULT 创建 PXLoader 对象。

该类包含两个可用于载入粒子系统定义文件的方法：

```
ParticleSystem createFromAsset(final Context pContext,
                               final String pAssetPath)
ParticleSystem load(final InputStream pInputStream)
```

第一个方法通常用于从 assets 文件夹的子目录中载入定义文件以创建粒子系统。第二个方法可以从 InputStream 输入流对象中载入粒子系统。两种情况都会返回完整的 ParticleSystem 对象。如果在加载定义文件的过程中发生错误，将会抛出 PXLoadException 或 PXParseException 异常。

## 10.5 将粒子发射器加入 V3 游戏中

让我们把粒子发射器运用到 V3 游戏中吧！这样可以让游戏变得生动起来。尽管现在并未加入碰撞检测，不过所有的吸血鬼都会走向屏幕左侧 Bliss 女士所在的学校。现在的墓地场景将加入新的效果：当玩家触摸某个吸血鬼时，它将化为一团火焰并从屏幕上消失（Buffy<sup>⊖</sup>，干得好！）。

### 10.5.1 以传统方式制作 V3 的爆炸效果

本章将使用传统方式与 XML 文件这两种方式来创建粒子系统并制作粒子效果。首先来看传统方式，程序清单 10.2 中的 Level1Activity.java 是由第 9 章之中的代码修改而成的。

程序清单 10.2 以传统方式制作粒子效果的 Level1Activity.java 代码

```
package com.pearson.lagp.v3;
...
public class Level1Activity extends BaseGameActivity {
```

⊖ Buffy 是美国电视系列剧《Buffy the Vampire Slayer》（中文名《吸血鬼猎人巴菲》或《魔法奇兵》），讲述了一个被命运选中的女孩巴菲与吸血鬼等邪恶势力对抗的故事。——译者注

```

private TextureRegion mParticleTextureRegion;
private ParticleSystem particleSystem;
private CircleParticleEmitter particleEmitter;

@Override
public void onLoadResources() {
    /* 加载贴图。 */

    TextureRegionFactory.setAssetBasePath("gfx/particles/");
    mParticleTexture = new Texture(32, 32,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mParticleTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mParticleTexture, this,
            "particle_fire.png",
            0, 0);
    mEngine.getTextureManager().loadTexture(
        this.mParticleTexture);
}

@Override
public Scene onLoadScene() {

    particleEmitter = new CircleParticleEmitter(
        CAMERA_WIDTH * 0.5f, CAMERA_HEIGHT * 0.5f + 20, 40);
    particleSystem = new ParticleSystem(particleEmitter,
        100, 100, 500, this.mParticleTextureRegion);

    particleSystem.addParticleInitializer(
        new ColorInitializer(1, 0, 0));
    particleSystem.addParticleInitializer(
        new AlphaInitializer(0));
    particleSystem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE);
    particleSystem.addParticleInitializer(
        new VelocityInitializer(-2, 2, -2, -2));
    particleSystem.addParticleInitializer(
        new RotationInitializer(0.0f, 360.0f));

    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-
        ScaleModifier(1.0f, 2.0f, 0, 5));
    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-
        .ColorModifier(1, 1, 0, 0.5f, 0, 0, 0, 3));
    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-
        .ColorModifier(1, 1, 0.5f, 1, 0, 1, 2, 4));
    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-

```

```

AlphaModifier(0, 1, 0, 1));
particleSystem.addParticleModifier(
    new org.anddev.andengine.entity.particle.modifier-
        .AlphaModifier(1, 0, 3, 4));
particleSystem.addParticleModifier(
    new ExpireModifier(2, 4));

particleSystem.setParticlesSpawnEnabled(false);
scene.getLastChild().attachChild(particleSystem);

return scene;
}

@Override
public void onLoadComplete() {

private Runnable mStartVamp = new Runnable() {
    public void run() {
        int i = nVamp++;
        Scene scene = Level1Activity.this.mEngine.getScene();
        float startY = gen.nextFloat() * (CAMERA_HEIGHT - 50.0f);
        asprVamp[i] = new AnimatedSprite(CAMERA_WIDTH - 30.0f,
            startY, mScrumTextureRegion.clone()) {
            @Override
            public boolean onAreaTouched(
                final TouchEvent pAreaTouchEvent,
                final float pTouchAreaLocalX,
                final float pTouchAreaLocalY) {
                switch(pAreaTouchEvent.getAction()) {
                    case TouchEvent.ACTION_DOWN:
                        /* 吸血鬼消失了吗? */
                        for (int j=0; j<nVamp; j++){
                            if (
                                (Math.abs(asprVamp[j].getX() + (asprVamp[j].getWidth()/2) -
                                    pAreaTouchEvent.getX()) < 10.0f) &&
                                    (Math.abs(asprVamp[j].getY() +
                                        (asprVamp[j].getHeight()/2) - pAreaTouchEvent.getY()) < 10.0f)) {
                                    particleEmitter.setCenter(
                                        pAreaTouchEvent.getX(),
                                        pAreaTouchEvent.getY());
                                    particleSystem.setParticlesSpawnEnabled(
                                        true);
                                    mHandler.postDelayed(mEndPESpawn, 3000);
                                    asprVamp[j].clearEntityModifiers();
                                    asprVamp[j].registerEntityModifier(
                                        new AlphaModifier(1.0f, 1.0f, 0.0f));
                                    asprVamp[j].setPosition(
                                        CAMERA_WIDTH,
                                        gen.nextFloat() *
                                        CAMERA_HEIGHT);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    break;
}
return true;
}
};
scene.registerTouchArea(asprVamp[i]);
...
private Runnable mEndPESpawn = new Runnable() {
    public void run() {
        particleSystem.setParticlesSpawnEnabled(false);
    }
};
}

```

现在来依次查看方法代码，以了解如何用传统方式显示粒子效果：

- ❑ 在 `onLoadResources()` 方法中，加载粒子系统所使用的 `Texture` 对象。所需图像文件已经预先放入 `assets/gfx/particles` 文件夹下了，这么做的理由是游戏中有多个地方都需要创建粒子系统，所以要将其贴图资源统一管理。
- ❑ 在 `onLoadScene()` 方法中，以 `CircleParticleEmitter` 对象为构造器参数，创建了 `ParticleSystem` 对象，同时也传入了合适的粒子数量以及生成频率（所谓“合适的”值一般是要通过反复实验得出的）。
- ❑ 为 `ParticleSystem` 对象增加“合适的”初始化器与修改器，以产生期望的效果。现在需要调用 `setSpawnEnabled(false)` 方法，因为在吸血鬼未被触摸时，不需要生成粒子。接下来，将粒子系统对象加入 `Scene` 对象中。
- ❑ 在 `mStartVamp` 这个实例所用的 `Runnable` 匿名内部类中，通过在 `AnimatedSprite` 匿名内部类中覆写 `onAreaTouched()` 方法来捕捉触摸事件。该方法会在精灵被触摸时回调，然而它并未告知究竟是哪个吸血鬼被触摸到了，所以需要在已经生成的吸血鬼中迭代式地查询。这种做法现在还能勉强应付过去，因为任何时候屏幕上的吸血鬼都不会超过 10 个。然而如果是一个可能有成百上千个活动精灵的游戏，那么当然需要使用一种更加成熟的策略来进行碰撞检测了（本书第 12 章将讨论这种办法）。程序清单 10.2 的格式有点儿乱，不过真正的源码文件要比这里显示的代码格式好很多。
- ❑ 如果触摸点距吸血鬼的中心点小于 10 个像素，那么就判定为玩家击中吸血鬼了。
- ❑ 当侦测到吸血鬼被击中时，将粒子生成器放置于其上，并打开粒子生成开关以产生爆炸效果。代码将爆炸效果放在了击打点上，其实也可以将它放在吸血鬼的坐标上，不过将火焰放在击打点上显得更加合适。
- ❑ 代码请求 `Android` 系统在 3 秒后执行另一个用于停止爆炸效果的 `Runnable` 任务。

- 将现有的修改器从吸血鬼精灵对象上移除，这样可以使精灵不会再改变位置了，从而避免了 CPU 周期的浪费。接下来在精灵上运用透明度修改器，使其在 1 秒钟内消失。
- 在方法的最后返回 true，以便告知 Android 系统，不要再继续通知其他吸血鬼精灵了。
- mEndPESpawn() 任务将会停止粒子效果，关闭粒子的生成。

### 10.5.2 以 XML 文件方式制作 V3 的爆炸效果

在下载代码中的 V310PX 项目中，包含有修改后的版本，它可以用来生成与上例相同的粒子效果。此版本使用了等效的代码来替换传统方式中用于生成粒子系统的那部分代码。该等效代码使用了程序清单 10.1 中所示的 explo.px 文件来创建粒子系统。这些修改都在程序清单 10.3 中做了标注。

程序清单 10.3 以 XML 文件方式制作粒子效果的 Level1Activity.java 代码

---

```
package com.pearson.lagp.v3;
...
public class Level1Activity extends BaseGameActivity {
...
    private ParticleSystem particleSystem;
    private BaseParticleEmitter particleEmitter;
...
    @Override
    public Scene onLoadScene() {
...
        try {
            final PXLoader pxLoader = new PXLoader(this,
                this.mEngine.getTextureManager(),
                TextureOptions.BILINEAR_PREMULTIPLYALPHA);
            particleSystem = pxLoader.createFromAsset(this,
                "gfx/particles/explo.px");
        } catch (final PXLoadException pxle) {
            Debug.e(pxle);
        }
        particleSystem.setBlendFunction(GL10.GL_SRC_ALPHA,
            GL10.GL_ONE);
        particleSystem.setParticlesSpawnEnabled(false);
        particleEmitter =
            (BaseParticleEmitter) particleSystem.getParticleEmitter();

        scene.getLastChild().attachChild(particleSystem);
...
    }
```

---

来看看此方式同传统方式的区别：

- 在这种方式里，不需要声明 Texture 或 TextureRegion 对象，因为该任务已由 PXLoader 完成了。开发者只需要将 PX 文件中指明的贴图图像导入项目相应位置即可，PXLoader 会找到它的。比如 explo.px 文件中提到的 particle\_fire.png 文件就应该放在现有的 Texture 资源路径下（在本 Activity 环境中，所有的 Texture 对象都将以 gfx/Level1 作为其资源路径）。该类的 particleEmitter 字段被声明为 BaseParticleEmitter 类型，因为代码并不知道具体的发射器类型。
- 不再需要在 onLoadResources() 方法中加载 TextureRegion 与 Texture 对象了。
- 在 onLoadScene() 方法中创建了 PXLoader 对象，并调用其 createFromAsset() 方法从 explo.px 文件中创建粒子系统。该文件先前已被导入项目的 assets/gfx/particles 目录下。接下来调用 getParticleEmitter() 方法，获取粒子系统的发射器对象，因为稍后需要重新设置其位置。
- 其余部分都是相同的。精灵的触摸检测逻辑代码与传统版本一样。用 PXLoader 创建的粒子系统与其他 ParticleSystem 对象没有差别，因此开发者也可以在其上加入修改器，或者重新设置发射器的位置，这与传统方式下创建的粒子系统所具有的功能是一致的。

## 10.6 总结

粒子效果可以让游戏更加有趣。通过制作复杂、逼真的特效，可以使游戏更加好玩。AndEngine 所提供的粒子系统不会耗费太多资源，然而如果用重复的动画精灵来做的话，则需占用大量的硬件资源。

在游戏中加入粒子效果很简单，而且粒子效果的制作也不复杂。将粒子系统加入游戏中至少可用如下三种方式：

1. 使用 AndEngine 所提供的初始化器与修改器对象编码创建粒子系统对象。
2. 创建描述粒子系统的 PX 文件，并在程序运行时使用 PXLoader 对象将其载入。
3. 使用他人已制作并调试好的粒子系统。可通过复制并粘贴代码或者用 PX 文件将其导入项目中。

不论选择用哪种方法创建粒子系统，都可以根据游戏中发生的事件来设置粒子效果的位置并激活粒子的生成。在第 11 章中，将会讲解如何通过加入声音而使特效更具吸引力。

## 10.7 习题

1. 使用程序清单 10.2 中所示的传统方式来创建粒子特效，修改 particleSystem 对



象，使粒子效果看起来像是一团烟雾。使用下载代码中位于 `assets/px` 目录下的 `particle_smoke.png` 文件作为贴图图像，并选择适当的粒子颜色。调整粒子系统的效果，直到看起来很逼真为止。

2. 使用程序清单 10.3 中所示方式，通过 `PXLoader` 来构建同习题 1 一样的粒子效果。可以自己写 `smoke.px` 文件的内容，也可用 `PXEditor` 软件来创建它，不管使用哪种方法，以 `explo.px` 为基础来做，都是最简单的。

3. 创建全新的粒子效果，用于演示下雨的场景。使用 `PXLoader` 来创建粒子系统，并用 `ParticlePlayer` 程序来展示其效果。





## 第 11 章 声 音

- 11.1 如何在游戏使用声音
- 11.2 音乐与音效的来源
- 11.3 音乐与音效制作工具
- 11.4 音频解码器
- 11.5 使用 AndEngine 播放声音
- 11.6 将声音加入 V3 游戏
- 11.7 总结
- 11.8 习题

声音可以烘托游戏的气氛,使得其中的动作栩栩如生,它提供了一条其他手段所无法达成的情感纽带。自从以《The Jazz Singer》<sup>①</sup>为标志的有声电影诞生后,电影制作人便意识到了这一点,他们投入巨资制作符合电影情节的背景音乐及音效。读者也许没有如此大的财力,也没有 John Williams<sup>②</sup>那般的音乐天赋,但是声音对您的游戏来说,依然很重要。AndEngine 所提供的 API 是构建在 Android 系统原生的多媒体硬件解码能力之上的,这使得开发者可以很容易地在游戏中加入声音功能。

## 11.1 如何在游戏中使用声音

电子游戏中的声音制作是一门完全独立的职业,甚至有专门的贸易会议(例如 <http://www.gamesoundcon.com/>)。虽说看完下面几段之后,读者不可能变成游戏声音方面的专家,但您仍然可以学到制作游戏声音的一些基本原则。

游戏中的声音可分为两种基本类型:

□ 音乐 (Music), 通常指背景音乐。

□ 音效 (Sound Effect), 它伴随着游戏中发生的即时事件而播放。

### 11.1.1 音乐

音乐对游戏氛围的设定是很重要的。忧郁的音乐会让玩家觉得压抑,而能够使人回想起旋转木马、手摇风琴等快乐时光的音乐,则会令玩家愉悦。音乐的速度通常与当前的游戏节奏相符。对于一个难度随着关卡逐级增加的游戏来说,可以随着挑战的困难度提高与敌人的进攻速度加快而增加音乐的速度。

似乎没有一条固定的规则来限定音乐与其引发的特定情绪之间的关系。“欢快”的音乐多用大调,而小调则多用于“悲伤”的音乐,不过这种规则也有例外。最好的建议就是:如果某段音乐玩家听了有身临其境之感,那就是好的游戏音乐。为游戏选取背景音乐时,要多听其他游戏的音乐以及通常的音乐,试着选出那些百听不厌的曲子。

在玩手机游戏时,越来越多的玩家选择不听游戏中的配乐,而是播放自己设备中存放的音乐。Android 手机可以在运行其他程序的同时播放本机音乐,玩游戏时也是如此,所以在这一点上不需要再做处理。

### 11.1.2 音效

音乐通常在游戏背景中是持续播放的,而音效则是随着游戏中的突发事件而播放的

① 中文名《爵士歌手》、《爵士歌王》,是1927年拍摄上映的一部美国歌舞片。它是第一部全片使用同步对话的电影,标志着商业性有声电影的出现和无声电影的结束。——译者注

② 约翰·威廉姆斯,好莱坞知名作曲家、钢琴家、指挥家、电影配乐家。——译者注

短暂声音。如果枪响了，玩家会听到“嘣”的一声。如果敌人碰到了某物体，也应该会听到撞击声，也许还可能有一声“啊”的惨叫。如果吸血会被火焰烧着了，玩家则会听到一阵猛烈的爆炸声。

用于创建和编辑音效的工具也能够用于音乐，但其关注点不同。音效通常很短，更加重视声音的裁剪与过滤，相比之下，音乐则更具完整性，更加强调作曲中音符与休止符的交替使用，以及用于演奏音乐的乐器音色。

## 11.2 音乐与音效的来源

制作专用机与 PC 游戏的公司要花费数以万计的资金去制作独特且动听的游戏声音，其公司内部通常有专职的游戏音乐作曲人。我们也许没有这样的财力，不过仍然有办法制作出色的游戏声音。

尽管网上有大量的音乐与音效文件可供下载，不过仍然需要高度关注授权许可问题，这与制作图形与动画时需要考虑的版权问题一致。我们最不希望看到的就是在游戏发布之后，某律师会以侵犯其委托人的知识产权为由找上门来。

可以经由如下渠道获取音乐与音效资源：

- ☐ 自己制作音乐与音效。如果有作曲天赋以及乐器的话，就可以为游戏进行原创作曲了，这是迄今为止最好的选择。以此方式制作与演奏的音乐，您将拥有全部的权利。
- ☐ 使用朋友所做的音乐。如果有具备音乐天赋与相关乐器的朋友，则可请其为游戏制作原创背景音乐，这也是个不错的办法。务必在音乐的使用问题上与其达成清晰的协议，而且最好以书面形式将协议确定下来。
- ☐ 经授权后使用他人的音乐。如果“他人”指的是诸如 Rolling Stones<sup>①</sup> 这样的明星，这种方式的开销将会很大，不过很多网站都以合理的费用提供专业级音乐的使用授权，例如下列网站就提供此服务：
  - ☐ <http://www.partnersinrhyme.com>：包含音乐与音效
  - ☐ <http://www.5alarmmusic.com/>
  - ☐ <http://www.mymusicsource.com>
- ☐ 使用公有领域<sup>②</sup>的音乐与音效。V3 游戏中的背景音乐就是如此，游戏主界面所用

① 滚石乐队，20 世纪 60 年代成名的英国摇滚乐乐队，是音乐史上最杰出的乐队之一。——译者注

② Public Domain，中文译为“公有领域”、“公共领域”或“公众领域”，是人类的一部分作品与知识的总汇，包括文章、艺术品、音乐、科学理论、发明等等。领域内的知识财产，任何个人或团体都不具所有权益。这些知识发明属于公有文化遗产，任何人可以不受限制地使用和加工它们。——译者注

的是由 J.S. Bach<sup>Ⓐ</sup> 谱写的曲子。该段乐曲的 MIDI 文件可在此网站下载：<http://www.midiworld.com>。

## 11.3 音乐与音效制作工具

由于音乐与音效都是以音频文件的方式存放的，所以从这个角度来看，其编辑软件是一样的。声音文件的处理有多种方式，必须根据具体需要来选择最合适的工具。以下列出常用的处理方式。

- ❑ 如果要录制现场演奏的音乐，有很多专业级的工具软件可供选择，例如：Sony Sound Forge、M-Audio Pro Tools、Cubase 以及 Adobe Soundbooth 等。以上软件都不是免费的，但大都提供有廉价的“精简版”。
- ❑ 如果是对已有乐曲进行改编，则可以使用 MuseScore、MakeMusic Finale 等工具。这些工具可以进行小范围的音乐编辑，例如改变语音、调整速度，甚至是谱写音符。本章稍后将会举例说明 MuseScore 的使用方法。
- ❑ 如果是要编辑音效的话，像 Audacity 这样的音频编辑软件就能满足需求。这类编辑程序在音乐功能方面没有上述几款灵活，不过它们擅长录制与编辑原始音频文件。本章稍后也会有详细的范例来讲解如何使用 Audacity 录制音效。

## 11.4 音频解码器

Android 媒体播放器支持很多种类的媒体文件，Android 3.0 系统所支持的文件格式如下所列，每种类型后面均附有其作为游戏声音文件的优缺点。

- ❑ AAC：在这里列出它容易误导读者。截至 Android 3.0 版本，所有的 AAC 编码音频仅在嵌入 3GP 或 MP4 视频文件时才可以播放。所以目前还不能在 Android 游戏中使用此格式。
- ❑ MP3：一种传统的基于心理声学（psycho-acoustically）的音频编码格式，它对于音乐和音效都非常适用。由于编码器许可等原因，并非所有软件都支持编辑 MP3 文件。许多 Android 手机包括 MP3 硬件编码器，可以直接从文件导出 MP3，节省了宝贵的存储器资源和处理器周期。
- ❑ MIDI：与直接压缩声波不同，MIDI 文件记录下弹奏音乐所需的音符，以及一些元数据（metadata），例如弹奏这些音符所用的乐器。与其他格式相比，MIDI

---

<sup>Ⓐ</sup> 约翰·塞巴斯蒂安·巴赫（Johann Sebastian Bach, 1685–1750），巴洛克时期的德国作曲家，杰出的管风琴、小提琴、大键琴演奏家，被公认为是音乐史上最重要的作曲家之一。——译者注

文件占用空间很小，不过声音听起来有些机械化，不很自然。通过使用名为“gunshot”（射击）、“applause”（鼓掌）等的乐器，MIDI 确实可以有限度地制作一些音效。在 Android 手机上，MIDI 比 MP3、Ogg、WAV 等文件更加耗费 CPU 周期与内存，因为需要将其以文件流的方式读入，并通过内存中的波表来计算出需要播放的声音波形。

- ❑ Ogg Vorbis：这种编码格式与 MP3 格式相仿，但其具有开放式许可协议，并且被广泛支持。除了可以自由使用之外，它还有一些优于 MP3 之处。与 MP3 格式相同，大部分 Android 手机也都有专属的硬件用于读入并解码 Ogg 格式的音频文件。
- ❑ WAV：WAV 格式是完全数字化的音频文件。通常使用线性脉冲编码调制（linear pulse code modulation, LPCM）来进行声音的数字化，其数据通常是未压缩的。数字化的时候可以采用多种参数设定，而且 WAV 文件可以完全提供“CD 品质”的音乐，它也是商用音乐 CD 所采取的编码格式。因为其数据未压缩，所以这种编码格式的文件所占用的空间是迄今为止最大的。这一点在制作手机游戏时需要格外重视，尤其是制作背景音乐时。

## 11.5 使用 AndEngine 播放声音

AndEngine 使用“music”指代音乐，使用“sound”指代音效。一般来说，持续时间超过 5 秒的声音会被当做音乐，而少于 5 秒的则被当做音效，尤其是那些少于 3 秒的。

对于大多数游戏开发者来说，每个 Activity 程序仅需使用一种配乐主题，所以 AndEngine 用 Music 类的对象来表示音乐，并将其常驻内存，相比之下，音效是此起彼伏的，所以 AndEngine 使用 Android 系统的 SoundPool 类对象来管理这些短小的声音片段。令人高兴的是，作为游戏开发者的您不需要担心上述这些幕后发生的事情，因为 Nicolas 及其他 AndEngine 的开发者已经为我们做了非常周全的处理。

从开发者的角度看，需要知道如下 4 个类的使用方法：

1. Music：该类表示音乐。
2. Sound：该类表示音效。
3. MusicFactory：用于从文件中创建音乐对象的单例。
4. SoundFactory：用于加载音效的单例。

如果读者熟悉 Android 系统 MediaPlayer 类的 API，就会发现 AndEngine 的音乐与音效 API 同其很相似。AndEngine 在底层也是使用 MediaPlayer 的，所以这种相似非常合理。还存在一些相似的类，比如 MusicManager、SoundManager、SoundLibrary 等，



不过仅仅播放与控制音乐和音效是不需要用到它们的。

### 11.5.1 Music 类

通常不是用构造器创建该类对象，而是用 MusicFactory 来创建需要的 Music 对象（下一节将会讲具体做法）。有了 Music 对象之后，可以调用该类所提供的用于控制音乐播放的方法：

```
void play()
void stop()
void pause()
void resume()
void release()
```

这些方法的功能与其方法名一样<sup>⊖</sup>。如果在 Music 对象上调用 release() 方法，它的资源将被释放，不再能够播放音乐了。

```
void setLooping(final boolean pLooping)
void setVolume(final float pLeftVolume, final float pRightVolume)
void seekTo(final int pMilliseconds)
boolean isPlaying()
```

以上方法分别控制音乐播放的某个属性<sup>⊖</sup>。如果音乐处于暂停或停止状态，isPlaying() 将返回 false。

```
void setOnCompletionListener(final OnCompletionListener pOnCompletionListener)
```

通过该方法可以注册监听器，在音乐播放完毕时，监听器的回调方法会被调用。

### 11.5.2 Sound 类

正如预想的那样，该类所提供的方法与 Music 类非常相似。不过由于音效的短暂性，其与 Music 类仍有一些重要的区别：

- 没有 seekTo() 方法。
- 没有 isPlaying() 方法。
- 无法设置用于侦测音效结束播放的监听器。

另外也增加了一非常易用的方法，这得益于 Sound 类直接使用 Android 系统底层的 SoundPool 类来管理音效。

⊖ 中文意思分别是：播放、停止、暂停、继续、释放。——译者注

⊖ 中文意思分别是：设定循环次数、设定音量、跳转到指定位置播放。——译者注

```
void setLoopCount(final int pLoopCount)
```

该方法同 setLooping() 一起使用。如果使用 true 为参数调用 setLooping(), 音效将会持续播放, 直到调用 stop() 为止。如果使用 false 调用 setLooping(), 并且 setLoopCount 的参数值非 0, 音效则会重复播放指定的次数。有点别扭的是, Android 的循环次数是从 0 开始计算的, 所以如果需要播放 5 次音效, 则 pLoopCount 应该取值为 4。

```
void setRate(final float pRate)
```

可通过调用此方法改变播放速度。pRate 为 1.0f 代表使用正常速度播放, 允许的取值范围从 0.5f (以原来速度的一半播放) 到 2.0f。

### 11.5.3 MusicFactory 类

与其他用于创建媒体对象的工厂类一样, MusicFactory 也提供了一组形式为 createFrom... 的方法, 用于从不同类型的资源中加载音乐:

```
Music createMusicFromFile(final MusicManager pMusicManager,
    final Context pContext, final File pFile)
Music createMusicFromAsset(final MusicManager pMusicManager,
    final Context pContext, final String pAssetPath)
Music createMusicFromResource(final MusicManager pMusicManager, final
    Context pContext, final int pMusicResID)
```

这些方法与前文讲到的用于加载 Textures 对象、TMX 文件、PX 文件等所用到的工厂方法很相似, 不过据一些开发者反馈, createMusicFromResource() 方法存在一些问题。与 TextureFactory 等其他类相似, MusicFactory 类也提供了用于设置 assets 默认子目录的方法:

```
void setAssetBasePath(final String pAssetBasePath)
```

### 11.5.4 SoundFactory 类

SoundFactory 类也提供了一组类似的方法, 这些方法都利用了底层 SoundPool 的功能:

```
Sound createSoundFromPath(final SoundManager pSoundManager,
    final Context pContext, final String pPath)
Sound createSoundFromAsset(final SoundManager pSoundManager,
    final Context pContext, final String pAssetPath)
Sound createSoundFromResource(final SoundManager pSoundManager,
    final Context pContext, final int pSoundResID)
Sound createSoundFromFileDescriptor(final SoundManager pSoundManager,
    final FileDescriptor pFileDescriptor, final long pOffset,
    final long pLength)
```

虽说本书中的代码都是从 assets 文件夹中载入媒体资源的, 不过读者也应该了

解其他几种加载资源的方式，因为有时会需要从文件或 Resource 资源中加载它们。AndEngine 中的其他工厂类提供了与 Sound Factory 中的 `setAssetBasepath()` 方法相同的方法。

## 11.6 将声音加入 V3 游戏

现在需要制作 V3 游戏所需的音乐和音效，本节将通过范例来讲解如何制作这两种资源：

- 创建与实现游戏的背景音乐。需要在游戏运行时就开始播放背景音乐，直到正式切换到游戏关卡（例如 `Level1Activity`）为止。如果游戏暂停了，背景音乐也应停止，并在游戏继续时恢复播放。如果玩家在 Option 页面将音乐关闭，则需要停止播放音乐，直到玩家再次将本设置打开为止。音乐与音效的开关设定应当保存起来，即使关掉手机，此信息也不会丢失。
- 创建并实现游戏的音效，将其用于第 10 章制作的粒子效果中。当玩家通过点击屏幕而消灭吸血鬼时，应该伴随有一声表示胜利的音效，不过如果用户已在 Options 页面将音效关闭，那么就不应再播放音效了。音乐和音效应该可以分别打开和关闭。
- 为游戏中的武器增加音效。在第 10 章中加入的代码使得玩家可以将武器拖放至游戏场地，现在需要在玩家松开武器时做点事情，而且发生这些动作时需要有随之播放的音效（例如发射子弹或投掷斧头的声音）。

### 11.6.1 创建音效

为了给击杀吸血鬼的动作选配音效，笔者从前面讲到的 `PartenersInRhyme` 网站中选择了一份放置于公有领域的 WAV 文件。该站点既有公有领域的音效，也有一些发布于免版税（royalty-free）许可协议下的音效。这两种方式的差别在于，公有领域的音效在使用方面无任何限制，由免版税协议所授权的音效只需花少量的钱即可购得使用权，其协议规定了以后使用该音效将不再需要支付版税。

笔者选择了一份公有领域的文件 `Fireball3.wav` 作为音效，这个文件不是特别大（仅 18KB），播放时会听见一声巨响。不过现在需要将其稍加调整。首先打开 Audacity 软件，选择“File”菜单下的“Open”菜单项，将其导入。此时软件界面如图 11.1 所示。

这份声音剪辑的长度很短，不足 1 秒钟，故而非常适合作音效用，它的渐入渐出效果也做得很好。Audacity 软件可以做多种波形分析，作为演示，本书讲一下如何对其做声音频谱分析。选择“Analyze”菜单下的“Plot Spectrum”菜单项，此时屏幕将如图 11.2 所示，显示波形在不同频率下的相对振幅。尽管此图未将人类听觉的非线性特质考虑在内，但它仍然有助于音效的调整。

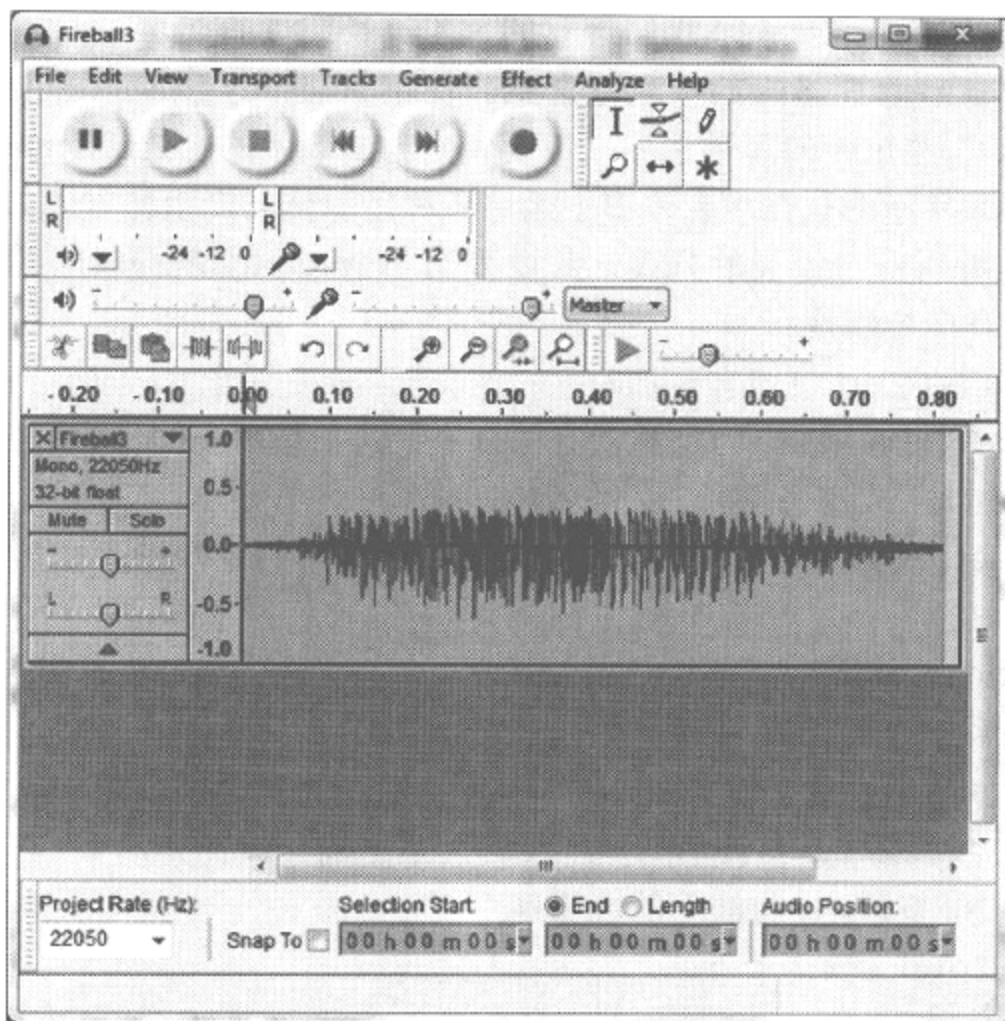


图 11.1 载入 Fireball3.wav 之后的 Audacity 软件界面

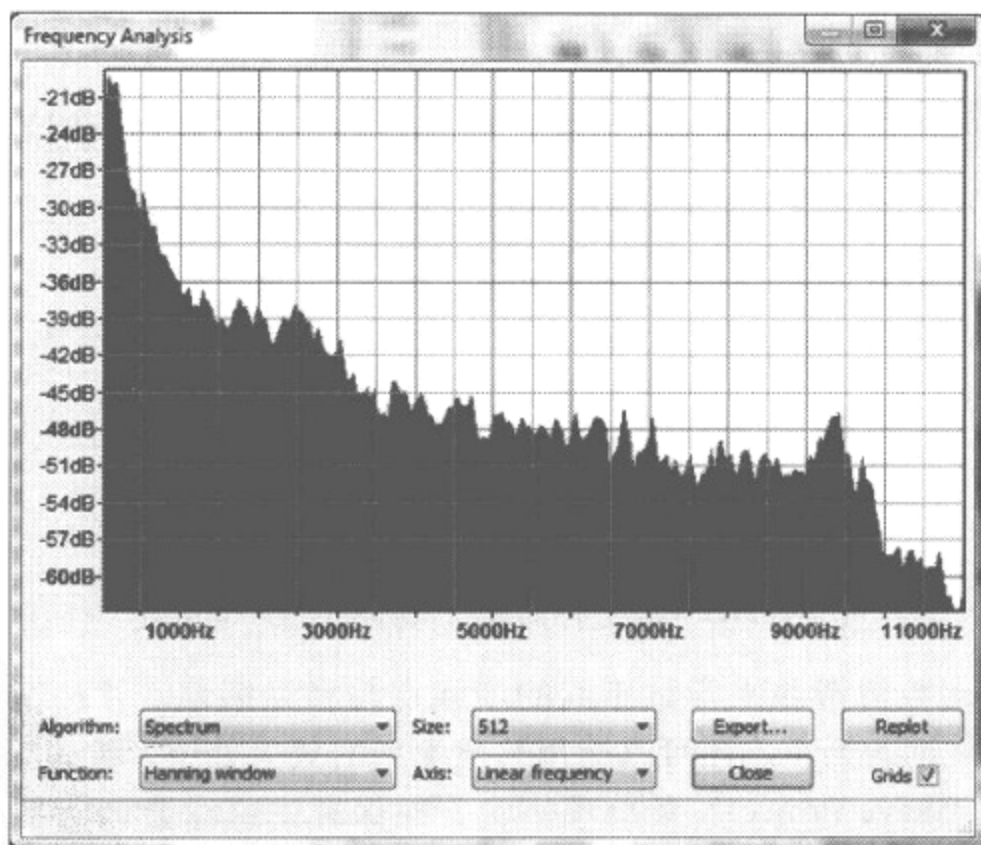


图 11.2 显示 Fireball3.wav 频谱分析的 Audacity 软件界面

低频率部分的这些声音正是我们想要的，不过还需要再增强一些。要让火球的音效听起来有痛击这些作恶多端的吸血鬼之感。使用 Audacity 调整声音的方法之一，就是使用其“Effect”菜单下的各种选项。不要将它与游戏中的音效（Sound Effect）混淆了，该软件的 Effect 指的是在声音波形上运用各种运算使其产生特定的效果。点击该菜单之后，就可以看到是一个很长的列表，其中列出了 Audacity 所提供的各种声音特效处理，比较有用的几个分别是：

- ☐ Amplify（放大）：使声音更加响亮
- ☐ Bass boost（低音增强）：增强低音部分——这正是本范例将要用到的选项
- ☐ Change pitch（改变音高）
- ☐ Compressor（压缩器）：将声音波形的振幅进行非线性压缩
- ☐ Echo（回音）
- ☐ Equalization（均衡化）
- ☐ Fade In（淡入）
- ☐ Fade Out（淡出）
- ☐ Inverter（反转器）：将波形的振幅反转（不是倒播声音）
- ☐ Leveller（平整器）：将波形的波峰削平
- ☐ Noise Removal（噪声消除）
- ☐ Normalize（规格化）：调整振幅以移除直流偏移（DC Offset），并依照给定的值将音量标准化
- ☐ Reverse（逆向播放）：将波形按照时间逆向播放
- ☐ Truncate Silence（切掉静音部分）

选择 Bass Boost 菜单项，增强火球音效的低音部分。会弹出如图 11.3 所示对话框。

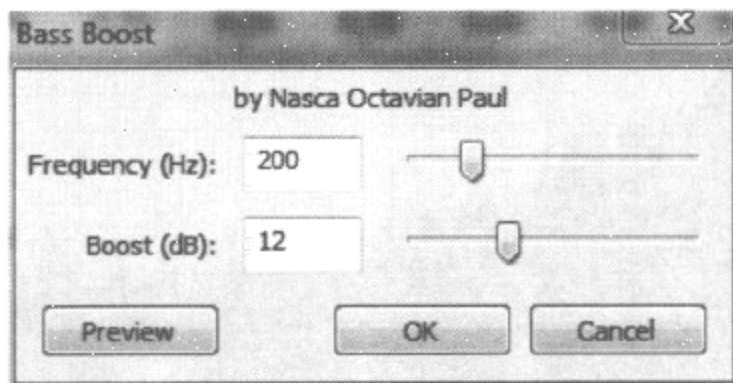


图 11.3 对 fireball3.wav 进行低音增强所使用的 Audacity 对话框

可以通过拖动数值滑块来尝试不同的增强点与增强振幅，并按下“Preview”按钮试听，直到效果满意为止，还可以运用其他类型的效果处理。编辑完成之后，选择“File”菜单下的“Save Project”菜单项，将文件保存。这样会创建扩展名为.aup 的 Audacity 项目文件，并将所有的改动存于其中。然而它并不会生成新的 WAV 文件。如



果要生成新的 WAV 文件,则需要点击“File”菜单下的“Export...”菜单项,这样就可以将波形文件以原始的 WAV 格式保存了。此外还可选择用其他编码器保存,例如 MP3、Ogg/Vorbis 等等。

以 .wav 格式保存的文件大小是 18.01KB,不算太大,不过游戏中需要导入很多这样的文件作为音效,所以总体看来也要占用相当一部分空间。如果想以 MP3 格式保存文件的话,在 Windows 系统下,Audacity 会要求您下载名为 lame\_enc.dll 的编码器程序库。在笔者所居的美国,法律对 LAME 库的使用授权问题界定得不是很明确,尽管在现实中,MP3 许可证持有者不会因为笔者在游戏或书中使用 MP3 而提出异议,但是有一种更稳妥的方式能够完全规避风险,那就是使用 Ogg/Vorbis 格式。

Ogg/Vorbis 使用心理声学技术对音频文件进行压缩,这一点与 MP3 一样,区别是此格式是开源的,不需要使用许可。Audacity 本身支持 Ogg/Vorbis 格式的编码器,使用“File”菜单下的“Export...”菜单项,在弹出对话框的“Save As Type”下拉列表框中选择“Ogg/Vorbis”格式,然后以 fireball.ogg 为名将文件保存。它仅有 9.73KB,是未压缩的 WAV 文件大小的一半多一点。试听之后,觉得与原始的 WAV 文件播放效果没有任何差别,所以笔者将其选定为 V3 游戏所使用的音效格式。

笔者以同样的方式创建了用于子弹发射音效的文件 gunshot.ogg,以及用于斧头投掷的音效文件 whiffle.ogg,并将这些文件导入游戏项目的 assets/mfx 文件夹中。

### 11.6.2 创建背景音乐

前文已经提及,本游戏将使用 J. S. Bach 谱写的 Fugue in G Minor<sup>①</sup> 作为背景音乐,本乐曲的版权早已进入公有领域了,所以不会出现使用问题。然而其演奏则是另外一回事,乐曲演奏的授权与乐曲本身的授权是分开的。

幸运的是,公有领域中也能找到以 MIDI 形式演奏的本乐曲。虽说并非所有 MIDI 都是无版权的,但是碰巧可以在 MIDIWorld 网站(其网址在 11.2 节中)上找到它。

为了能在游戏中使用它,首先将其下载,用 MuseScore 软件打开并编辑,如图 11.4 所示。在软件中可以很明显地看到本 MIDI 的作者是 Jason Fortunato,不过网站上并未有其联系方式。在这里谨对进行总谱输入工作的 Jason 表示感谢。

正如读者所见,MuseScore 所提供的界面非常适合音乐编辑,类似的产品 Make Music's Finale 也是如此。此软件并非编辑声音波形,而是编辑五线谱上的音符与休止符。MuseScore 软件非常适合编辑音乐,尤其是 MIDI 这样的乐器接口。

听完整首歌之后,发现有两处需要修改:一个是速度,另一个是声部,也可以说是乐器在五线谱上排布的方式。利用 MuseScore 软件,可以轻松修改这两个属性。尽管用

① g 小调赋格曲,曲目编号为 BWV 578。赋格是复音音乐的一种固定的创作形式,主要特点是相互模仿的声部在不同的音高和时间相继进入,按照对位法组织在一起。——译者注



Audacity 也能够调整音乐速度, 不过将这两处修改用一个软件完成, 更加合理一些。

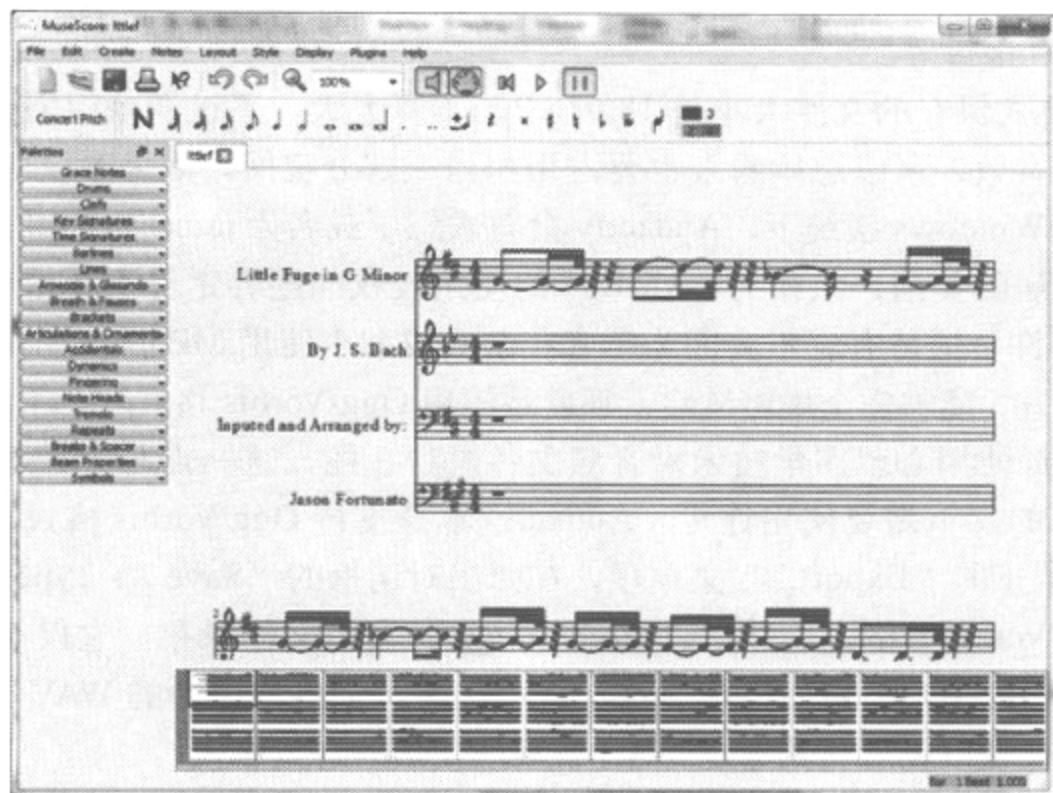


图 11.4 使用 MuseScore 软件编辑 MIDI 格式的 Bach 赋格曲

在 MuseScore 软件中, 点击 “Display” 菜单下的 “Mixer” (调音器) 菜单项, 就会弹出用来查看 MIDI 文件声部的对话框。如图 11.5 所示, 在这个标题为 “MuseScore: Part List” 的对话框中, 可以看到演奏每个声部所用的乐器。对于本文件来说, 所有声部都是用名为 “Strings CLP” 的一种弦乐乐器演奏的。

现在使用调音器面板来修改每个声部所用的乐器, 在可选的 128 种乐器中, 笔者选择了一种自认为合适的乐器类型, 这并不是因为本人乐感很好, 只是觉得它听起来不错罢了。

点击 MuseScore 软件 “Display” 菜单下的 “Play Panel” (声音播放面板) 菜单项, 会弹出如图 11.6 所示对话框, 在此便可以调整音乐速度了。

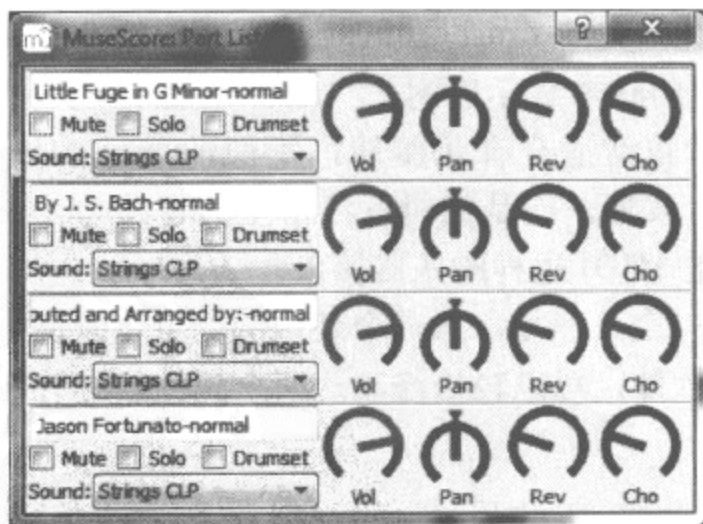


图 11.5 MuseScore 的调音器面板

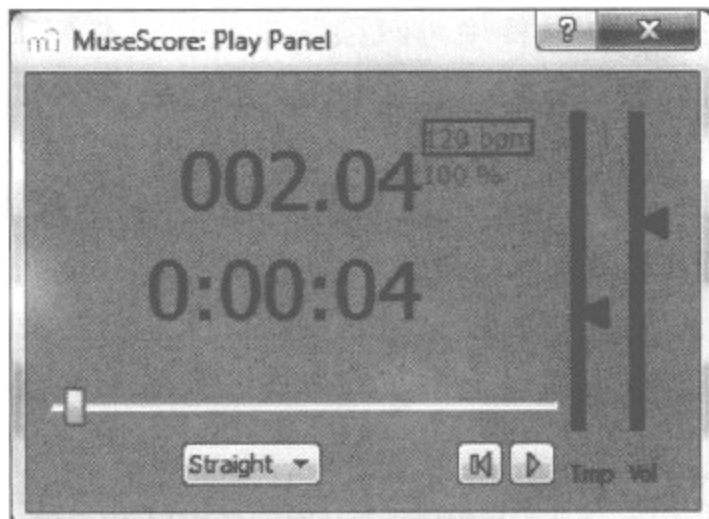


图 11.6 MuseScore 的声音播放面板

对话框中上方的大数字表示歌曲的长度，而下方的数字则表示当前播放点距离歌曲开头的时间。在歌曲长度数字的右上方，可以看到以本歌曲的“每分钟节拍数”（beats per minute, BPM）。底部的水平滑块表示当前播放点在整首歌时间轴上所处的相对位置，而垂直滑块则可用来调整速度与音量。MuseScore 软件的帮助文档没有说明下拉列表框中“Straight/Swing/Shuffle”这几个选项的含义，不过没关系，我们不需要调整该项。将拍子调整到 70BPM，这样音乐节奏就舒缓下来了，听上去有点儿像一曲挽歌（Dirge）。调整之后，整首歌的长度略小于 4 分钟。

选择“File”菜单下的“Save”或“Save As...”菜单项，都可将当前编辑的曲目保存为 MuseScore 的项目文件。不过这里我们使用“Save As Type”子菜单下的各个菜单项，将同一份歌曲保存为不同格式。为了方便比较研究，表 11.1 列出了 MIDI 格式（.mid）、Ogg/Vorbis 格式（.ogg）以及 WAV 格式（.wav）的文件大小。

表 11.1 Bach 所谱 G 小调赋格曲以不同文件格式保存后所占用空间对比表

格 式	文件大小 (KB)
MIDI	12
Ogg/Vorbis	1979
WAV	23 537

WAV 文件非常大，所以做游戏时根本不用考虑此格式，否则每加入一首音乐，游戏安装程序将会增大 20MB 还多。尽管理论上也可以用 MIDI 这种占空间很小的格式来做，可是 Android 手机并没有 MuseScore 所支持的全部 MIDI 乐器列表，所以有时声音听起来会显得很机械化，不够自然。相比之下，使用 Ogg/Vorbis 格式，则可以在保持文件紧凑的同时充分运用 MuseScore 所支持的音乐调节功能，所以最终选定以此格式来保存游戏音效，并将其导入项目的 assets/mfx 文件夹中。

### 11.6.3 修改 V3 游戏的代码

为了将声音功能加入到 V3 游戏中，需要做出如下修改：

- 使用全局的 Boolean 值来保存音乐及音效开关选项。在先前的章节中，我们只是用局部的 Boolean 值将其保存在 OptionsActivity 中，但是现在需要让整个游戏都能访问该值。所幸，利用 Android 系统的 System Preferences 类可轻松完成此任务。
- Music 与 Sound 对象需要在 Activity 对象的生命期回调方法 onPause() 与 onResume() 中进行适当处理，以便在暂停游戏、继续游戏及切换 Activity 时，能够正确地播放声音。
- 背景音乐与音效在播放前必须先载入内存中。

- ❑ 如果玩家打开了音乐选项，那么当进入 StartActivity 时，就要开始播放背景音乐了，直到玩家进入 Level1Activity 为止。
- ❑ 需要调整武器动画，使子弹看起来具有被发射出去的效果，同时使斧头具有被投掷出去的效果。（十字架不需要再做处理，它只需要静候吸血鬼踩上来。）如果玩家将音效打开，那么播放动画的同时还需有同步的音效。
- ❑ 如果音效打开了，则需要在 Level1Activity 的触摸回调方法中，播放吸血鬼被火球击中的音效。

下述程序清单 11.1 (StartActivity.java)、11.2 (OptionsActivity.java) 与 11.3 (Level1-Activity.java) 标注了较为重要的代码修改。

程序清单 11.1 修改后的 StartActivity.java

```
package com.pearson.lagp.v3;

...

public class StartActivity extends BaseGameActivity {
    ...
    static protected Music mMusic;
    private SharedPreferences audioOptions;
    private SharedPreferences.Editor audioEditor;
    ...
    @Override
    public Engine onLoadEngine() {
        mHandler = new Handler();
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        audioOptions = getSharedPreferences("audio", MODE_PRIVATE);
        audioEditor = audioOptions.edit();
        if (!audioOptions.contains("musicOn")) {
            audioEditor.putBoolean("musicOn", true);
            audioEditor.putBoolean("effectsOn", true);
            audioEditor.commit();
        }
        return new Engine(new EngineOptions(true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera).setNeedsMusic(true));
    }

    @Override
    public void onLoadResources() {
        ...
        MusicFactory.setAssetBasePath("mfx/");
        try {
            StartActivity.mMusic = MusicFactory.createMusicFromAsset(
```

```

        this.mEngine.getMusicManager(),
        getApplicationContext(),
        "bach_fugue.ogg");
        StartActivity.mMusic.setLooping(true);
    } catch (final IOException e) {
        Debug.e(e);
    }
}

@Override
public Scene onLoadScene() {
    . . .

    // 开始播放音乐!
    mMusic.play();
    if (!audioOptions.getBoolean("musicOn", false)) {
        mMusic.pause();
    }
    return scene;
}

@Override
public void onGamePaused() {
    super.onGamePaused();
    StartActivity.mMusic.pause();
}

@Override
public void onGameResumed() {
    super.onGameResumed();
    if (audioOptions.getBoolean("musicOn", false))
        StartActivity.mMusic.resume();
    mHandler.postDelayed(mLaunchTask, 3000);
}
}

```

改动的部分均以粗体标出，现在逐个讲述其含义：

- ❑ 增加了一个指向 Music 对象的静态变量。这看起来确实有些蹩脚，不过我们需要在整个游戏项目里都能访问此 Music 对象，而 Android 系统的 SharedPreferences 类又不提供存储和检索对象实例的功能，所以其他 Activity 只能以这种非面向对象的方式来直接访问 mMusic 了。这种方式能勉强应付过去，因为只有一个 Music 对象，所以管理起来不难。
- ❑ 代码还加入了指向 SharedPreferences 对象及其编辑器的私有变量。正如本节开

头所述, 需要使用 SharedPreferences 来保存音乐与音效的开关设定。

- ❑ 在 onLoadEngine() 方法中, 初始化了 SharedPreferences 对象, 如果已经存在上次运行游戏时的设定值, 直接使用这些值即可, 否则就需提交两个默认的值。
- ❑ 还是在 onLoadEngine() 方法中, 代码在构建了 EngineOptions 对象之后, 在其上调用了 setNeedsMusic(true) 方法, 含义是告知 AndEngine, 本模块将会调用 Music 对象的相关方法。
- ❑ 在 onLoadResources() 方法中, 使用 MusicFactory 以项目 assets/mfx 文件夹下的文件来创建 Music 对象, 并将其载入内存中。
- ❑ 调用 setLooping(true) 之后, Android 手机将会持续地播放音乐, 直到玩家抓狂后把它砸向墙壁为止。

#### 程序清单 11.2 修改后的 OptionsActivity.java

```
package com.pearson.lagp.v3;
...
public class OptionsActivity extends BaseGameActivity implements
    IOnMenuItemClickListener {
    ...
    private SharedPreferences audioOptions;
    private SharedPreferences.Editor audioEditor;
    ...
    @Override
    public Engine onLoadEngine() {
        mHandler = new Handler();
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        audioOptions = getSharedPreferences("audio", MODE_PRIVATE);
        audioEditor = audioOptions.edit();
        return new Engine (new EngineOptions (true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,
                CAMERA_HEIGHT, this.mCamera));
    }
    ...
    @Override
    public void onGamePaused() {
        super.onGamePaused();
        StartActivity.mMusic.pause();
    }
    @Override
    public void onGameResumed() {
```

```

super.onGameResumed();
if (audioOptions.getBoolean("musicOn", false))
    StartActivity.mMusic.resume();
mMainScene.registerEntityModifier(new ScaleAtModifier(0.5f,
    0.0f, 1.0f, CAMERA_WIDTH/2, CAMERA_HEIGHT/2));
mOptionsMenuScene.registerEntityModifier(
    new ScaleAtModifier(0.5f, 0.0f, 1.0f,
        CAMERA_WIDTH/2, CAMERA_HEIGHT/2));
}

@Override
public boolean onOptionsItemSelected (final MenuScene pMenuScene,
    final IMenuItem pMenuItem, final float pMenuItemLocalX,
    final float pMenuItemLocalY) {
    switch(pMenuItem.getID()) {
        case MENU_MUSIC:
            if (audioOptions.getBoolean("musicOn",
                true)) {
                audioEditor.putBoolean("musicOn", false);
                if (StartActivity.mMusic.isPlaying()) {
                    StartActivity.mMusic.pause();
                } else {
                    audioEditor.putBoolean("musicOn", true);
                    StartActivity.mMusic.resume();
                }
            }
            audioEditor.commit();
            createOptionsMenuScene();
            mMainScene.clearChildScene();
            mMainScene.setChildScene(mOptionsMenuScene);
            return true;
        case MENU_EFFECTS:
            if (audioOptions.getBoolean("effectsOn", true)) {
                audioEditor.putBoolean("effectsOn", false);
            } else {
                audioEditor.putBoolean("effectsOn", true);
            }
            audioEditor.commit();
            createOptionsMenuScene();
            mMainScene.clearChildScene();
            mMainScene.setChildScene(mOptionsMenuScene);
            return true;
    }
}

```



- ❑ 这段代码依然是在 `onLoadEngine()` 方法中初始化了 `SharedPreferences` 及其编辑器变量, 以便能够获得与修改游戏配置的选项值。
- ❑ 覆写了 `onGamePaused()` 与 `onGameResumed()` 方法, 以便在游戏暂停时关闭音乐, 在游戏继续时再恢复音乐的播放。这两个方法在每次 `Activity` 对象暂停与继续的时候都会调用。在 `onGameResumed()` 方法内, 首先需要判断游戏在暂停之前是不是处于播放音乐的状态, 如果不是的话, 则不需要在游戏继续时播放音乐。
- ❑ `onMenuItemClicked()` 回调方法中的 `switch` 语句代码会根据玩家正在改变的菜单项来采取相应的措施, 例如玩家将音乐开关设置为 “Music Off” 时, 则会关闭正在播放的音乐。相比之下, 音效的处理则要简单一些, 因为本游戏并没有用 `setLoop(true)` 将其设置为循环播放。如果读者不熟悉 `SharedPreferences` 类的用法, 请记得在对其数据做出修改之后, 一定要调用 `commit()` 方法, 这样所做的改动才会提交到共享存储空间。

程序清单 11.3 修改后的 `Level1Activity.java`

```
package com.pearson.lagp.v3;
...
public class Level1Activity extends BaseGameActivity {
...

    private Sound mExploSound, mGunshotSound, mWhiffleSound;
    private SharedPreferences audioOptions;

...
    @Override
    public Engine onLoadEngine() {
        mHandler = new Handler();
        gen = new Random();
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        audioOptions = getSharedPreferences("audio", MODE_PRIVATE);
        return new Engine(new EngineOptions(true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,
                CAMERA_HEIGHT),
            this.mCamera).setNeedsSound(true));
    }
    @Override
    public void onLoadResources() {
...
}
```

```

SoundFactory.setAssetBasePath("mfx/");
try {
    this.mExploSound = SoundFactory.createSoundFromAsset(
        this.mEngine.getSoundManager(),
        getApplicationContext(), "fireball.ogg");
    this.mGunshotSound = SoundFactory.createSoundFromAsset(
        this.mEngine.getSoundManager(),
        getApplicationContext(), "gunshot.ogg");
    this.mWhiffleSound = SoundFactory.createSoundFromAsset(
        this.mEngine.getSoundManager(),
        getApplicationContext(), "whiffle.ogg");
} catch (final IOException e) {
    Debug.e(e);
}

@Override
public Scene onLoadScene() {
    bullet = new Sprite(20.0f, CAMERA_HEIGHT - 40.0f,
        mBulletTextureRegion) {
        @Override
        public boolean onAreaTouched(
            final TouchEvent pAreaTouchEvent,
            final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            switch(pAreaTouchEvent.getAction()) {
                case TouchEvent.ACTION_DOWN:
                    break;
                case TouchEvent.ACTION_UP:
                    fireBullet(pAreaTouchEvent.getX(),
                        pAreaTouchEvent.getY());
                    break;
                case TouchEvent.ACTION_MOVE:
                    this.setPosition pAreaTouchEvent.getX() -
                        this.getWidth() / 2,
                        pAreaTouchEvent.getY() -
                        this.getHeight() / 2;
                    break;
            }
        }
    };
    return true;
}

```

```

        };
        . . .
        hatchet = new Sprite(cross.getInitialX() + 40.0f,
            CAMERA_HEIGHT - 40.0f, mHatchetTextureRegion){
            @Override
            public boolean onAreaTouched(
                final TouchEvent pAreaTouchEvent,
                final float pTouchAreaLocalX,
                final float pTouchAreaLocalY) {
                switch(pAreaTouchEvent.getAction()) {
                case TouchEvent.ACTION_DOWN:
                    break;
                case TouchEvent.ACTION_UP:
                    throwHatchet(pAreaTouchEvent.getX(),
                        pAreaTouchEvent.getY());
                    break;
                case TouchEvent.ACTION_MOVE:
                    this.setPosition(pAreaTouchEvent.getX() -
                        this.getWidth() / 2,
                        pAreaTouchEvent.getY() -
                            this.getHeight() / 2);
                    break;
                }
                return true;
            }
        };
        . . .
    }

    @Override
    public void onGamePaused() {
        super.onGamePaused();
        mGunshotSound.stop();
        mExploSound.stop();
    }

    private void fireBullet(float pX, float pY){
        // 将子弹精灵顺时针旋转 90 度, 迅速向右飞去, 并播放枪声音效
        bullet.registerEntityModifier(new SequenceEntityModifier (
            new IEntityModifierListener() {
                @Override
                public void onModifierFinished(
                    final IModifier<IEntity>
                    pEntityModifier,

```

```

        final IEntity pEntity) {
        Level1Activity.this.runOnUiThread(
            new Runnable() {
                @Override
                public void run() {
                    bullet.setVisible(false);
                    bullet.setPosition(0,0);
                }
            });
    },
    new RotationModifier(0.5f, 0.0f, 90.0f),
    new MoveXModifier(0.5f, pX, CAMERA_WIDTH),
    new AlphaModifier(0.1f, 1.0f, 0.0f));

    mHandler.postDelayed(mPlayGunshot, 500);
}

private void throwHatchet(float pX, float pY){
    // 斧头围绕其中心点旋转着向右方飞去
    hatchet.registerEntityModifier(new ParallelEntityModifier (
        new IEntityModifierListener() {
            @Override
            public void onModifierFinished(
                final IModifier<IEntity>
                pEntityModifier,
                final IEntity pEntity) {
                Level1Activity.this.runOnUiThread(
                    new Runnable() {
                        @Override
                        public void run() {
                            hatchet.setVisible(false);
                            hatchet.setPosition(0,0);
                        }
                    });
            }
        }),
    new RotationAtModifier(5.0f, 0.0f, 5.0f*360.0f,
        20.0f, 20.0f),
    new MoveXModifier(5.0f, pX, CAMERA_WIDTH));
    playSound(mWhiffleSound);
}

private Runnable mPlayGunshot = new Runnable() {

```

```

        public void run() {
            playSound(mGunshotSound);
        }
    };

    private Runnable mStartVamp = new Runnable() {
        public void run() {
            . . .

            asprVamp[i] = new AnimatedSprite(CAMERA_WIDTH - 30.0f,
                startY, mScrumTextureRegion.clone()) {
                @Override
                public boolean onAreaTouched(
                    final TouchEvent pAreaTouchEvent,
                    final float pTouchAreaLocalX,
                    final float pTouchAreaLocalY) {
                    switch(pAreaTouchEvent.getAction()) {
                        case TouchEvent.ACTION_DOWN:
                            /* 有一个吸血鬼消失了吗? */
                            if (. . .
                                playSound(mExploSound);
                            . . .
                        }
                    };
                }
            };
        }
    };
}

```

- ❑ 创建指向 Sound 对象及 SharedPreferences 对象的变量。本 Activity 类的代码不需要 SharedPreferences.Editor 对象，因为它只获取配置，不修改其值。
- ❑ 该类的代码也会在 onLoadEngine() 方法中创建 EngineOptions 对象，不过这次创建完毕之后调用的是 setNeedSound(true) 而非 setNeedsMusic(true)，因为这次需要使用的是与音效有关的功能。
- ❑ 在 onLoadResources() 中，我们使用 SoundFactory.createFromAsset() 方法载入所有的三个音效。
- ❑ 在 onLoadScene() 方法中，需要分别修改子弹对象和斧头对象所在匿名内部类的 TouchEvent 回调方法中的 switch 语句。不过这两处修改略有不同：
  - ❑ 当用户将子弹拖至游戏场地并松开时，需要先将其顺时针旋转 90 度，然后再

发射出去。在播放射击音效的同时，子弹会沿着屏幕向右方飞去。这一系列动作都被封装入 `fireBullet()` 方法了，以便在发射子弹时执行。

- 当玩家选择使用斧头做武器时，需要在播放投掷音效的同时，立刻让其旋转着飞向屏幕右方。经过精细的代码调整，现在斧头于投掷声播放之后，恰好在向右移动的过程中旋转了7圈，然后再飞出屏幕。这一系列动作被封装入 `throwHatchet()` 方法中，以便在投掷斧头时执行。
- 尽管音效在播放完毕之后就会自己停止，不过 `onGamePaused()` 的代码还是直接停止了正在播放的所有音效。这个 `Activity` 类中并不需要 `onGameResumed()` 方法，因为它在从别的 `Activity` 对象切换回来时不需要做任何处理，而且也没有其他附属的 `Activity` 子对象。
- `fireBullet()` 方法使用一系列修改器对象来执行游戏所需的旋转与移动。由于我们不需要子弹发射音效在子弹旋转时就播放，所以将其封装为名为 `mPlayGunshot` 的 `Runnable` 任务投递到系统，令其延时执行，而这段延时恰好让子弹完成旋转动作。还有一种方法，就是向 `RotationModifier` 对象注册监听器，在其中播放音效，不过本书所采用的方法更加简单些。在这一系列动作完成之后，代码将子弹对象的位置设置为 (0, 0)，这么做主要是考虑到在将来的成品游戏中，可以让玩家在经过某段时间之后再次使用子弹当做武器。
- `throwHatchet()` 方法与 `fireBullet()` 方法类似，但其音效是立即播放的。
- 本类的最后，将检查音效设置的逻辑提取到名为 `playSound()` 的方法中，这样就可以避免出现大量重复的检测代码。

## 11.7 总结

其实本章只是对游戏声音的制作进行了很浅的研究而已，制作优秀的背景音乐与逼真的音效，其本身就是一门单独的艺术。如果读者想要组织一支游戏开发团队，那么非常需要吸纳一位声音制作方面的专家进入团队，令其专注于游戏声音的制作。

本章浅谈了音乐对于烘托游戏气氛的重要性。这正是当下热门的研究项目，有企业已经致力于将音乐所烘托的心情在无人干预的情况下提取总结出来。不过就眼前的游戏制作来说，仍需要有专人为游戏选择或制作背景音乐。

音效和音乐一样，也是很主观的东西，很难为游戏制作出合适的音效。可以支付一定金额的授权使用费而使用专家制作好的音效，这样做有时效果也很好。

所幸用于创建与编辑音乐及音效的工具非常多，甚至其中某些开源软件所提供的功能都足以使用户花时间学习好一阵子了，而功能更加丰富且支持更多格式（其中有些会引起版权争议）的商业软件，也能以合理的价格购得。

`AndEngine` 现在所提供的接口还不能让游戏开发者利用 `Android` 系统所提供的诸如



JetPlayer 声音引擎等功能。JetPlayer 可以利用乐曲中的各个小节动态地谱写出一大段乐曲来。如果 AndEngine 能够充分利用 JetPlayer 等 Android 系统所提供的功能,那么它将会在现有基础上进一步拓展应用范围。

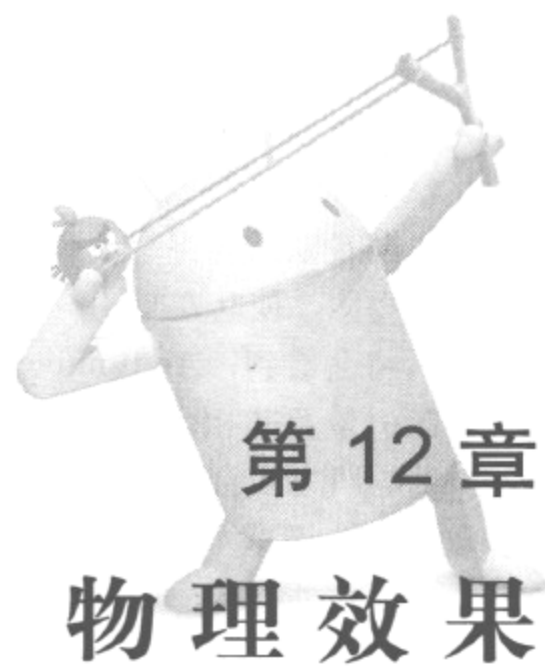
## 11.8 习题

1. 不知读者感受如何,反正笔者觉得三种武器中只为子弹与斧头设计了音效与动画,而丢下十字架不管,有点可惜,不如为它加入一小段音效吧?当玩家将其从武器库拖至屏幕上时,播放一小段合唱效果的“Onward Christian Soldiers”<sup>⊖</sup>。这段音效不需要制作得很华丽,用你的电脑麦克风将其录下来,再使用 Audacity 编辑一下就好了(当然如果读者愿意,也可以直接用 Audacity 录制)。这需要在 Level1Activity 中增加游戏音乐播放功能,现有版本的 Level1Activity 只能播放音效。

2. 在游戏项目的 assets/mfx 文件夹下,也有一份 MIDI 格式的 Bach 赋格曲,现在修改 StartActivity.java,令其使用该文件作为背景音乐。这个版本的演奏速度稍微有些快,不巧的是 MediaPlayer 并没有提供控制播放速度的接口。

3. 正如 Android 开发者文档所述,SharedPreferences 不仅可以用来存放用户设置,还能用来存放其他有用的信息。修改 StartActivity 的代码,使其在启动时创建一个名为“scores”的 SharedPreferences 对象,以键值对的形式保存玩家在每个游戏场景中所取得的分数(现在只有“WAV”及“Level1”两个游戏场景)。再写一个名为的 ScoresActivity 类,用其显示分数。

⊖ 顺便说一下,“Onward Christian Soldiers”(中文意为“前进吧!基督战士”)有一段有趣的历史,它是由 Sabine Baring-Gould 所填词的一首圣歌。而后来将其重新谱曲的人,则是知名歌剧创作组合“吉尔伯特与萨利文”(Gilbert and Sullivan)中的阿瑟·萨利文爵士。这个 G&S 组合正是推动国际版权法的先驱,因为很多人在未事先征得同意或购买使用权的情况下就擅自使用了其歌剧作品,这正好提醒我们,在游戏中使用任何具有知识产权的作品时,一定要获得产权所有者的许可才行。不过这里要声明一点:“Onward Christian Soldiers”这首歌早已属于公有领域,而且习题中所用的这个(糟糕的)演唱版本正是读者您自己哼出来的,所以不用担心授权问题。——译者注



- 12.1 Box2D 物理引擎
- 12.2 构建物理学游戏的关卡
- 12.3 AndEngine 与 Box2D
- 12.4 《愤怒的村民》：V3 中的物理学小游戏
- 12.5 实现 IV 游戏
- 12.6 总结
- 12.7 习题

在游戏开发中，“物理”(physics)一词指的是所有与真实世界物理现象相仿的效果。并非每种游戏都需要运用物理效果，比方说，棋牌游戏通常就不需要物理效果。而另外一些游戏则需要完全依赖物理效果来运作，比方说在天空中划过一道弧线的导弹击中了一堆物体，使其在重力的作用下纷纷散落。

稍后将会看到，AndEngine 以扩展包的形式提供了一套完整的物理引擎，本章将深入研究此引擎及其相关工具，并且为 V3 游戏增设一个基于物理效果的小游戏。本书对物理引擎的研究不会面面俱到，而是使玩家对其有一个初步的了解和掌握。

做好的小游戏如图 12.1 所示，玩家可以通过触摸或拖拽游戏中的物体来向屏幕场景中投掷小木桩，甚至可以把已有的木桩砍断。游戏的目标是使吸血鬼的头碰到地面。

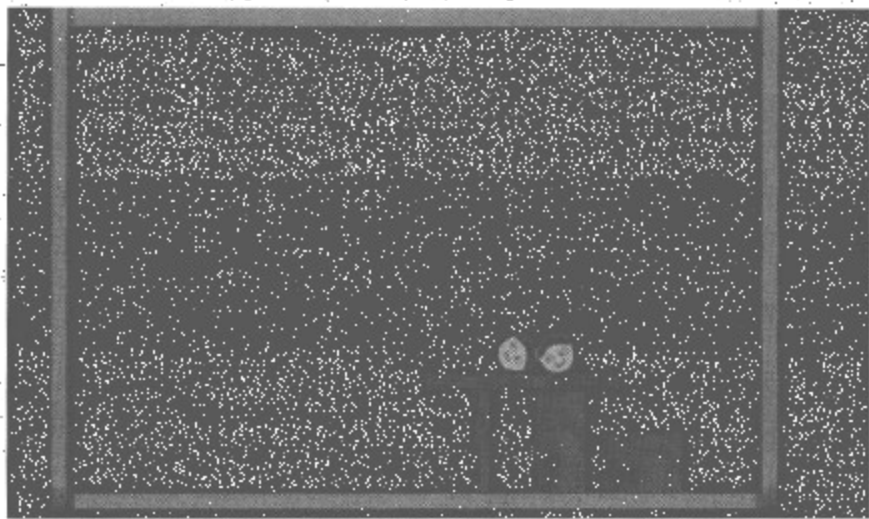


图-12.1 V3 中的物理小游戏

## 12.1 Box2D 物理引擎

AndEngine 所使用的物理引擎叫做 Box2D，该引擎原来是 Erin Catto 为了讲授物理引擎的制作技术而以 C++ 语言编写的，其后被拓展到了很多应用领域，并且以许多其他的编程语言进行了移植，同时还被集成进了众多 2D 游戏引擎之中。

本书受篇幅所限，只能研究 Box2D 引擎的一小部分，不过本章所包含的知识及范例代码能够为读者深入研究该引擎起到抛砖引玉的作用。下面将开始讲述如何在 AndEngine 中制作出基于 Box2D 物理引擎的游戏场景。如果想对其进行更加深入的探讨，笔者强烈建议查阅以下站点的 Box2D 使用手册：

<http://www.box2d.org/manual.html>

### 12.1.1 Box2D 概念

在开始讨论诸如 Box2D 这种物理引擎之前，需要先界定一些名词，这些术语在 Box2D 手册中都有更加详细的讲述，如果读者需要深入研究的话，可以查阅。



### Units (物理单位)

Box2D 所执行的物理模拟效果很依赖于质量、速度等属性所使用的物理单位。Box2D 能够很好地支持 MKS (米-千克-秒) 单位体系, 而且对于“通常”大小物体之间的相互作用, 模拟得很好。这里所说的“通常”物体指的是大小在 0.1 米至 10 米之间的运动物体, 或 50 米以内的静止物体。稍后将会介绍一个非常有用的常数: `PIXEL_TO_METER_RATIO_DEFAULT`, 它可以用来在像素与物理坐标之间换算。直接在 Box2D 中使用像素单位模拟出来的效果不是很逼真。

### World (虚拟世界)

Box2D 构建了一个虚拟世界, 在其中可以模拟真实世界中的物体运动。Box2D 中的虚拟世界是由运动物体 (body)、固定器 (fixture)、关节 (joint) 以及约束器 (constraint) 共同组成的。

### Rigid-Body (刚性物体)

物体是 Box2D 引擎模拟物理效果的基础所在, 其本身只有为数不多的属性, 不过它可以同本节稍后将要介绍的形状、固定器、约束器一起搭配, 制作出丰富的物理效果来。Box2D 对于物体的主要限定就是, 它必须是刚性物体, 也就是不会变形。根据 Box2D 手册, 刚性物体的定义是:

一团如钻石般坚硬的物质, 其中任何两部分之间的距离永远恒定不变。

物体的类型又分为三种:

1. 静止的: 此种物体是固定不动的, 用户可以设定其位置, 不过这并不会导致物理效果模拟。静态物体看起来好像是具有无限大的质量 (在创建时用 0 表示其质量), 并且只能同动态物体碰撞。
2. 运动的 (Kinematic): 此种物体只会凭借其速度来移动, 不具有物理效果模拟, 也不会对受力做出反应。它们也可以认为是质量无限大的, 并且只同动态物体碰撞。
3. 动态的: 此种物体是运动的、完全具备物理模拟效果的物体, 它们的质量是非 0 值, 并且可以同其他静止的、运动的以及动态的物体相碰撞。如果把某动态物体的质量设置为 0, 那么它将自动重置为 1kg。

### Shape (形状)

Box2D 支持两种用于模拟真实世界的基本形状: 圆形与多边形。当使用 Box2D 来创建物体时, 通常需要使用一种形状搭配下面讲述的固定器 (下节讨论)。当 Box2D 模仿物体之间的相互物理作用时, 它会使用物体的形状作为碰撞检测的标准。

### Fixture (固定器)

固定器可以用来将形状关联至物体上, 除此之外, 它还提供了物体的密度、弹性以

及摩擦系数。同一个固定器的属性值可能会用于构建多个物体，所以最简单的办法就是预先将其定义好，然后创建物体时反复使用。

### Constraint（约束器）

约束器用于阻止物体在某个方向上进行移动，2D 世界下的无约束物体可以有三个方面的自由度：改变横坐标、纵坐标以及旋转角度。约束器可以用来限制针对上述三个属性的更改。

### Joint（关节）

在 Box2D 中，关节用于连接两个物体，它可以有一定的形状限制（例如 L 形的肘关节不可以反向弯折），也可以有用于驱使其移动的马达。Box2D 支持很多种关节，诸如转动关节、棱柱关节、距离关节、滑轮关节、鼠标关节、焊接关节等，以上各种关节的详情请查阅 Box2D 用户手册。

### Sensor（传感器）

游戏中有些物体需要参与碰撞检测，但是并不需要针对碰撞做出反应。传感器就是用来满足这种需求的，稍后将会看到，任何物体都可以声明为传感器对象。

### Bullet（子弹）

Box2D 在进行动态物体与静止物体之间的碰撞检测时，需要扫描动态物体运动路径上的每一帧（这又叫做持续碰撞检测），以防“动态物体如穿越隧道般快速地穿过静态物体”这种碰撞形式被忽略掉。考虑到执行效率，Box2D 在检测动态物体之间的碰撞时通常不会使用这种持续碰撞检测方式。然而，如果用户将某静态物体标注为子弹，Box2D 就将使用持续碰撞检测来侦测它与其他动态物体的碰撞。如果游戏中包含迅速移动的动态物体，并且需要与其他动态物体进行碰撞检测，那么就应当将该物体标注为子弹。

在 Android 模拟器上运行物理引擎的效果不是很好，因为帧率很低。如果在游戏测试中发现大量的隧道式穿越，那么可以将所有物体的持续碰撞检测功能打开。比如说，如果 `mPhysicsWorld` 是您所创建的物理模拟环境，那么可使用如下代码打开持续碰撞检测：

```
mPhysicsWorld.setContinuousPhysics(true);
```

当然了，这么做的结果是所有物体都将运行得更加缓慢，因为需要做更多的运算。最好在真机上测试该功能。

## 12.1.2 设定 Box2D

需要依照如下步骤构建物理模拟环境并启动模拟效果：

1. 使用类构造器创建 PhysicsWorld 对象，在创建时可以指定重力加速度值，也可以使用“allow sleeping”功能来关闭对闲置对象的模拟以减少 CPU 消耗。

2. 创建模拟环境中所需的静止物体，可能包括地板、墙壁、天花板，等等，这些都是防止物体飞出屏幕范围的阻挡物。创建这些静止物体需要分两步：

- 创建物体（通常是 Sprite 对象）所具有的形状。
- 创建物体对象本身，并将其加入模拟环境中，同时使用适当的固定器将物体同形状关联起来。

3. 将物体的形状加入 AndEngine 的 Scene 对象中，以便显示。

4. 使用 PhysicsConnector 对象将 Sprite 对象连接到 Physics 对象上。

5. 将 Box2D 的 PhysicsWorld 对象作为一个 UpdateHandler 注册到 Scene 对象上，使它能够更新 Sprite 对象的位置。

在本章最后的代码中，我们将实践上述过程。

## 12.2 构建物理学游戏的关卡

物理学游戏经常由许多关卡组成，在每一关中，玩家将会面对由各种物体所组成的谜题，并将其解开，获得一定的分数。作为游戏开发者来说，可以用许多种方式来设计关卡：

- 用代码创建关卡：可以直接写 Dalvik 代码来创建物理模拟环境所无的物体，并将其摆放到合适的位置。然而开发者必须自己完成从代码到物理空间的思维转换，这个过程非常容易出错，而且会导致工作量增大，故而不推荐使用这种方式。

- 使用 Box2D 关卡编辑器：有些开发者编写了用于关卡编辑的程序并慷慨大方地将其放在网上供大家使用。这类编辑器通常来说是很直观的，用户只用将需要的物体拖放到桌布上即可，此外还可以调整其大小，为其附加所需的属性，等等。安排好关卡之中的物体之后，可以用编辑器将其保存为 XML 文件，该文件中包含了物理模拟环境下所有物体的属性。接下来开发者就可以在游戏中读入它，用以动态地创建游戏关卡了。

- 编写自己定制的关卡编辑器：很多开发者都喜欢为自己的游戏制作一款专用的编辑器。制作这样的编辑器并不是很困难，然而它不在本书的讨论范围内。

本书将使用一款名为 Bison Kick 的 Box2D 关卡编辑器，它是由 Jacob Schatz 所编写的一款 Flash 应用程序。其 Beta 版的网址如下，请读者使用您喜好的网页浏览器打开它：

<http://www.jacobschatz.com/bisonkick-beta/>

图 12.2 是该软件的界面截图，在本章稍后的范例演示中将会讲到如何使用此编辑器，以及如何将该编辑器所生成的关卡载入到 AndEngine 中。



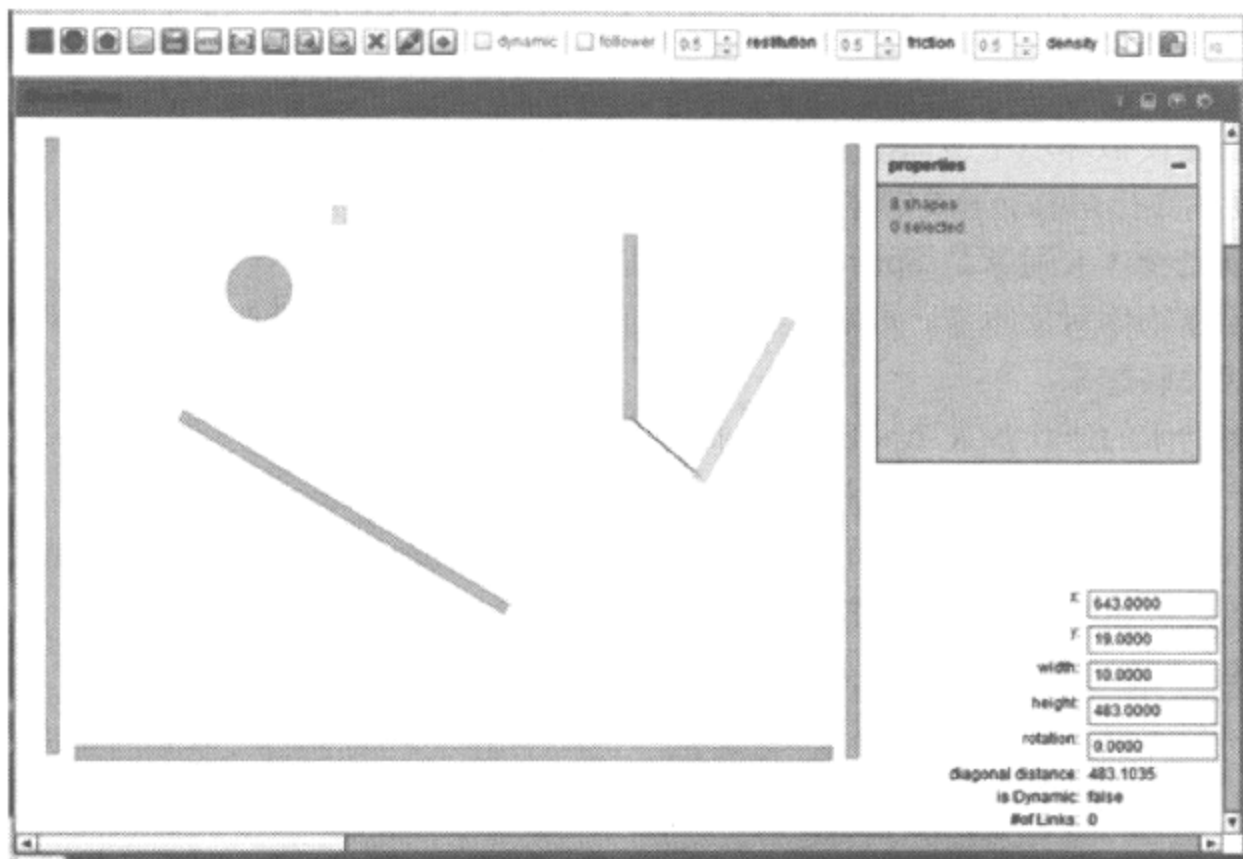


图 12.2 Bison Kick 关卡编辑器界面截图

## 12.3 AndEngine 与 Box2D

AndEngine 是以扩展包的形式来支持物理引擎的，所以必须将其下载并单独安装。AndEngine 使用了由 Mario Zechner 领导开发的 JNI 封装库 libgdx，该库提供了用以访问底层 C++ 版 Box2D 库的全部接口。AndEngine 提供了一层封装后的 API，通过它可以轻松地访问 libgdx 所提供的功能。

### 12.3.1 下载 AndEnginePhysicsBox2DExtension 并将其加入游戏项目

要使用 Box2D 物理引擎，需要为 Eclipse 项目增加两个库。第一个库是包含 libgdx 封装库的 AndEngine 扩展包，第二个则是含有 C++ 版 Box2D 的 ARM 库。取得这两个库的最简便方式就是从 AndEngineExamples 项目中将其拷贝出来。如果想从源代码直接编译 AndEngine 扩展包的话，请参阅下面的“用源代码构建 AndEngine 扩展包”这一段文字。

#### 用源代码构建 AndEngine 扩展包

虽说 AndEngine 的维护者会及时更新 AndEngineExamples 项目下的 .jar 文件，不过有时您也许需要从头开始构建这些扩展包。开源程序的优势之一就是可以在必要的时

候修改源代码，并在开发工程中使用重新构建后的 .jar 文件。

不论构建哪个扩展包，甚至是整个 AndEngine 项目，所用的步骤都是一样的：

1. 使用您所喜好的 Mercurial 代码库管理方式（Windows 系统通常用 TortoiseHg，Linux 与 Macintosh Os 通常使用命令行）从 code.google.com 网站上将源代码下载下来。AndEngine 及其每个扩展包均有独立的维护网站。Mercurial 将会在开发系统上建立一份源码库的拷贝。

2. 在 Eclipse 中，新建 Android 项目。在弹出的对话框中，选择“Create Project from Existing Source”，然后按下“Location:”按钮，选定经由 Mercurial 所下载的代码库。选定之后按下“Finish”按钮，Eclipse 将会建立一个名字很长的项目，其名称与扩展包的名称相同。

3. 在“Project Explorer”视图的项目名称上右击鼠标，选择弹出菜单中的“Export...”菜单项，在接下来的对话框中，选择“Java”节点下的“JAR file”子节点，并点击“Next”按钮，现在的“JAR Export”对话框将会如图 12.3 所示。

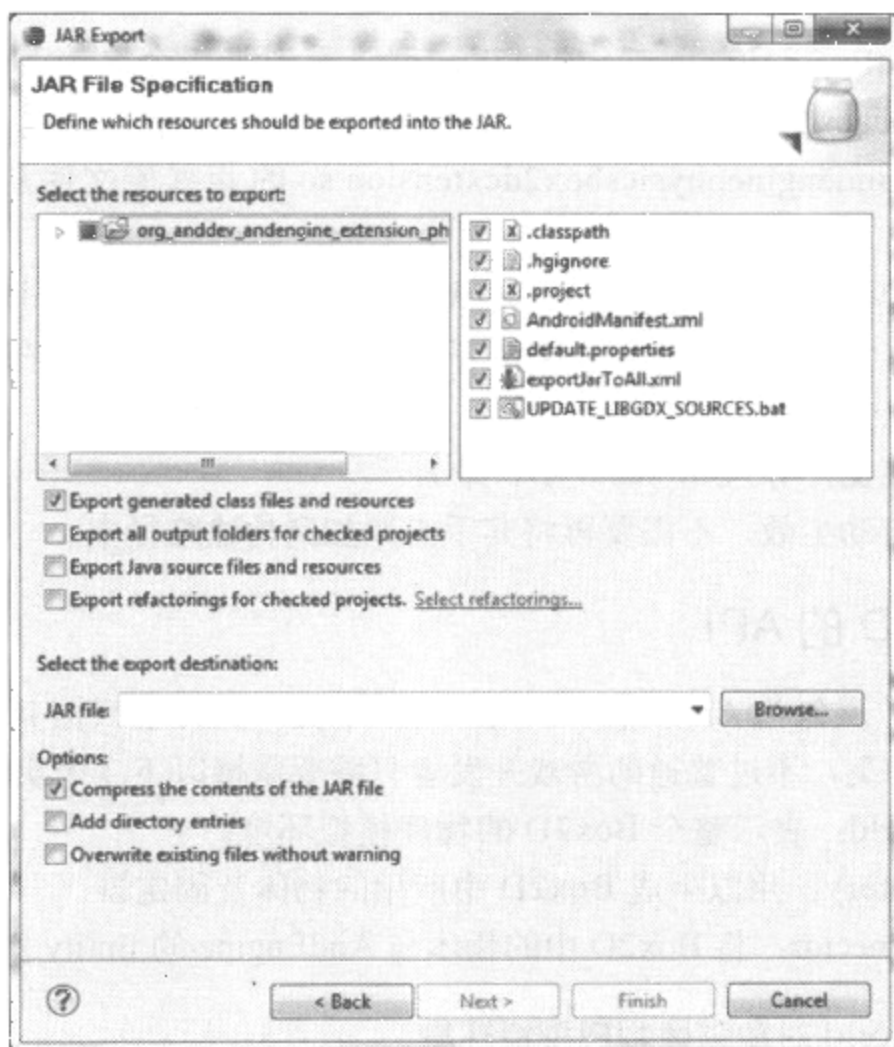


图 12.3 JAR Export 对话框

4. 在“Select the resources to export:”标签右下方的面板中，必须将所有项目都取消选定。而且，必须在“Select the export destination”下方的文本框中指定导出扩展包

的目标位置，该位置需要包含扩展包的文件名，而不仅仅是目录。

5. 点击“Finish”按钮之后，Eclipse 会将扩展包以 .jar 文件的形式导出至指定位置。

**andenginephysicsbox2dextension.jar**

在 AndEngineExamples 项目的网站（网址为 <http://code.google.com/p/andengine-examples/>）上可以找到很多基于物理效果的 Activity 类代码。打开项目的 lib 文件夹，将会看到所有的 AndEngine 扩展包都以 .jar 文件的形式存放于其中，这中间也包括有 andenginephysicsbox2dextension.jar。AndEngine 开发团队，尤其是 Nicolas，在及时更新扩展包这件事上做得很好。

1. 下载物理引擎扩展包对应的 .jar 文件，将其保存至您的开发系统上。
2. 将 .jar 文件导入项目的 lib 文件夹中。
3. 在 Eclipse 软件的 Project Explorer 视图中，右击此 .jar 文件，在弹出菜单中点击“Build Path”子菜单下的“Add to Build Path”菜单项。

**armeabi**

在同一个 AndEngineExamples 项目下，可以看到另一个名为 libs 的文件夹，其中又有一个名为 armeabi 的子文件夹。

1. 将名为 libandenginephysicsbox2dextension.so 的共享库文件从子文件夹 armeabi 下载至开发系统上。
2. 在 Eclipse 项目中再创建一个名为 libs 的顶级子目录（在 Project Explorer 视图上右击，在弹出菜单中选择“New”子菜单下的“Folder”菜单项）。在该子目录下创建名为 armeabi 的子目录。
3. 将刚才的库文件导入 armeabi 子目录。
4. 此库将会自动生效，不需要再将其手工添加到构建路径中。

### 12.3.2 Box2D 的 API

Box2D 物理扩展包为 AndEngine 增加了 16 个类，同时也根据 libgdx 库定义了一些扩展包自己所用的类，不过普通的游戏开发者只需要掌握以下 3 个类即可：

- PhysicsWorld：表示整个 Box2D 的物理模拟环境。
- PhysicsFactory：用以生成 Box2D 中所用的物体及固定器。
- PhysicsConnector：将 Box2D 中的物体与 AndEngine 的 Entity 实体对象关联起来。

#### 用 PhysicsWorld 对象创建物理模拟环境

使用 PhysicsWorld 类的构造器来创建游戏动作所发生的物理模拟环境，将创建之后的对象加入 AndEngine 的 Scene 对象中，这样它就会显示出来了。该类有两个构造器：

```
PhysicsWorld(final Vector2 pGravity, final boolean pAllowSleep)
PhysicsWorld(final Vector2 pGravity, final boolean pAllowSleep, final int
              pVelocityIterations, final int pPositionIterations)
```

其中的参数不难理解:

- Vector2 pGravity: 该参数用于设定虚拟环境中的重力加速度值。Vector2 这个类是我们首次遇见的, 它用于表示 Box2D 中的二维向量。如果需要在模拟环境中加入重力效果, 通常使用如下代码构建该对象:

```
new Vector2(0, SensorManager.GRAVITY_EARTH)
```

上述代码利用了 Android 系统所提供的重力加速度常数, 这在 MKS 单位体系中很方便。如果不需要重力效果, 则使用如下参数值构建对象:

```
new Vector2(0, 0)
```

Android 系统提供了月球、太阳以及包括冥王星在内所有行星的重力加速度值, 最后还有一个名为 GRAVITY\_DEATH\_STAR\_I<sup>⊖</sup> 的常数, 笔者不知道到这个值是如何计算出来的, 不过它确实存在。

- boolean pAllowSleep: 如果该值为 true, Box2D 将会停止对于闲置物体的模拟计算, 通常说来, 这是个好的做法, 因为可以借此减少 CPU 消耗, 提高游戏执行效率。
- int pVelocityIterations: 该参数用于指定 Box2D 引擎计算速度时的迭代次数。如果用第一个构造器来构建 PhysicsWorld 对象, 那么 pVelocityIterations 参数将会使用默认值 8。可以将它设置为小于 8 的值以提高性能, 不过这么做将会在计算的精确度上打折扣。
- int pPositionIterations: 与上个参数相似, 本参数是用于指定进行位置计算时的迭代次数。它的默认值也是 8。

也可以使用 PhysicsWorld 对象的方法来创建与管理物体之间的关节对象。那些方法将会在本章稍后的小节里讲述。该类中还有很多方法也可用于设置与获取物理模拟环境的各种属性, 如果读者感兴趣的话, 可以在以下网址查看 PhysicsWorld.java 的源代码:

<http://code.google.com/p/andenginephysicsbox2dextension/source/browse/src/org/anddev/andengine/extension/physics/box2d/PhysicsWorld.java>

### 使用 PhysicsFactory 创建物体

有了 PhysicsWorld 对象之后, 就可以用物体、形状、固定器及关节这些组件来创建游戏模型了。AndEngine 提供了名为 PhysicsFactory 的单例类用以完成上述任

⊖ Death Star 是指美国科幻题材系列电影《星球大战》(Star Wars) 中的卫星武器死星 I 号。——译者注

务。不需要实例化 PhysicsFactory 类，AndEngine 已经将它实例化了，它的用法与 TextureRegionFactory、SoundFactory 等其他单例类类似。该类定义了如下用以创建物体的方法：

```
Body createBoxBody(final PhysicsWorld pPhysicsWorld, final IShape pIShape, final
    BodyType pBodyType, final FixtureDef pFixtureDef)
Body createCircleBody(final PhysicsWorld pPhysicsWorld, final IShape pIShape,
    final BodyType pBodyType, final FixtureDef pFixtureDef)
Body createLineBody(final PhysicsWorld pPhysicsWorld, final Line pLine, final
    FixtureDef pFixtureDef)
Body createPolygonBody(final PhysicsWorld pPhysicsWorld, final IShape pIShape,
    final Vector2[] pVertices, final BodyType pBodyType, final FixtureDef
    pFixtureDef)
Body createTrianglulatedBody(final PhysicsWorld pPhysicsWorld, final IShape
    pIShape, final List<Vector2> pTriangleVertices, final BodyType pBodyType,
    final FixtureDef pFixtureDef)
```

以上这些方法都有一个包含 float pPixelToMeterRation 参数的版本，那个版本用于以默认值（32.0f）之外的值来自己设置像素的缩放比例。其余的参数解释如下：

- ❑ PhysicsWorld pPhysicsWorld：物体存在的模拟环境，通常说来，它就是刚创建的那个 PhysicsWorld 对象。
- ❑ IShape pIShape 或 Line pLine：物体需要具备的形状。只要是几何图形就都可以当 Shape 来用，包括 Sprite 对象或者其他使用 AndEngine 中的 Rectangle 对象所创建的图形。
- ❑ BodyType pBodyType：该参数用于告知 Box2D 该物体的类型，其可取的值如下：
  - ❑ BodyType.StaticBody
  - ❑ BodyType.KinematicBody
  - ❑ BodyType.DynamicBody
- ❑ FixtureDef pFixtureDef：创建该物体所用的固定器，它的创建将在下一小节讲述。
- ❑ TrianglulatedBody：希望读者看到本书时，AndEngine 已经修正了这个拼写错误<sup>⊖</sup>。该方法需要用户将三角形的顶点坐标通过 List<Vector2> pTriangleVertices 参数传入。

### 使用 PhysicsFactory 创建固定器

PhysicsFactory 类也提供了用于创建固定器的方法：

```
FixtureDef createFixtureDef(final float pDensity, final float pElasticity, final float
    pFriction, final boolean pSensor)
```

⊖ 正确的拼写是 triangulate，中文意思是“使之呈三角形”，此处的拼写多了一个“u”。——译者注

```
FixtureDef createFixtureDef(final float pDensity, final float pElasticity, final float
    pFriction, final boolean pSensor, final short pCategoryBits, final short
    pMaskBits, final short pGroupIndex)
```

第一个方法的参数都很简单明了，除了这个可选的 boolean pSensor 参数，如果它是 true 的话，将会使这个固定器成为感应器，它的含义在 12.1.1 节中提过，意思就是只参与碰撞检测，但不碰撞采取任何处理。

### PhysicsConnector 类

该类用于在 AndEngine 的实体对象（例如 Sprite）与 Box2D 的物体之间搭建一座桥梁，以便开发者可以在组合之后的对象身上利用 AndEngine 所提供的诸如修改器等强大的功能，同时 Box2D 也将对这些对象进行物理模拟。

该类有 4 个构造器：

```
PhysicsConnector(final IShape pShape, final Body pBody)
PhysicsConnector(final IShape pShape, final Body pBody, final float
    pPixelToMeterRatio)
PhysicsConnector(final IShape pShape, final Body pBody, final boolean
    pUpdatePosition, final boolean pUpdateRotation)
PhysicsConnector(final IShape pShape, final Body pBody, final boolean
    pUpdatePosition, final boolean pUpdateRotation, final float pPixelToMeterRatio)
```

前两个参数分别是需要连接的形状（比如 Sprite 对象）与物体。值得注意的是，即便在创建物体时已经使用 Sprite 对象作为 Shape 参数值了，还是需要创建 PhysicsConnector 对象的。

pPixelToMeterRatio 这个参数是可选的，它的含义与 PhysicsFactory 工厂方法中的相同，默认值也是 (32.0f)。两个 boolean 类型的参数指定了 Box2D 在模拟物理效果时是否会更新 Sprite 对象的位置与旋转角度。通常我们会需要更新这两项，所以默认值是 true。

### 12.3.3 简单的物理效果范例

如果读者是第一次接触 Box2D 的话，上述材料可能一时间有点儿消化不过来，不过在看了下面这个范例之后，它们就会变得清晰起来。图 12.4 展示了名为 Demolition 的 Activity 程序，它是 AndEngineExamples 演示项目中那个 Activity 程序的简化版本。当用户触摸屏幕之后，会有物体加进来。当手机倾斜时，所有物体会滑至屏幕的角落处。物体之间如果发生碰撞，则会各自弹开。程序清单 12.1 提供了此 Activity 类的代码。



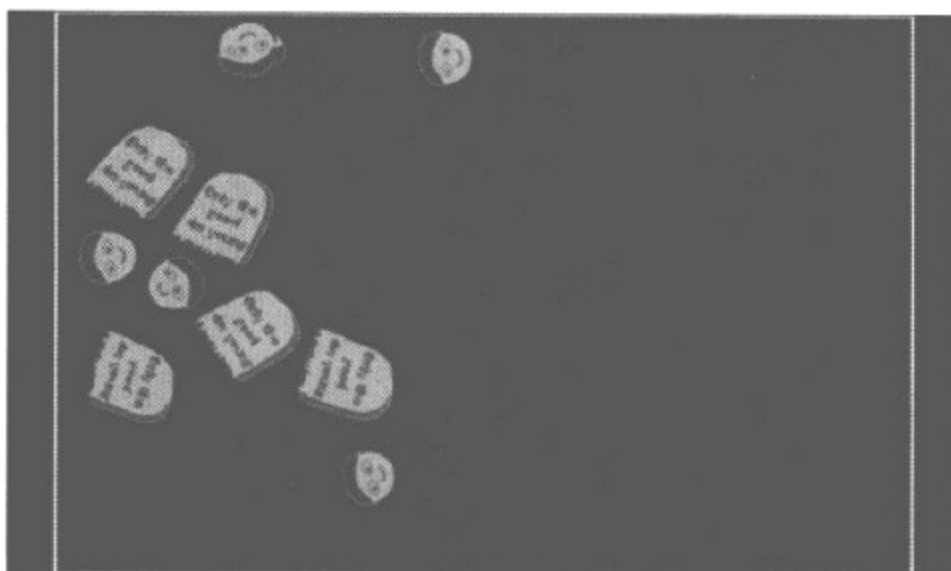


图 12.4 Demolition 范例程序的运行截图

程序清单 12.1 用于演示简单物理效果的范例程序 Demolition 类代码

```

package com.pearson.lagp.demolition;

+imports . . .

public class Demolition extends BaseGameActivity implements
    IAccelerometerListener, IOnSceneTouchListener {

    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    private static final FixtureDef FIXTURE_DEF =
        PhysicsFactory.createFixtureDef(1, 0.5f, 0.5f);
    private static final int MAX_BODIES = 50;

    // =====
    // 字段
    // =====

    private Texture mTexture;
    private TextureRegion mTStoneTextureRegion;
    private TextureRegion mMatHeadTextureRegion;
    private PhysicsWorld mPhysicsWorld;
    private int mBodyCount = 0;

    @Override
    public Engine onLoadEngine() {
        Toast.makeText(this, "Touch the screen to add objects.",
            Toast.LENGTH_LONG).show();
    }

```

```

final Camera camera = new Camera(0, 0, CAMERA_WIDTH,
    CAMERA_HEIGHT);
final EngineOptions engineOptions = new EngineOptions(true,
    ScreenOrientation.LANDSCAPE,
    new RatioResolutionPolicy(CAMERA_WIDTH,
    CAMERA_HEIGHT), camera);
engineOptions.getTouchOptions().setRunOnUpdateThread(true);
return new Engine(engineOptions);
}

@Override
public void onLoadResources() {
    /* Textures */
    mTexture = new Texture(64, 128,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);

    /* TextureRegions */
    mTStoneTextureRegion =
        TextureRegionFactory.createFromAsset(mTexture,
            getApplicationContext(), "tombstone.png",
            0, 0); // 50x50
    mMatHeadTextureRegion =
        TextureRegionFactory.createFromAsset(mTexture,
            getApplicationContext(), "mathead.png", 0, 50); // 32x32
    this.mEngine.getTextureManager().loadTexture(mTexture);
    this.enableAccelerometerSensor(this);
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    final Scene scene = new Scene(2);
    scene.setBackground(new ColorBackground(0, 0, 0));
    scene.setOnSceneTouchListener(this);

    this.mPhysicsWorld = new PhysicsWorld(new Vector2(0, 0),
        true);
    final Shape ground = new Rectangle(0, CAMERA_HEIGHT - 2,
        CAMERA_WIDTH, 2);
    final Shape roof = new Rectangle(0, 0, CAMERA_WIDTH, 2);
    final Shape left = new Rectangle(0, 0, 2, CAMERA_HEIGHT);
    final Shape right = new Rectangle(CAMERA_WIDTH - 2, 0, 2,
        CAMERA_HEIGHT);

    final FixtureDef wallFixtureDef =
        PhysicsFactory.createFixtureDef(0, 0.5f, 0.5f);
    PhysicsFactory.createBoxBody(mPhysicsWorld, ground,
        BodyType.StaticBody, wallFixtureDef);
    PhysicsFactory.createBoxBody(mPhysicsWorld, roof,
        BodyType.StaticBody, wallFixtureDef);

```

```

        PhysicsFactory.createBoxBody(mPhysicsWorld, left,
            BodyType.StaticBody, wallFixtureDef);
        PhysicsFactory.createBoxBody(mPhysicsWorld, right,
            BodyType.StaticBody, wallFixtureDef);

        scene.getFirstChild().attachChild(ground);
        scene.getFirstChild().attachChild(roof);
        scene.getFirstChild().attachChild(left);
        scene.getFirstChild().attachChild(right);
        scene.registerUpdateHandler(mPhysicsWorld);

        return scene;
    }

    @Override
    public void onLoadComplete() {
    }

    @Override
    public boolean onSceneTouchEvent(final Scene pScene,
        final TouchEvent pSceneTouchEvent) {
        if(mPhysicsWorld != null) {
            if(pSceneTouchEvent.isActionDown()) {
                addBody(pSceneTouchEvent.getX(),
                    pSceneTouchEvent.getY());
                return true;
            }
        }
        return false;
    }

    @Override
    public void onAccelerometerChanged(
        final AccelerometerData pAccelerometerData) {
        final Vector2 gravity =
            Vector2Pool.obtain(pAccelerometerData.getY(),
                pAccelerometerData.getX());
        mPhysicsWorld.setGravity(gravity);
        Vector2Pool.recycle(gravity);
    }

    // =====
    // 方法
    // =====

    private void addBody(final float pX, final float pY) {
        final Scene scene = this.mEngine.getScene();
        if (mBodyCount >= MAX_BODIES) return;
        mBodyCount++;

        final Sprite matSprite;

```

```

final Body body;

if (mBodyCount % 2 == 0) {
    matSprite = new Sprite(pX, pY,
        mTStoneTextureRegion);
    body = PhysicsFactory.createBoxBody(mPhysicsWorld,
        matSprite, BodyType.DynamicBody,
        FIXTURE_DEF);
} else {
    matSprite = new Sprite(pX, pY,
        mMatHeadTextureRegion);
    body = PhysicsFactory.createCircleBody(
        mPhysicsWorld, matSprite,
        BodyType.DynamicBody, FIXTURE_DEF);
}

scene.getLastChild().attachChild(matSprite);
mPhysicsWorld.registerPhysicsConnector(
    new PhysicsConnector(matSprite, body, true, true));
}

// =====
// 匿名类与内部类
// =====
}

```

代码中比较重要的部分讲解如下：

- ❑ 代码在定义常数的区域声明了程序中创建动态物体所使用的固定器。对于固定不变的墙壁、地板、天花板等物体，应该分别为其定义不同的固定器。
- ❑ `onLoadEngine()` 与 `onLoadResources()` 方法现在看起来非常相似。`onLoadResources()` 方法在最后打开了加速度计 (Accelerometer)，并将本对象作为监听器注册，以便通过 `onAccelerometerChanged()` 回调方法接收事件通知。
- ❑ `onLoadScene()` 方法包含了一些原来没有遇到过的技巧：
  - ❑ 代码首先创建 `PhysicsWorld` 对象，并将向量值 `[0,0]` 作为重力加速度传给构造器。稍后屏幕倾斜时将会根据加速度计返回的结果来更新这个值。本程序选择跳过针对闲置对象的模拟计算过程。
  - ❑ 接下来使用细长的矩形来分别定义地板、天花板以及左右墙壁的形状，然后创建用于创建静止物体的固定器。该固定器与稍早创建的那个有所不同，它的 `density` 参数值是 0。Box2D 将静止物体的质量视为无限大，所以它们在受力之后不会有加速度。尽管左右墙壁、地板、天花板等物体不需要在屏幕上显示，不过为了效果演示的方便，本范例还是将其显示出来了。
  - ❑ 分别将左右墙壁、地板、天花板创建成箱体 (Box Body)，并将其加入 `Scene` 对象中，以便显示在屏幕上。

- 将 PhysicsWorld 对象作为 UpdateHandler 向 Scene 对象注册，这样的话，每次渲染 Scene 对象时，所有 PhysicsWorld 之中的物体位置将会被更新。
- 覆写 onSceneTouchEvent() 方法，在每次用户触摸屏幕后，向其中加入一个新的物体。
- 覆写 onAccelerometerChanged() 方法，在每次读出新的手机倾斜度后，重新设定物理模拟环境中的重力加速度值，然后将 Vector2 对象回收，以防止频繁的垃圾回收。
- addBody() 方法所做的工作就是在生成 Mad Mat 头像与生成墓碑之间来回切换，并且将新建的 Sprite 对象加入到 Scene 对象中，最后通过 PhysicsConnector 将 Sprite 对象同新建的物体关联起来。

### 12.3.4 关卡加载

前面的讨论中已经讲过，我们需要从编辑器生成的 XML 文件之中载入物理小游戏所需的关卡。AndEngine 提供了一些类，开发者可以用其解析 XML 文件，从中构建关卡和 Box2D 物体。

#### SAX 解析器

AndEngine 使用 SAX (Simple API for XML) 来解析 XML 文件，因为它是 Android 开发包的一部分，所以用它来解析是个明智的选择。如果读者不熟悉 SAX 解析器，可以参阅下列网址所提供的简介：

<http://www.saxproject.org/quickstart.html>

如果想要深入研究 SAX 的话，可以在上述网站或互联网上找到大量的相关文档。对于本书的游戏制作来说，只需要了解以下基本概念和方法即可：

- SAX 是由事件驱动的序列化解析器，它会按顺序访问 XML 文件流的内容，当发现 XML 元素时，会经过回调方法通知客户代码，回调方法就是用于接收在解析 XML 文件流的过程中所发生的事件。
- 每发现一个元素，解析器均会告知回调方法它的元素名称、所含属性以及各属性的值。
- 解析器也会在读到元素末尾时通知监听器。

可以通过 AndEngine 所提供的两套 API 来使用 SAX 解析库：

- LevelLoader，一个专门用于设定解析器并处理解析细节的类，它也会转发回调方法中所含的数据。
- SAXUtils 类，在 SAX 回调方法中使用它可以简化对元素属性值的获取过程。

#### LevelLoader 类

该类与第 9 章中所述的 TMXLoader 类很相似，两者都是通过设置 SAX 解析器来



读取 XML 数据文件的内容。不过这两个类有一个重要的区别,那就是 TMXLoader 类会包揽全部的解析工作,因为它是专门用于解析 TMX 文件的,而 LevelLoader 类则支持许多种不同的关卡编辑器,所以要更加灵活一些,这也意味着有一部分解析工作要交由客户代码来完成。在下一节讲述 Irate Villager 游戏的制作时,将会谈到如何解析 Bison Brick 编辑器所生成的 XML 关卡文件。

创建 LevelLoader 类对象是很简单的,它的构造器没有任何参数:

```
LevelLoader()
```

另外还存在一个以 assets 子文件夹路径作为参数的构造器,不过,与加载 Texture 对象、Sound 对象与 TMX 文件时所用的方式相似,我们也可以在 LevelLoader 对象上调用如下方法来设置资源文件的查找路径:

```
void setAssetBasePath(final String pAssetBasePath)
```

调用 registerEntityLoader() 方法即可向 LevelLoader 对象注册 XML 实体监听器,该方法有两个版本:

```
void registerEntityLoader(final String pEntityName, final IEntityLoader
    pEntityLoader)
void registerEntityLoader(final String[] pEntityNames, final IEntityLoader
    pEntityLoader)
```

这两个方法的区别在于传入的是单个的实体名称还是包含多个实体名称的字符串数组,而第二个参数则通常传入一个覆写了 onLoadEntity() 的内部类实例。

还有一个 LevelLoader 类的方法,用于将 SAX 解析器连接至 XML 文件流:

```
void loadLevelFromAsset(final Context pContext, final String pAssetPath)
```

该方法的两个参数分别是应用程序的环境对象,以及用于查找 XML 文件所用的 asset 子文件夹路径。

### SAXUtils 类

该类提供的方法可以用于在 SAX 解析回调方法中提取属性值,用于获取属性值的方法都是以下列形式命名的:

```
xxx SAXUtils.getXXXAttribute(final Attributes pAttributes, final String pAttribute-
    Name, final xxx pDefaultValue)
```

在实际的方法名中,将会以 Double、Float、Long、Int、Short、Byte、String、Boolean 等具体的数值类型来替换 XXX,而返回值中与 pDefaultValue 参数类型中的 xxx 则会以对应的原始类型来替换。如果觉得有些费解的话,看一下程序清单 12.2 就会清楚其用法了。第一个参数 pAttributes 的值应该是 SAX 解析器通过 onLoadEntity() 回



调方法而传入的那个属性列表对象。

此外还有一组 getXXXAttribute 方法，在找不到相关属性时会抛出异常：

```
xxx SAXUtils.getXXXAttributeOrThrow(final Attributes pAttributes, final String
    pAttributeName, final xxx pDefaultValue)
```

除了会抛出异常之外，该组方法与不抛出异常的那组方法是一样的。

此外该类还有一些方法用于向属性列表中增加新的属性，不过本书所做游戏不需要用到它们。

### 使用 LevelLoader

要使用该类，首先需要定义并注册需要由 SAX 来解析的所有 XML 实体，然后将 SAX 解析器连接至 XML 输入流（通常使用文件构建输入流）。在注册 XML 实体的时候，会传入一个 IEntity 对象，需要在该对象所在的匿名内部类中覆写 onLoadEntity() 方法。在该方法中，使用 SAXUtils 类的方法来检测 XML 实体所具有的属性，并以此构建对应的 Box2D 对象。LevelLoader 对象处理了所有底层的 SAX 解析事务。程序清单 12.2 所展示的这个简短范例，使用 LevelLoader 来解析一个简单的 XML 文件中所含的实体。首先列出的是 XML 文件的内容，其后是 AndEngine 游戏代码。

程序清单 12.2 使用 LevelLoader 解析简单的 XML 文件

---

```
. . .XML file. . .
<level>
    <entity x="40.0" y="100.0" width="20.0" height="10.0"
        isDynamic="true">
    </entity>
</level>

. . .AndEngine game code. . . .
final LevelLoader levelLoader = new LevelLoader();
levelLoader.registerEntityLoader("level", new IEntityLoader() {
    @Override
    public void onLoadEntity(final String pEntityName,
        final Attributes pAttributes) {
        mLevels++;
    }
});

levelLoader.registerEntityLoader("entity", new IEntityLoader() {
    @Override
    public void onLoadEntity(final String pEntityName,
        final Attributes pAttributes) {
        final float x = SAXUtils.getFloatAttributeOrThrow(
            pAttributes, "x");
        final float y = SAXUtils.getFloatAttributeOrThrow(
            pAttributes, "y");
```

```

final float width = SAXUtils.getFloatAttributeOrThrow(
    pAttributes, "width");
final float height = SAXUtils.getFloatAttributeOrThrow(
    pAttributes, "height");
final boolean isDyn = SAXUtils.getBooleanAttributeOrThrow(
    pAttributes, "isDynamic");
if (isDynamic) bodyType = BodyType.DynamicBody;
    . . .

try {
    levelLoader.loadLevelFromAsset(this, "ex");
} catch (final IOException e) {
    Debug.e(e);
}

```

## 12.4 《愤怒的村民》：V3 中的物理学小游戏

为了演示物理效果、物体碰撞以及关卡加载，现在将为 V3 加入另一个小游戏。在第 9 章中，通过加入 Whack-A-Vampire 小游戏演示了瓦片地图的使用。这里将增加一个用于演示物理效果的小游戏，它叫做《愤怒的村民》(Irate Villagers, 简称 IV)，这个名字对于一个吸血鬼题材的游戏来说，起得不错。

该游戏的相关背景是这样的：Miss B 所居村子的村民将一个名为 Mad Mat 的吸血鬼（还记得第 5 章中讨论过的 Mat 吗）与其克隆人包围在一片遍布木板、玻璃渣与碎石的瓦砾堆中。村民们使用弹弓朝吸血鬼发射小木桩。在这个游戏中，木桩并不会刺入吸血鬼体内将其杀掉，而是使吸血鬼掉到地上，这样它就会自己死掉了。村民可使用的小木桩没有数量限制。图 12.1 中所展示的就是这个小游戏的画面。

## 12.5 实现 IV 游戏

为了实现 V3 中的 IV 小游戏，需要增添如下内容：

- 准备如下游戏图片：吸血鬼头像、小木桩、木栅栏以及杂草堆中的玻璃渣与碎石。
- 修改 OptionsActivity.java，让玩家可以在选项菜单中直接进入 IV 游戏。随着范例游戏的制作，这些小游戏将会依次串联起来，不过现在只需要能够直接运行它就好。
- 创建一个新的 IVActivity.java 文件，用该类来创建 IV 游戏的物理模拟环境。IV 小游戏需要完成如下任务：
  - 使用 Bison Kick 编辑器所制作的关卡文件来构建关卡，搭建好游戏中的物理模拟环境。

- 处理 Box2D 引擎对物体位置所做的更新。
- 实现用弹弓朝吸血鬼发射小木桩这个功能。

### 12.5.1 创建关卡

这个小游戏的关卡将使用 Bison-Kick 编辑器来制作，它的界面如图 12.5 所示。

位于绘制区域上方的工具栏中提供了编辑关卡所需的工具。图 12.6 是上图左上角工具栏部分的特写。

图 12.6 中的各个图标所表示的功能简述如下：

- 点击前三个按钮可以分别创建矩形、圆以及多边形，不过这个小游戏中只用到了圆和矩形。也可以使用键盘来创建这些形状，按下 r 键创建矩形，按下 c 键创建圆形。

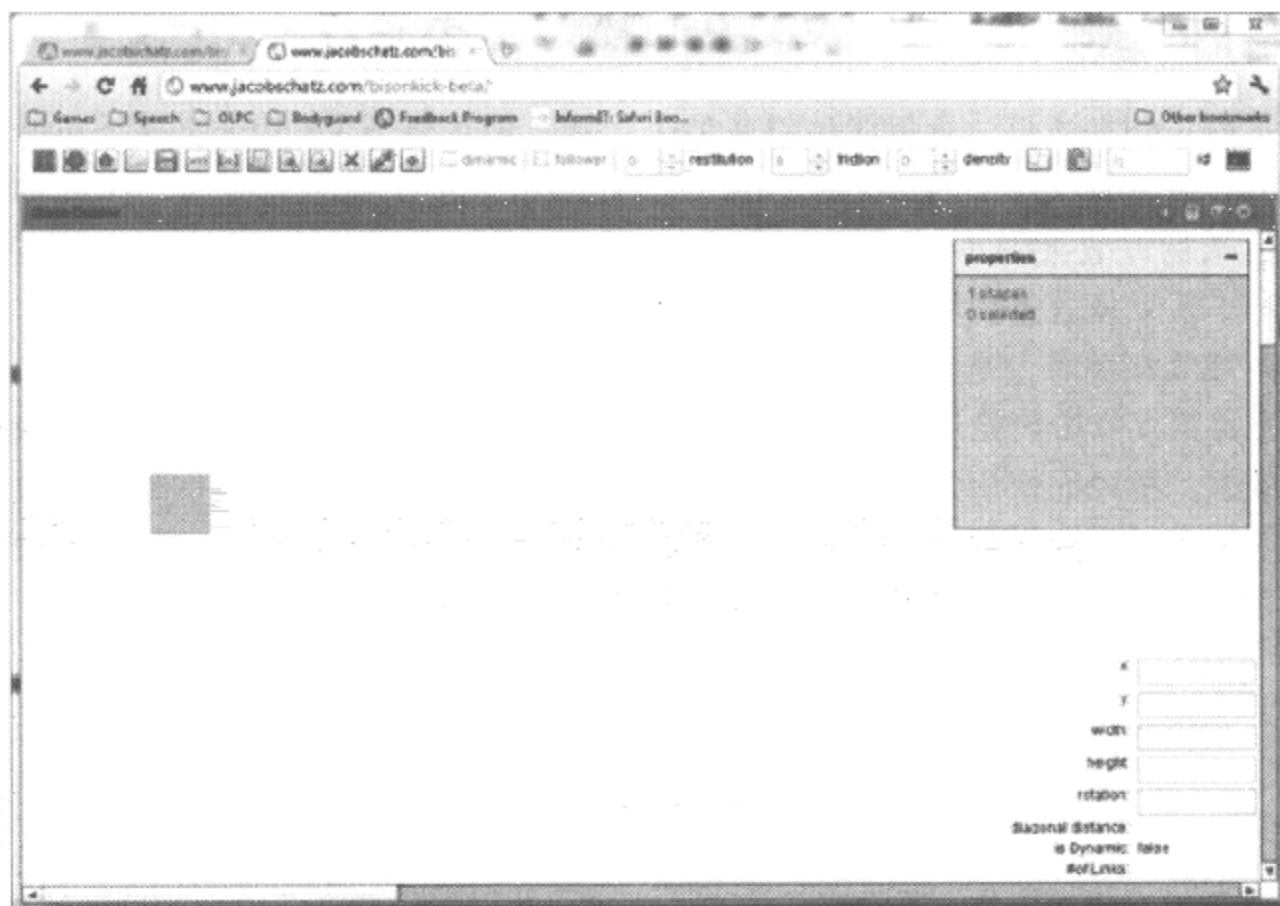


图 12.5 Bison Kick 编辑器初始界面

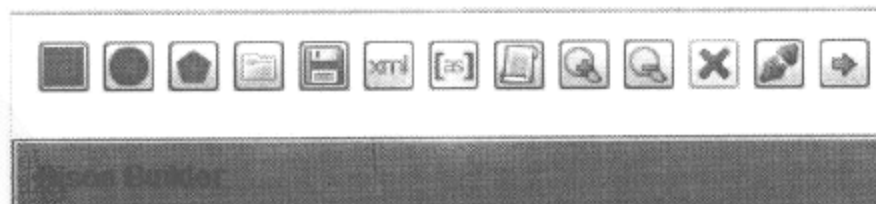


图 12.6 Bison Kick 编辑器左侧的工具栏

- 可以随时保存当前编辑的关卡。第 4 个图标可用于打开已有的关卡文件，第 5 个则用于保存当前关卡。
- 第 6 个图标中写有“xml”字样，点击它之后，编辑器将会以 XML 格式显示当前编辑的关卡，我们将使用此工具将当前关卡内容提取至游戏项目中。
- 接下来的两个按钮可以忽略，它们用于创建 Flash 版本（ActionScript）和 Box2D 的关卡。
- 两个放大镜图标分别用于放大及缩小当前关卡。
- 红叉图标用于将当前选中的物体从绘制区域中删去。
- 连接器图标可以用于连接物体，可惜这种连接关系并未反映到 XML 文件中，不过使用该工具可以对正在编辑的关卡在游戏运行时的真实效果有一个更加直观的印象。
- 右箭头图标用于启动对当前绘制区域所进行的物理模拟，模拟效果会持续显示，直至用户按下模拟画面右上角的“X”按钮。

接下来讲一下工具条右方各个控件的功能，读者可以在图 12.5 中看到它们：

- 如果选中“dynamic”复选框，那么当前物体将会变为动态物体，否则，它就是静止（static）的，该版本的 Bison Kick 尚未提供创建运动（kinematic）物体的功能。
- 该复选框右侧还有一个复选框，可以将物体标注为“follower”，这样的话，在进行物理模拟时，摄像头就会尽可能地跟随它。AndEngine 尚未支持此功能。
- 接下来的 3 个微调按钮旁边分别写有“restitution”（弹性）、“friction”（摩擦系数）及“density”（密度）字样，利用它们可以设定当前物体所用固定器的对应属性。“restitution”属性等价于 Box2D 中的“elasticity”属性。
- 接下来的两个图标用于复制与粘贴物体。
- 标有“id”的文本框可以用于设定当前物体的识别码，本游戏将利用这个属性来判断物体的材质，看它是木头、石头还是玻璃。石头是不能被打碎的，如果用力足够大，木桩就可以被打断，而玻璃则一碰就碎。吸血鬼头像以一个识别码为“vamp”的圆形来表示。
- 最右边的那个按钮是为 Flash 开发者提供的。

使用 Bison Kick 编辑完成的关卡如图 12.7 所示。

现在点击 XML 图标，将当前关卡内容以 XML 文件形式显示到屏幕上，如图 12.8 所示。用户必须手动选中窗口内的所有文字，并按下键盘上的 Ctrl+C 组合键将其复制。然后就可以将其粘贴到自己爱用的文本编辑器中，并已将其保存为游戏所需的 XML 文件。

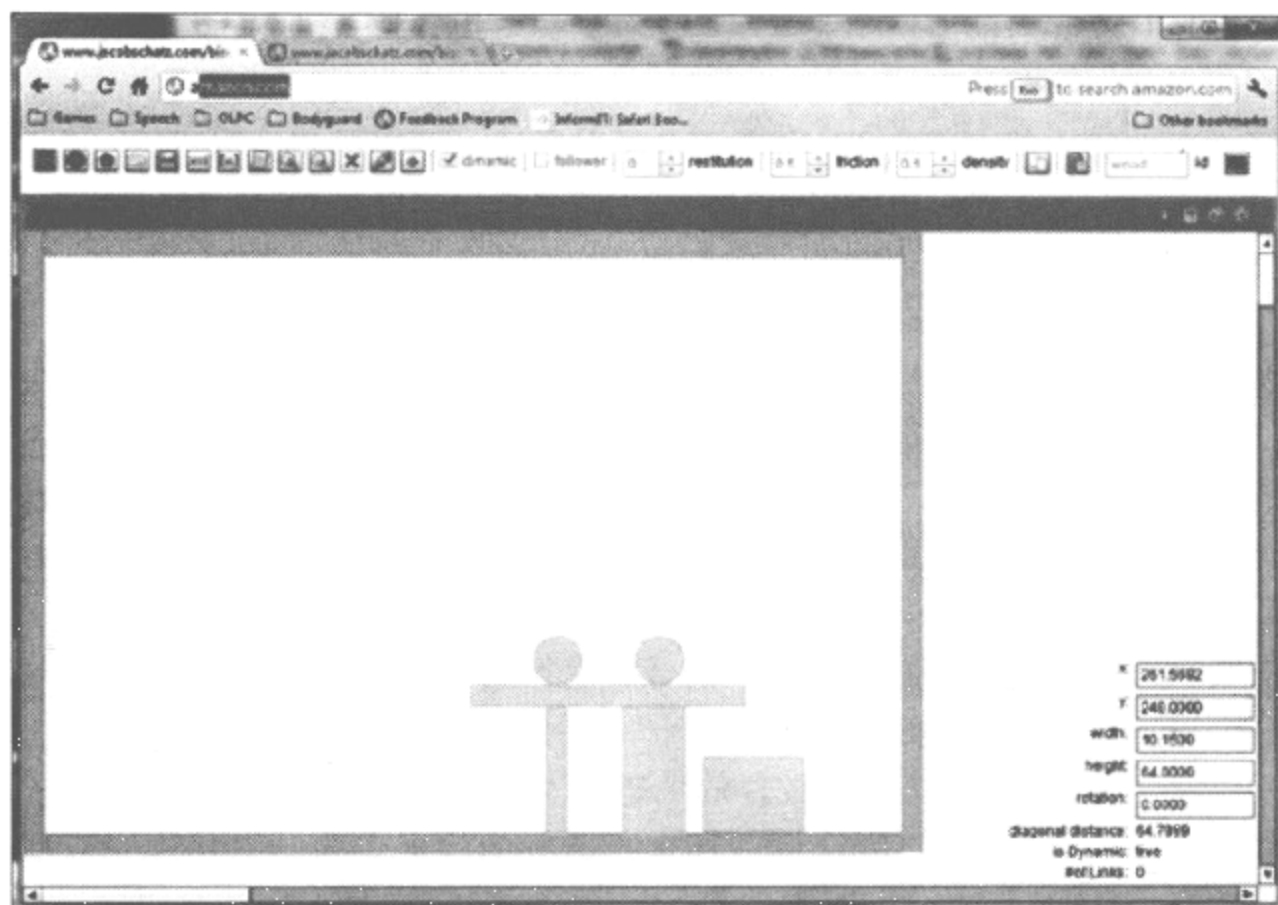


图 12.7 使用 Bison Kick 软件编辑完成的关卡



图 12.8 Bison Kick 软件中用于复制 XML 内容的窗口

程序清单 12.3 列出了关卡文件开头部分的内容。

程序清单 12.3 IV 小游戏所用 XML 格式的关卡文件

---

```

<level>
  <completeShape>
    <xprop>15.0000</xprop>
    <yprop>272.5000</yprop>
    <height>525.00</height>
    <width>10.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>false</isDynamic>
    <shape>SQUARE</shape>
    <physicsandID>0.5,0.5,0.5,'stone'</physicsandID>
    <verts>'()'</verts>
  </completeShape>
  <completeShape>
    <xprop>423.0000</xprop>
    <yprop>533.0000</yprop>
    <height>10.00</height>
    <width>802.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>false</isDynamic>
    <shape>SQUARE</shape>
    <physicsandID>0.5,0.5,0.5,'stone'</physicsandID>
    <verts>'()'</verts>
  </completeShape>
  . . .
</level>

```

---

请注意，上述文件中的所有内容都是以实体的方式罗列出来的，LevelLoader 无法解析这种格式，必须像程序清单 12.4 一样，将属性附在其对应实体的后面，这样 LevelLoader 才能正确解析它。

程序清单 12.4 带有属性值的 XML 格式关卡文件

---

```

<level>
  <completeShape>
    xprop=15.0000
    yprop=272.5000
    height=525.00
    width=10.00
    rotation=0.0000
    isDynamic='false'
    shape=SQUARE
    physicsandID='0.5,0.5,0.5,stone'
    verts='()'
  </completeShape>
  <completeShape>
    xprop=423.0000

```

---



```

        yprop=533.0000
        height=10.00
        width=802.00
        rotation=0.0000
        isDynamic='false'
        shape=SQUARE
        physicsandID='0.5,0.5,0.5,stone'
        verts='()'
    </completeShape>
    . . .
</level>

```

不过我们并不想把 Bison Kick 编辑器做好的文件都逐个按此修改，而是希望直接解析程序清单 12.3 这种 Bison Kick 所生成的关卡。但不巧的是，LevelLoader 又不能直接访问 XML 实体中除属性值之外的部分<sup>①</sup>，不过没关系，可以专门为 Bison Kick 文件格式创建一款名为 BKLoader 的文件加载器。它的使用方式同 LevelLoader 一样，不过 onLoadEntity() 方法还会顺带着返回被载入实体中的文本内容。此外还需做出一处修改，使解析器读至实体末端时才回调 onLoadEntity() 方法，而不要像现在这样，一发现新实体就立即回调。比方说，修改后的 onLoadEntity() 方法会通知我们，<completeShape> 标记已被解析好了，这样开发者就可以访问其中所有参数的内容了。

BKLoader 类继承了 LevelLoader，并且实现了一个新的接口类，IBKEntityLoader，而这个接口又继承自 IEntityLoader 接口。在增加与覆写适当的方法之后，BKLoader 就可以返回所需的实体文本了。此外还需要一个继承自 LevelParser 类的 BKParser 类，它和 BKLoader 类一样，也需要增加与覆写一些必要的方法。若想一探究竟，可以查看本章所附的源代码。

### 12.5.2 编写 IMainActivity.java

这里将对 OptionsActivity.java 的修改略过，因为它与第 9 章所讲的将 Whack-A-Vampire 小游戏加入选项菜单所用方法是一致的，如果读者想看看具体内容，可以下载本章所附源代码。

IMainActivity.java 的代码则更有趣些，程序清单 12.5 将被分成 7 部分，每部分代码之后都会解释这段代码所做的关键修改。

程序清单 12.5 IMainActivity.java 文件第 1 部分

```

package com.pearson.lagp.v3;

+imports. . .

```

① 比如在“<xprop>15.0000</xprop>”这个实体中，“15.0000”这部分文本内容无法经由 LevelLoader 直接访问。——译者注

```

public class IVActivity extends BaseGameActivity implements
    IOnSceneTouchListener, BKConstants {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    private static final FixtureDef FIXTURE_DEF =
        PhysicsFactory.createFixtureDef(1, 0.5f, 0.5f);

    // =====
    // 字段
    // =====

    private BuildableTexture mTexture;

    private TextureRegion mStakeTextureRegion;
    private TextureRegion mGlassTextureRegion;
    private TextureRegion mWoodTextureRegion;
    private TextureRegion mStoneTextureRegion;
    private TextureRegion mMatHeadTextureRegion;
    private Sprite stakesprite;
    private Scene mScene;

    private PhysicsWorld mPhysicsWorld;

    private boolean isStakeSpawning = false;
    private float stakeX, stakeY;
    private float velX, velY;
    private Line stakeLine;

    private Vector2 gravity;
    private Body stake;

    private float mX, mY;
    private float mWidth, mHeight;
    private float mRotation;
    private boolean mIsDynamic;
    private BodyType mBodyType;
    private String mShape;
    private String mPhysicsAndID;
    private float mDensity;
    private float mFriction;
    private float mElasticity;
    private String mID;
    private String mVerts;

    private float PtoM =
        PhysicsConstants.PIXEL_TO_METER_RATIO_DEFAULT;

```

第1部分没有多少新代码，只是增加了一些需要用到的变量和一个常数。

#### 程序清单 12.5 IVActivity.java 文件第2部分

```
@Override
public Engine onLoadEngine() {
    final Camera camera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    final EngineOptions engineOptions = new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
        CAMERA_HEIGHT), camera).setNeedsSound(true);
    engineOptions.getTouchOptions().setRunOnUpdateThread(true);
    return new Engine(engineOptions);
}

@Override
public void onLoadResources() {
    /* Textures. */
    this.mTexture = new BuildableTexture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    TextureRegionFactory.setAssetBasePath("gfx/IV/");

    /* TextureRegions. */
    this.mStakeTextureRegion =
        TextureRegionFactory.createFromAsset(this.mTexture,
            getApplicationContext(), "stake40.png");
    this.mGlassTextureRegion =
        TextureRegionFactory.createFromAsset(this.mTexture,
            getApplicationContext(), "glass.png");
    this.mWoodTextureRegion =
        TextureRegionFactory.createFromAsset(this.mTexture,
            getApplicationContext(), "wood.png");
    this.mStoneTextureRegion =
        TextureRegionFactory.createFromAsset(this.mTexture,
            getApplicationContext(), "stone.png");
    this.mMatHeadTextureRegion =
        TextureRegionFactory.createFromAsset(this.mTexture,
            getApplicationContext(), "mathead.png");
    try {
        mTexture.build(
            new BlackPawnTextureBuilder(2));
    } catch (final TextureSourcePackingException e) {
        Log.d("V3",
            "Sprites won't fit in mTexture");
    }
    this.mEngine.getTextureManager().loadTexture(this.mTexture);
}
```

第2部分的代码现在看起来应该比较熟悉了吧？这段代码创建了 BuildableTexture 对象并将创建不同物体所需的 TextureRegion 对象载入其中。

#### 程序清单 12.5 IMainActivity.java 文件第3部分

```
@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    mScene = new Scene(2);
    mScene.setOnSceneTouchListener(this);

    /* 将摄像机置于中央。 */
    final int centerX = CAMERA_WIDTH / 2;
    final int centerY = CAMERA_HEIGHT / 2;
    mScene.setBackground(new ColorBackground(0.0f, 0.0f,
        0.0f));
    mPhysicsWorld = new PhysicsWorld(new Vector2(0,
        SensorManager.GRAVITY_EARTH), false);

    final BKLoader bkLoader = new BKLoader();
    bkLoader.setAssetBasePath("level/iv/");
```

第3部分的代码开始变得有趣了，它先创建 BKLoader 对象，以便接下来可以向它注册各种需要监听的实体。每当 BKLoader 对象找到注册过的实体时，它就会回调注册时所用 IBKEntityLoader 对象的 onLoadEntity() 方法。诸如 TAG\_LEVEL、TAG\_BODY 这样与 XML 标记有关的常数都可以在 BKConstants.java 文件中找到。

#### 程序清单 12.5 IMainActivity.java 第4部分

```
bkLoader.registerEntityLoader(TAG_LEVEL,
    new IBKEntityLoader() {
        @Override
        public void onLoadEntity(final String pEntityName,
            final Attributes pAttributes,
            final String pValue) {
        }
        public void onLoadEntity(final String pEntityName,
            final Attributes pAttributes) {
        }
    });

bkLoader.registerEntityLoader(TAG_BODY,
    new IBKEntityLoader() {
        @Override
        public void onLoadEntity(final String pEntityName,
            final Attributes pAttributes,
            final String pValue) {
```

```

        if (mShape.equals(TAG_SHAPE_VALUE_SQUARE)) {
            final TextureRegion mTR = selectTexture(mID);
            final Sprite bodyShape = new Sprite(mX - mTR.getWidth()/2,
                mY - mTR.getHeight()/2, mTR);
            bodyShape.setScaleX(mWidth/mTR.getWidth());
            bodyShape.setScaleY(mHeight/mTR.getHeight());
            if (mRotation != 0.0f) {
                bodyShape.setRotation(mRotation);
            }
            final Body mBody =
                PhysicsFactory.createBoxBody(mPhysicsWorld,
                    bodyShape, mBodyType,
                        PhysicsFactory.createFixtureDef(mDensity,
                            mElasticity, mFriction));
            mScene.getLastChild().attachChild(bodyShape);
            mPhysicsWorld.registerPhysicsConnector(
                new PhysicsConnector(bodyShape, mBody, true,
                    true));
        } else if (mShape.equals(TAG_SHAPE_VALUE_CIRCLE)) {
            final TextureRegion mTR = mMatHeadTextureRegion;
            Sprite bodyShape = new Sprite(mX - mTR.getWidth()/2,
                mY - mTR.getHeight()/2, mTR);
            bodyShape.setScaleX(mWidth/mTR.getWidth());
            bodyShape.setScaleY(mHeight/mTR.getHeight());
            if (mRotation != 0.0f) {
                bodyShape.setRotation(mRotation);
            }
            final Body mBody =
                PhysicsFactory.createCircleBody(mPhysicsWorld,
                    bodyShape, mBodyType,
                        PhysicsFactory.createFixtureDef(mDensity,
                            mElasticity, mFriction));
            mScene.getFirstChild().attachChild(bodyShape);
            mPhysicsWorld.registerPhysicsConnector(
                new PhysicsConnector(bodyShape, mBody, true,
                    true));
        } else if (mShape.equals(TAG_SHAPE_VALUE_POLYGON)) {
            // 未实现
        } else {
            throw new IllegalArgumentException();
        }
    }

    public void onLoadEntity(final String pEntityName,
        final Attributes pAttributes) {
    }
}

```

与 TAG\_BODY 相关的那个 onLoadEntity() 方法是该部分代码的重点, BKLoader 会在载入了包含结束标记的完整 completeShape 定义之后, 回调该方法。此时所有创建

物体所需的参数都可以获取到了，所以代码在此处创建用于组合游戏场景的物体与固定器。其详细步骤可以分解如下：

- 首先判断物体是矩形（SQUARE）还是圆形，这段代码不支持多边形物体，因为它过于复杂了，况且并没有用于创建多边形的 Shape 类。如果确实想要用多边形，也要准备好每种多边形所用的贴图。
- 如果某物体是矩形，那么将会根据其 ID 来选择合适的 TextureRegion 对象。通过 ID 可以分辨出它究竟是木头、石头还是玻璃。selectTexture() 方法的代码将会在稍后给出。如果某物体是圆形，则非常简单，我们使用 Mad Mat 的头的贴图。
- 根据从 XML 文件中读取的宽度、高度、旋转角度等参数来缩放并旋转 Sprite 对象。
- 以 XML 文件中读取的信息为参数创建固定器，并用它创建游戏所需的物体。
- 将 Sprite 对象作为子对象加入到当前 Scene 对象中，并用 PhysicsConnector 对象将 Sprite 对象与刚创建的物体关联起来。
- 最后，为了符合 IBKEntityLoader 接口规范，代码定义了一个没有 pValue 参数的 onLoadEntity 方法，其中没有任何代码。

#### 程序清单 12.5 IActivity.java 文件第 5 部分

```
bkLoader.registerEntityLoader(TAG_X, new IBKEntityLoader() {
    @Override
    public void onLoadEntity(final String pEntityName,
        final Attributes pAttributes, final String pValue) {
        mX = new Float(pValue);
    }
    public void onLoadEntity(final String pEntityName,
        final Attributes pAttributes) {
    }
});

bkLoader.registerEntityLoader(TAG_Y, new IBKEntityLoader() {
    @Override
    public void onLoadEntity(final String pEntityName,
        final Attributes pAttributes, final String pValue) {
        mY = new Float(pValue);
    }
    public void onLoadEntity(final String pEntityName,
        final Attributes pAttributes) {
    }
});

bkLoader.registerEntityLoader(TAG_WIDTH, new IBKEntityLoader() {
    @Override
    public void onLoadEntity(final String pEntityName,
```



```

    final Attributes pAttributes, final String pValue) {
        mWidth = new Float(pValue);
    }
    public void onLoadEntity(final String pEntityName,
    final Attributes pAttributes) {
    }
    });
    bkLoader.registerEntityLoader(TAG_HEIGHT, new IBKEntityLoader() {
        @Override
        public void onLoadEntity(final String pEntityName,
    final Attributes pAttributes, final String pValue) {
            mHeight = new Float(pValue);
        }
        public void onLoadEntity(final String pEntityName,
    final Attributes pAttributes) {
        }
    });
    bkLoader.registerEntityLoader(TAG_ROTATION, new IBKEntityLoader() {
        @Override
        public void onLoadEntity(final String pEntityName,
    final Attributes pAttributes, final String pValue) {
            mRotation = new Float(pValue);
        }
        public void onLoadEntity(final String pEntityName,
    final Attributes pAttributes) {
        }
    });
    bkLoader.registerEntityLoader(TAG_ISDYNAMIC, new IBKEntityLoader() {
        @Override
        public void onLoadEntity(final String pEntityName,
    final Attributes pAttributes, final String pValue) {
            mIsDynamic = true;
            if (pValue.equals("false")) mIsDynamic = false;
            mBodyType = BodyType.StaticBody;
            if (mIsDynamic) mBodyType = BodyType.DynamicBody;
        }
        public void onLoadEntity(final String pEntityName,
    final Attributes pAttributes) {
        }
    });
    bkLoader.registerEntityLoader(TAG_SHAPE, new IBKEntityLoader() {
        @Override
        public void onLoadEntity(final String pEntityName,
    final Attributes pAttributes, final String pValue) {

```

```

        mShape = pValue;
    }
    public void onLoadEntity(final String pEntityName,
final Attributes pAttributes) {
    }
});
bkLoader.registerEntityLoader(TAG_PHYSICSANDID,
new IBKEntityLoader() {
    @Override
    public void onLoadEntity(final String pEntityName,
final Attributes pAttributes, final String pValue) {
        mPhysicsAndID = pValue;
        final String[] physTokens = mPhysicsAndID.split(",");
        mDensity = Float.valueOf(physTokens[1]).floatValue();
        mFriction = Float.valueOf(physTokens[0]).floatValue();
        mElasticity = Float.valueOf(physTokens[2]).floatValue();
        mID = trimQuotes(physTokens[3]);
    }
    public void onLoadEntity(final String pEntityName,
final Attributes pAttributes) {
    }
});
bkLoader.registerEntityLoader(TAG_VERTS, new IBKEntityLoader() {
    @Override
    public void onLoadEntity(final String pEntityName,
final Attributes pAttributes, final String pValue) {
    }
    public void onLoadEntity(final String pEntityName,
final Attributes pAttributes) {
    }
});

try {
    bkLoader.loadLevelFromAsset(getApplicationContext(), "iv1.lv1");
} catch (final IOException e) {
    Debug.e(e);
}

mScene.registerUpdateHandler(this.mPhysicsWorld);
return mScene;
}

@Override
public void onLoadComplete() {
    this.mPhysicsWorld.setContactListener(new ContactListener() {
        @Override

```

```

        public void beginContact(Contact contact) {
        }
        public void endContact(Contact contact) {
        }
    });
}

```

第 5 部分的各种 onLoadEntity() 方法都是用于从 XML 文件中提取参数值的, 其中某些参数, 比如 PhysicsAndID, 需要对 XML 中的字符串再次进行解析。

#### 程序清单 12.5 IVActivity.java 第 6 部分

```

@Override
public boolean onSceneTouchEvent(final Scene pScene,
    final TouchEvent pSceneTouchEvent) {
    if(this.mPhysicsWorld != null) {
        final Scene scene = this.mEngine.getScene();
        if(pSceneTouchEvent.isActionDown()) {
            isStakeSpawning = true;
            stakeX = pSceneTouchEvent.getX();
            stakeY = pSceneTouchEvent.getY();
            return true;
        }
        if (pSceneTouchEvent.isActionMove()) {
            if (isStakeSpawning) {
                // 绘制发射小木桩时所用的瞄准线
                if (stakeLine == null) {
                    stakeLine = new Line(pSceneTouchEvent.getX(),
                        pSceneTouchEvent.getY(), stakeX, stakeY);
                } else {
                    stakeLine.setPosition(pSceneTouchEvent.getX(),
                        pSceneTouchEvent.getY(), stakeX, stakeY);
                }
                scene.getLastChild().attachChild(stakeLine);
                return true;
            }
        }
        if (pSceneTouchEvent.isActionUp()) {
            // 发射小木桩
            velX = (stakeX - pSceneTouchEvent.getX())/6.0f;
            velY = (stakeY - pSceneTouchEvent.getY())/6.0f;
            this.addStake(stakeX, stakeY, velX, velY);
            if (stakeLine != null) stakeLine.setPosition(0.0f, 0.0f,
                0.0f, 0.0f);
        }
    }
}

```

```

        isStakeSpawning = false;
        return true;
    }
}
return false;
}

```

第6部分的代码在检测到屏幕触摸事件之后，就会创建出玩家用弹弓所发射的小木桩。这段代码很直白：首先确定玩家触摸的起止位置，算出 ACTION\_DOWN 与 ACTION\_UP 事件发生点之间的距离，按照一定的比例缩放之后，再根据方向创建速度向量。

#### 程序清单 12.5 IMainActivity.java 文件第7部分

```

// =====
// 方法
// =====

private void addStake(final float pX, final float pY, float velX,
    float velY) {
    final Scene scene = this.mEngine.getScene();
    stakesprite = new Sprite(pX, pY, this.mStakeTextureRegion);
    stakesprite.registerEntityModifier(new RotationModifier(0.1f, 0.0f,
        (float) ((360.0f/Math.PI)*Math.atan(velY/velX))));
    stake = PhysicsFactory.createBoxBody(this.mPhysicsWorld,
    stakesprite, BodyType.DynamicBody, FIXTURE_DEF);
    stake.setBullet(true);
    stake.setLinearVelocity(new Vector2(velX, velY));
    stake.setSleepingAllowed(true);

    scene.getLastChild().attachChild(stakesprite);
    this.mPhysicsWorld.registerPhysicsConnector(
    new PhysicsConnector(stakesprite, stake, true, true));
}

private TextureRegion selectTexture(String id){
    if (id.equals("wood")){
        return mWoodTextureRegion;
    } else if (id.equals("stone")){
        return mStoneTextureRegion;
    } else {
        return mGlassTextureRegion;
    }
}

public static String trimQuotes(String value)
{
    if (value == null)

```

```

        return value;

        value = value.trim();
        if (value.startsWith("\'") && value.endsWith("\'"))
            return value.substring(1, value.length() - 1);
        return value;
    }
}

```

第 7 部分的代码中定义了一些用于简化运算所用的方法，这些代码也很直白易懂。

## 12.6 总结

本章涉及的内容十分广泛，包括了游戏物理学与关卡加载。即便用了这么长的篇幅，也才讲了游戏物理学的皮毛而已，还存在许多有待研究的话题，例如：

- 各种类型的关节：
  - 距离关节 (distance joint)
  - 转动关节 (revolute joint)
  - 棱柱关节 (prismatic joint)
  - 滑轮关节 (pulley joint)
  - 齿轮关节 (gear joint)
  - 鼠标关节 (mouse joint)
  - 线段关节 (line joint)
  - 焊接关节 (weld joint)
- 碰撞检测过滤 (collision filtering)
- 固定步长 (fixed-step) 物理模拟
- 阻尼<sup>⊖</sup> (damping)
- 刚体绕定轴转动 (fixed rotation)
- 活化作用 (activation)
- 用户数据 (user data)

要研究这些功能，最好从 Box2D 的用户手册开始。

## 12.7 习题

1. 使用 Bison Kick 或自己喜欢的关卡编辑器给 IV 小游戏再设计一个关卡，作为其

⊖ 阻尼是指在任何振动系统中，由于外界作用（如流体阻力、摩擦力等）或系统本身固有的原因引起的振动幅度逐渐下降的特性，以及此特性的量化表征。——译者注

第2关。以新的关卡文件替换 `lv11` 文件，并运行游戏。调整密度、摩擦系数、弹性等参数直到关卡设计得比较合理为止。

2. IV 游戏可以让用户在任意地点发射小木桩。现在修改 `IVActivity.java` 文件，在屏幕左方创建一个用于表示弹弓的 `Sprite` 对象并令其静止不动，让玩家只能从弹弓附近的点发射小木桩。





- 13.1 游戏 AI 相关话题
- 13.2 实现 V3 游戏的 AI
- 13.3 总结
- 13.4 习题

如果玩家必须经过一番思考才能战胜对手,那么游戏会变得更有趣些。这里所说的对手,可以是多人游戏中的另外一名玩家,也可以是由开发者所设计的、会执行其指令的游戏角色。在游戏领域,后面这种方式通常叫做人工智能(Artificial Intelligence, AI),它可以让游戏更加好玩儿。

可以将游戏中的AI和研究领域的AI人为地划分开来,不过随着游戏的制作,这种界限会越来越模糊。AI研究的目标是用于减少人类操作电脑所费的脑力,然而游戏AI的目标则是让游戏更具可玩性,通常表现为游戏角色所执行的一些简单策略,不过即使是在最前沿的AI领域,游戏AI与科研AI还是可以结合起来的。比如IBM所研制的Watson系统<sup>①</sup>,它的推测能力在电视娱乐节目《Jeopardy!》<sup>②</sup>中得到了充分的展现。

本章将会讨论一些游戏制作中流行的AI策略与算法。虽说AndEngine并未提供特别的AI支持,但是我们可以用Java语言在V3范例游戏中实现本章所讲的其中一种AI策略。

## 13.1 游戏AI相关话题

永远不可能将所有AI算法与策略都一一列举出来,因为在每一天的游戏设计中,都会有新的AI算法发明。本节将会讨论几种经典的AI策略,以及它们在不同类型游戏中的用法。

### 13.1.1 简单的脚本

最简单的、使用最广的游戏AI其实非常不智能,它仅仅通过一个或多个脚本定义了游戏角色的行为而已。这种脚本可以命令游戏角色移动到何处、发出何种声音等,不过这些动作都是被有限的几个事件所激发的固定行为。迄今为止V3游戏中的行为都属于上述类型,虽说这些动作也可以用脚本语言来写,不过本书还是直接以Java语言代码的方式来实现的。

### 13.1.2 决策树、Minimax树与状态机

更高级一些的人工智能则使用更为复杂的软件算法,这些算法会根据游戏角色当前

① 沃森(英文名Watson)是能够使用自然语言来回答问题的人工智能系统,由IBM公司的首席研究员David Ferrucci所领导的DeepQA计划小组开发并以该公司创始人托马斯·J·沃森的名字命名。——译者注

② 《危险边缘》(英文名Jeopardy!),美国的电视智力竞赛节目,比赛问题内容涵盖了历史、文学、艺术、流行文化、科技、体育、地理、文字游戏等多方面。比赛采取一种独特的问答形式:参赛者需根据以答案形式提供的各种线索,以问题的形式做出正确的回答。沃森曾经在节目中打败了最高奖金得主布拉德·鲁特尔和连胜纪录保持者肯·詹宁斯。——译者注

及早先的状态做出更为复杂也更为有趣的决策。如果读者对此类算法不太熟悉，请看以下的简要说明。

### 决策树

在决策树中，算法将会根据一系列决策标准而找出最终的答案。在图表中，决策标准使用节点表示，而最终的决策则以叶节点的形式表示。整个运算过程很像故障诊断图（troubleshooting chart）或者生物检索表（biology identification key）。图 13.1 是一个假想的游戏决策树。

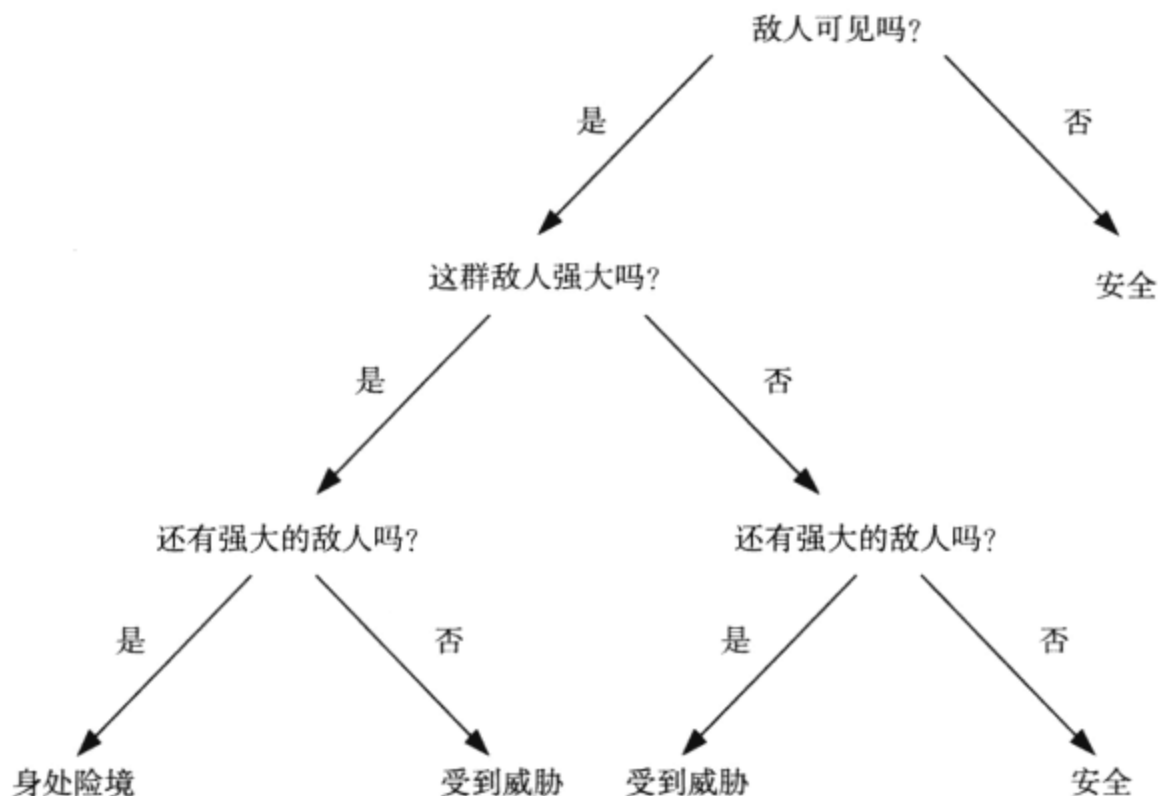


图 13.1 简单的决策树

如果决策树中的决策标准都是二选一的，那么用一系列简单的 if-else 语句就可以实现了，然而，如果决策标准是多选一的，那么就需要用一系列复杂一些的 switch 语句来实现了。

### Minimax 树

Minimax 树<sup>①</sup>通常用于在诸如棋类游戏这种零和博弈（zero-sum game）中决定电脑选手的最优棋步。换句话说，它用于帮助电脑棋手从决策图上所有可以落子的地点中选出最优的走棋步骤。图 13.2 描述了 Minimax 树的一部分，它显示了 Tic-Tac-Toe 游戏<sup>②</sup>（英式英语叫做 Naughts and Crosses）中的最初几步棋。该树的一层对应于棋手

① 中文名为极小化极大算法树。——译者注

② 井字棋（Tic-Tac-Toe），玩法是两个玩家，一个打圈，一个打叉，轮流在 3×3 的格上打自己的符号，最先以横、直、斜连成一线则为胜。如果双方都下得正确无误，将得和局。——译者注



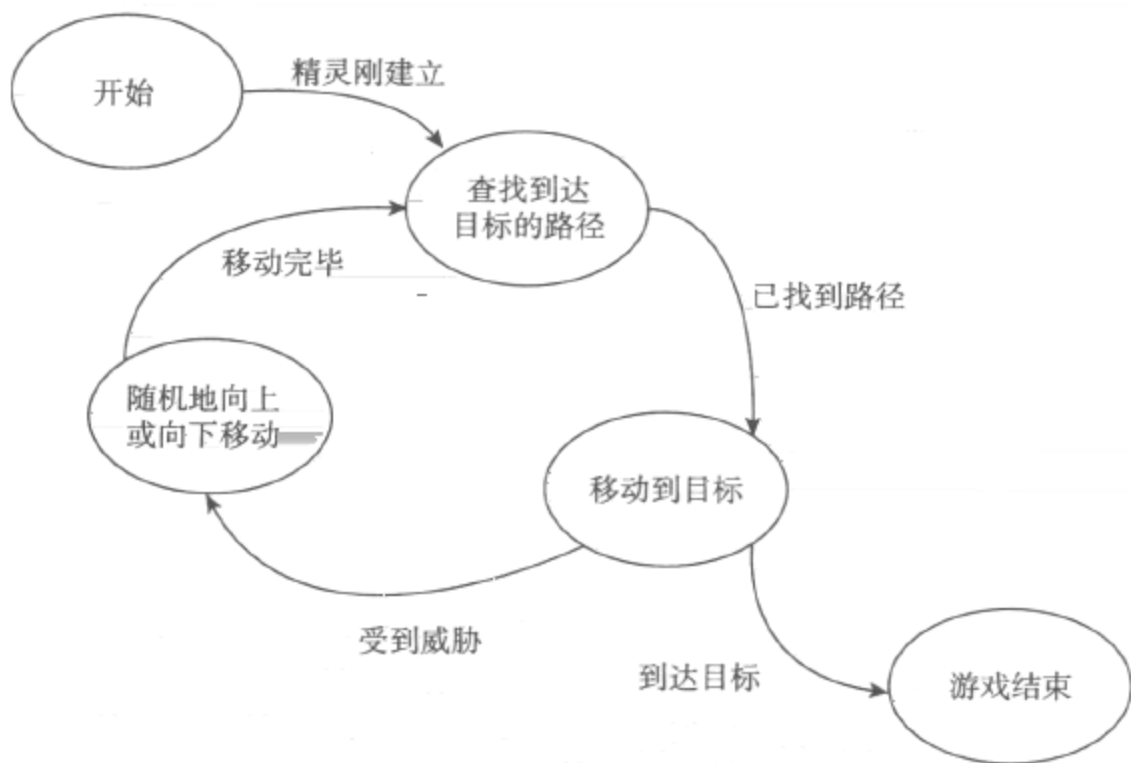


图 13.3 为吸血鬼设计的状态机

### 13.1.3 专家系统或基于规则的决策系统

在偏向学术研究的人工智能领域中，专家系统会从某个专业玩家的经验中总结出一套以“如果……那么……”（if-then）为形式的规则，用以命令计算机依此行事或据此推算出最优策略。这种基于规则的决策系统通常由如下要素组成：

- ❑ 以基于规则的语言所编写的一套规则代码，其中每条代码都含有一个条件（或谓词）与符合条件之后需要执行的动作。
- ❑ 一个工作存储器（working memory），用于保存判断规则的状态，并且可以修改。
- ❑ 一个推断引擎，会遍历每一条规则，并且决定依照哪条来执行。通常每次运算只会根据其中的一项规则来执行，如果有许多规则都满足适用条件，那么将需要另外的算法来在其中选定最终适用的那条。
- ❑ 一个用于向决策系统推送事件与数据，并获取最终决策的接口。

当然也可以使用蛮力法（brute force），用 if-then-else 语句写代码来达成同样的效果。不过这么做会导致低效、混乱且难于维护的代码。基于规则的系统则提供了一种更加清晰高效的结构，用于解析规则集，基于规则的语言是“陈述式”（declarative）而非“命令式”（imperative）的。换句话说，代码书写者需要告诉规则引擎当前的状况以及适用的规则，这样它就会将这些因素合并起来，从而得出最终的决策。

某些基于规则的系统（例如 OPS5 和 JESS）采用演绎法（forward-chaining），其形式为：“如果符合某条件，将采取何种行动”，而另外一些（例如 Prolog）则采用回纳法（backward-chaining），其形式为“这是真的吗？”大多数新的系统都会支持这两种方

式，还有一些规则系统可以在创立规则的过程中动态地学习推演新的规则。

基于规则系统的电脑游戏有段时间曾非常流行，不过最近这个领域的研究多致力于业务规则。名为 JSR-94 的 Java 规范即是为这样的系统所制定的，并且有很多 Java 语言的实现版本可供使用，例如 JESS 以及 Drools（也就是 JBoss Rule）。

程序清单 13.1 展示了如何使用基于规则的语言。如果确定要使用规则系统的话，需要学习完整的语言语法，并且将这套系统与 Android 系统相连。如果的确需要实现某种复杂的推理论证，那么这份努力也许是值得的。

程序清单 13.1 Prolog 语言范例

---

```
. . . Prolog . . .
Vampire(Igor).
Ghoul(X) :- Vampire(X).
?- Ghoul(Igor).
Yes
?- Ghoul(X).
X = Igor
```

---

这几段 Prolog 代码的意思是：

- ☐ Igor 是一只吸血鬼。
- ☐ 所有吸血鬼都是食尸鬼（Ghoul）。
- ☐ Igor 是食尸鬼吗？Prolog 引擎打印出答案（是）。
- ☐ 食尸鬼是谁？Prolog 引擎打印出答案（Igor）。

仅仅扫一眼语法规则是远远不够的，如果要运用一门基于规则的语言，更需要做的是思考方式的转变。如果读者感兴趣，可以看看网上的教程与范例，它们会告诉你如何配置规则并用它来解决复杂的问题。

### 13.1.4 神经网络

神经网络是另一种手段，用于处理 Java 这种过程式语言难于解决的问题。这种神经网络尤其善于模式识别，其核心思想是建立一个由节点所组成的网络，用以模拟神经元行为，这也是该算法得名的原因。节点以分层的方式组织起来，并通过带有权重的箭头互相连接。当神经元兴奋时，它将把刺激传递给相连的其他节点，传递的刺激度会依照权重修改。每个神经元节点都有一个临界值（threshold），如果接收的刺激总量超过了临界值，那么该节点将会再度兴奋。

图 13.4 显示了一个简单的神经网络，它的输入（input）与输出（output）层有且仅有 1 个，不过隐藏（hidden）层的数量可以是任意的。

神经网络有趣的地方是，它不需要别人来手工指定权重与临界值，而是要以训练的方式来自己得出这些值。向它输入一些训练用的数据，然后将神经网络的输出结果同正确的输出模式比对，比对之后的差值将反馈给神经网络，让其据此调整隐藏层与输入层



的权重。神经网络训练好之后，它将非常容易识别相似的输入模式。反馈过程所需的数学算法已经完全解决了，并且会包含在神经网络软件当中。神经网络的理想大小，也就是每层中所含的节点个数，是个活跃的研究话题，不过根据经验，大多数模式识别都可以用 1 个隐藏层来完成（有些研究数据称应该是两个），每层所需的节点数应该比需要匹配的模式数小 1。

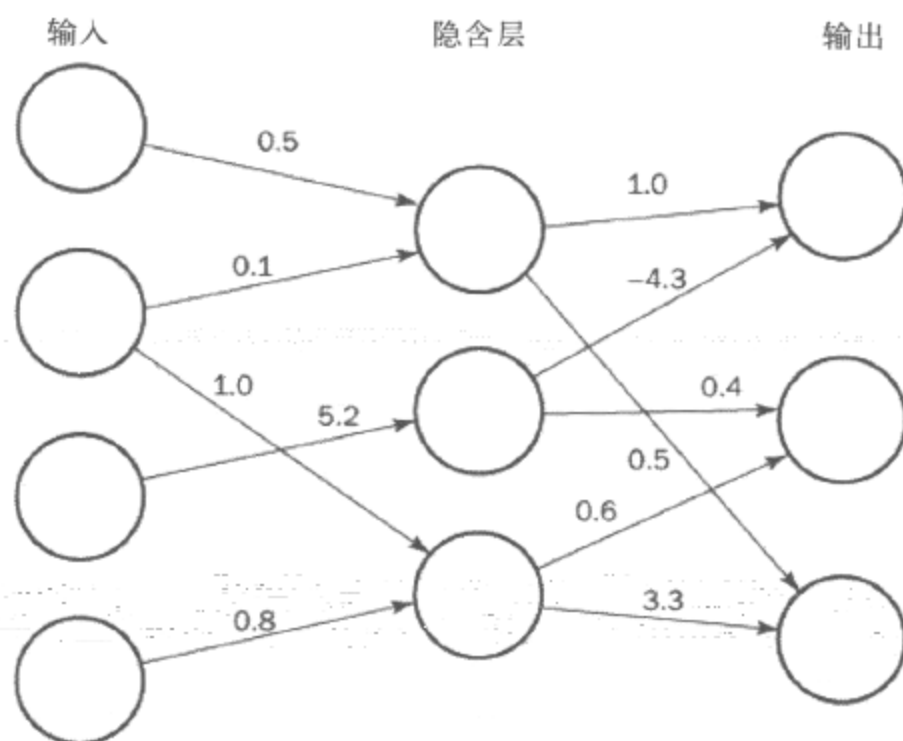


图 13.4 简单的神经元网络

笔者不知道是否有神经网络的开发库被移植到 Android 系统上了，不过网上能够找到很多优秀的参考资料和开源实现。如下两个网站可以为研究此话题指明方向：

□ Neuroph:

<http://neuroph.sourceforge.net/download.html>

□ Encog 框架:

<http://www.heatonresearch.com/encog>

### 13.1.5 遗传算法

遗传算法是另一种训练软件的方法，它也能处理过程语言难于解决的问题。更精确地说，如果某个命题空间内有针对给定问题的潜在解决方案，那么使用遗传算法就可以在该空间内根据某种标准找出“足够好”的解决方案。它的核心思想是通过模仿生物的进化，包括使用突变与自然选择的概念，来找寻所需解答。遗传算法软件通常包含如下要素。

□ 编码：该算法需要将解决方案的要素以“染色体”或者编码后的字符串形式表示出来。通常来说，这是一个二进制位串，不过也可能是更复杂的东西，例如一串整

数或一串字母。

- 自然选择：该算法需要以某种方式从每一代个体中选出最佳的配偶用以繁衍下一代。将每一代中最成功的解决方案进行“交叉配对”（crossover），我们就可以得到一个新的解决方案种群，也就是说，将每一对选定的配偶交叉配对，就可以得到下一代中的两个后代解决方案。
- 交叉配对：该算法需要使用某种技术，将两个成功配偶的染色体组合，用以创建下一代解决方案。一种简单的办法是在染色体中随机选取一点作为“交配点”。这样的后代将会继承母体 A 中的前 n 个二进制位，以及母体 B 中的剩余二进制位。另一个后代将会从母体 B 中继承前 n 个二进制位，从母体 A 中继承剩下的二进制位。
- 突变：像生物进化一样，算法在繁衍下一代的过程中也需要引入一些随机的改变。这项技术可以促进算法的搜寻，避免其陷入局部最优状态。
- 终结：搜索不需要一直执行，所以必须给出“足够好”的定义，以便在找到符合此种标准的解决方案后便停止搜索。

V3 游戏中并未使用遗传算法，而且本书也很难给出其精确描述。如果读者感兴趣，可以在网上找到许多好例子。以下网址中的动画演示了每一代解决方案的进化过程：

<http://www.obitko.com/tutorials/genetic-algorithms/example-function-minimum.php>

### 13.1.6 路径查找

路径查找是游戏中一种通常使用 AI 算法来解决的问题。通常角色要从某个指定的点走向另一个指定的点，不过会有障碍物阻挡其行进。路径查找就是要找出角色所应采取的最优路径。

A\*（A Star）寻路算法将游戏空间划分为网格，如图 13.5 所示。算法的核心概念就是找到一条由 S 到 T 的最佳路径（或者较优路径），同时避开其中的障碍物。它的过程相当简单：

- 标注出角色可以“合法”移动的格子，也就是没有障碍物的格子。
- 由 S 点开始：
  - 考虑是否可以移动到相邻的格子中（最多 8 个）。
  - 针对每个相邻的位置，计算出它的值：

$$F=G+H$$

F：该位置的值。

G：该位置与 S 的距离。垂直或水平移动一个算 1.0，斜向移动一格算 1.4。

H：一个试探值，大约为该地与 T 之间的距离。

一种常用的试探法（heuristic）名为“Manhattan”，它把仅通过垂直或水平方向移动到达 T 点所经距离当做 H 的值，在计算过程中忽略障碍物。其实此值就是该点所在

行与 T 所在行之间的距离，加上所在列与 T 所在列之间的距离。

□ 该算法接下来会要求角色移动至 F 值最低的那个位置，并重复上述过程直至到达目标，其中所经路径即是从 S 到 T 的近似最优路径（near optimal path）。之所以说“近似”，是因为所用的试探法在某些情况下找到的路径可能稍微有点儿不理想。

如果想以动画的方式观看 A\* 算法的执行，那么在维基百科（Wikipedia）的 A\* 算法页面中就能找到一个好例子（至少在本书写成时是如此，不过维基百科的内容一直在变）：

[http://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](http://en.wikipedia.org/wiki/A*_search_algorithm)

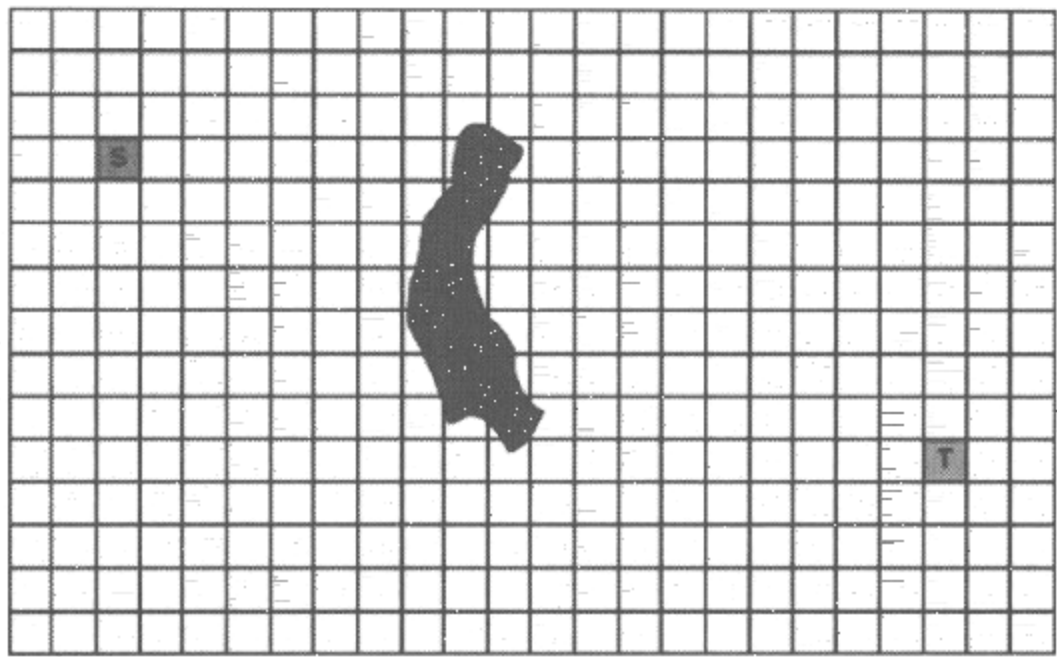


图13.5 A\*算法范例

### 13.1.7 动态困难度调节

另一个可以使用 AI 技术的地方就是根据当前玩家的能力来动态地调整游戏困难度。理想状况下，假如新手与专业级玩家都能达成与其能力相匹配的挑战，那么游戏应当对两者都进行奖励。这种挑战必须是经过努力之后可以达成的，同时又不宜过于简单。如果取得了玩家在某个时间段内（可以是较短的时间段）于给定关卡所得的分数，那么就可以把它当成代表玩家游戏水平的一个模式。上述 AI 技术可以获取该模式数据，依次调整游戏中的变量，例如敌人的数量、危险状况发生的频率、干扰玩家的次数等，这样就会使游戏更加容易或更加困难。

### 13.1.8 程序化的音乐生成

AI 研究的前沿领域之一，就是如何自动生成合适的音乐，此技术也可引入游戏中

来。这些算法非常庞大，需要耗费大量的 CPU 资源，所以可能不会在 Android 手机这种移动设备上运行，而是调用服务器端的 AI 程序，向其传入音调、气氛、拍子等参数，以此来获取合适的配乐。

如果你有可以自由支配的云端服务，那么很多想法将成为现实。大家已经看到，Google 的语音识别可以当做输入法来用，微软研究院 (Microsoft Research) 的 Songsmith 程序可以为用户唱出的任何旋律配上伴奏。程序化的音乐生成一定有办法集成到游戏中去。

## 13.2 实现 V3 游戏的 AI

现在给 V3 游戏加入一些人工智能吧。不需要大张旗鼓地加入一套基于规则的决策系统或者实现一套程序化的音乐生成系统，只需适度地加入一些 AI 就可以让游戏变得更加有趣。

看一下游戏的第 1 关，如图 13.6 所示。此时吸血鬼会从屏幕右侧的随机位置开始走进屏幕，踏过墓碑，朝着 Miss B 的屋子前进。而且，当玩家使用子弹、斧头、十字架等武器的时候，吸血鬼还是傻傻地朝着 Miss B 家门口走，不知道闪避。

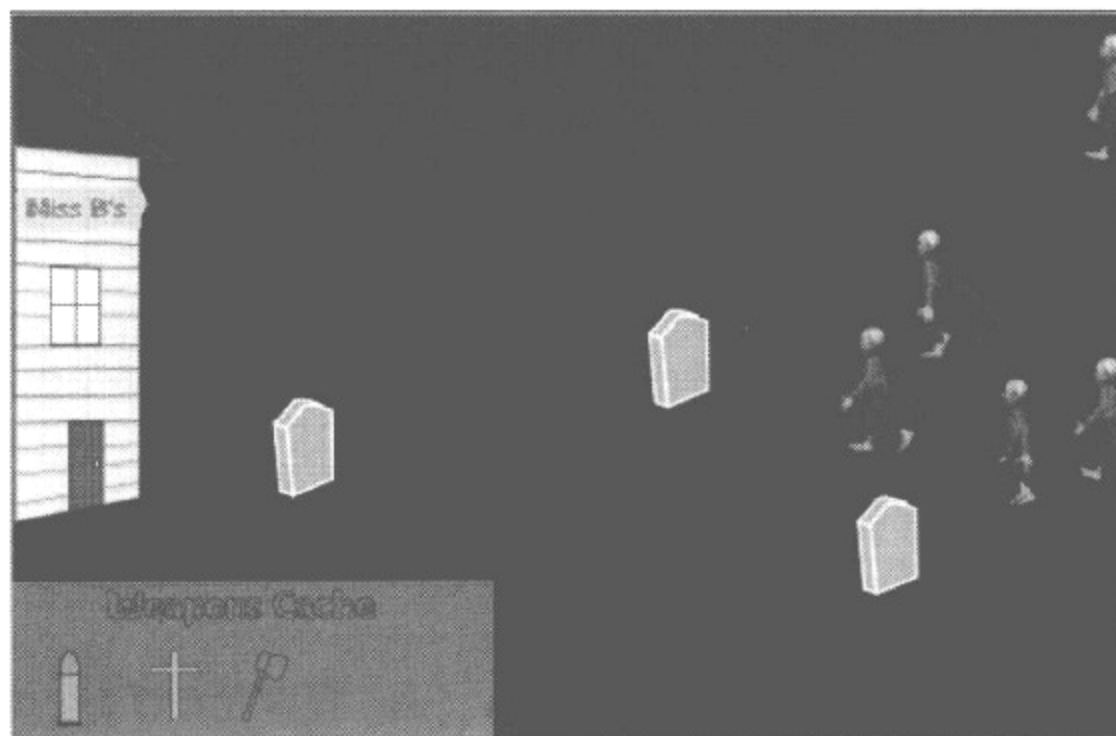


图 13.6 正在运行中的游戏第 1 关画面

做一些修改，让吸血鬼看起来稍微聪明一点儿：

- 在屏幕上划分出一些不可见的格子，使用 A\* 算法计算出每个吸血鬼到达 Miss B 房间的路径。
- 如果前进的路上有墓碑阻挡，就让吸血鬼绕开它。游戏若要做得更加真实一些的

话，应该另做两组动画以表示绕行墓碑的吸血鬼。一个表示吸血鬼正在朝玩家走来，另一个表示它正远离玩家。不过我们在这里不用做得那么花哨。

## 实现 A\* 算法

网上可以找到各种语言实现的 A\* 算法，不过本书不打算探究最有效率的实现，而是准备讲解一种最易理解的实现，以便读者能够看到路径究竟是如何计算出来的。

相对来说，A\* 算法比较耗费 CPU 资源，所以游戏中会尽可能减少需要计算的场合，仅在屏幕右方生成吸血鬼或者吸血鬼需要闪避玩家所发武器时才会计算。

根据范例游戏的屏幕大小，将其划分为  $24 \times 16$  的网格，并手工将墓碑放置在其中的某些网格内。如果该场景使用的是瓦片地图，那么这里就不需要再进行划分网格和放置墓碑这种工作了。为了清晰起见，在图 13.7 中，省略了场景的背景图，并画上了假定的网格线。在创建 A\* 算法时，需要标注出所有障碍物的位置（图 13.7 中的阴影区域）。

```

    public float h = 0.0f; // 与目标之间的距离 (该值由试探法推出)
    public boolean obstacle = false;
    public boolean footprint = false;
}

```

这些变量都很易懂，不过为了避免混淆，还是将其含义解释如下。

- float g：当前点与起点的距离，此值是已经测算好的。水平或垂直移动一格都计为 1.0f，斜向移动一格则计为 1.4f（约等于 2 的平方根）。
- float h：使用 Manhattan 估算法所预测的当前点与目标点的距离，计算时忽略障碍物。
- boolean obstacle：如果该处网格是障碍物，则此值为 true。
- boolean footprint：如果在路径查找中已经走过此处，则其为 true。它可以避免在路径查找中陷入死循环，或者卡在某两点之间来回移动。

## AStar 类

需要有一个类来实现 A\* 算法，它可以找到网格中任意起止点之间的路径，这个类叫做 AStar，其源码在程序清单 13.3 中。

程序清单 13.3 AStar.java

```

package com.pearson.lagp.v3;

import java.util.ArrayList;

import org.anddev.andengine.entity.modifier.PathModifier.Path;

import android.util.Log;

public class AStar {
    // =====
    // 常量
    // =====

    // =====
    // 字段
    // =====
    private GridLoc[][] grid;
    private int rowMax, colMax;
    private int cellWidth, cellHeight;
    private String tag = "AStar:";

    // =====
    // 构造器
    // =====
    public AStar(int pRows, int pCols, int pWidth, int pHeight) {
        // pWidth 是查找区域的总宽度，以像素为单位
    }
}

```



```

// pHeight 是查找区域的总高度, 以像素为单位
grid = new GridLoc[pRows][pCols];
rowMax = pRows-1;
colMax = pCols-1;
cellWidth = pWidth/pCols;
cellHeight = pHeight/pRows;
for (int i=0; i<pRows; i++) {
    for (int j=0; j<pCols; j++) {
        grid[i][j] = new GridLoc();
    }
}
}
// =====
// Getter 与 Setter
// =====
public void setObstacle(int pObstacleRow, int pObstacleCol){
    if (grid != null){
        grid[pObstacleRow][pObstacleCol].obstacle = true;
    }
}

public Path getPath(float pStartX, int pTargetCol, float pStartY,
    int pTargetRow, float pSpriteWidth, float pSpriteHeight){
    // 使用 A* 路径查找算法找出近似最优路径
    int nextCol, nextRow;
    int startCol, startRow;
    ArrayList<Integer> pathCols = new ArrayList<Integer>();
    ArrayList<Integer> pathRows = new ArrayList<Integer>();
    startCol = (int)pStartX/cellWidth;
    startRow = (int)pStartY/cellHeight;
    int currCol = startCol;
    int currRow = startRow;
    float[] f = new float[8];
    grid[currRow][currCol].g = 0.0f;
    grid[currRow][currCol].h =
        pTargetCol - currCol + pTargetRow - currRow;
    grid[currRow][currCol].footprint = true;

    while ((currCol != pTargetCol) || (currRow != pTargetRow)){
        // 考虑周围 8 个相邻的坐标点
        for (int i=0; i<8; i++) f[i] = 0;

        f[0] = fComp(currRow, currCol, -1, -1, 1.4f,
            pTargetRow, pTargetCol);
        f[1] = fComp(currRow, currCol, 0, -1, 1.0f,
            pTargetRow, pTargetCol);
        f[2] = fComp(currRow, currCol, +1, -1, 1.4f,
            pTargetRow, pTargetCol);
        f[3] = fComp(currRow, currCol, -1, 0, 1.0f,
            pTargetRow, pTargetCol);
        f[4] = fComp(currRow, currCol, +1, 0, 1.0f,

```

```

    pTargetRow, pTargetCol);
f[5] = fComp(currRow, currCol, -1, +1, 1.4f,
    pTargetRow, pTargetCol);
f[6] = fComp(currRow, currCol, 0, +1, 1.0f,
    pTargetRow, pTargetCol);
f[7] = fComp(currRow, currCol, +1, +1, 1.4f,
    pTargetRow, pTargetCol);

int lowidx = 0;
float pos = 10000.0f;
for (int j=0; j<8; j++){
    if (f[j]<pos){
        pos = f[j];
        lowidx = j;
    }
}
nextCol = currCol;
nextRow = currRow;
switch (lowidx){
    case (0):
        nextRow = currRow - 1;
        nextCol = currCol - 1;
        break;
    case (1):
        nextRow = currRow;
        nextCol = currCol - 1;
        break;
    case (2):
        nextRow = currRow + 1;
        nextCol = currCol - 1;
        break;
    case (3):
        nextRow = currRow - 1;
        nextCol = currCol;
        break;
    case (4):
        nextRow = currRow + 1;
        nextCol = currCol;
        break;
    case (5):
        nextRow = currRow - 1;
        nextCol = currCol + 1;
        break;
    case (6):
        nextRow = currRow;
        nextCol = currCol + 1;
        break;
    case (7):
        nextRow = currRow + 1;
        nextCol = currCol + 1;
        break;
}

```

```

    }
    // 将下一个位置添加到 Path 中,
    // 设置 footprint, 更新 currCol 和 currRow
    pathCols.add(nextCol);
    pathRows.add(nextRow);
    grid[currRow][currCol].footprint = true;
    currCol = nextCol;
    currRow = nextRow;
}
float[] xArray = new float[pathCols.size()+1];
float[] yArray = new float[pathRows.size()+1];
xArray[0] = pStartX;
yArray[0] = pStartY;

for (int i = 1; i < xArray.length; i++) {
    Float tmpX = (float)pathCols.get(i-1) * cellWidth
        - pSpriteWidth/2;
    xArray[i] = (tmpX != null ? tmpX : 0.0f);
    Float tmpY = (float)pathRows.get(i-1) * cellHeight
        - pSpriteHeight/2;
    yArray[i] = (tmpY != null ? tmpY : 0.0f);
}
return (new Path(xArray, yArray));
}
// =====
// 覆写超类及接口中的方法
// =====

// =====
// 方法
// =====

private float fComp(int pCurrRow, int pCurrCol, int pRowDiff,
    int pColDiff, float pDx, int pTargetRow, int pTargetCol){
    // 计算某个地点的 A* 值:
    // g: 与起点之间的距离
    // h: 与目标之间的距离
    // 返回值 f 等于 g + h
    // 如果该地被标记为障碍物, 或者已经走过, 又或者在网格界外,
    // 则返回一个非常大的值 (5000)
    if ((pCurrRow + pRowDiff) > rowMax) ||
        ((pCurrCol + pColDiff) > colMax) ||
        ((pCurrRow + pRowDiff) < 0) ||
        ((pCurrCol + pColDiff) < 0)) {
        return 5000.0f;
    }
    if((grid[pCurrRow + pRowDiff]
        [pCurrCol + pColDiff].obstacle) ||
        (grid[pCurrRow + pRowDiff]
        [pCurrCol + pColDiff].footprint)){
        return 5000.0f;
    }

```

```

    }

    grid[pCurrRow + pRowDiff][pCurrCol + pColDiff].g =
        grid[pCurrRow][pCurrCol].g + pDx;
    grid[pCurrRow + pRowDiff][pCurrCol + pColDiff].h =
        Math.abs(pTargetRow - (pCurrRow + pRowDiff)) +
        Math.abs(pTargetCol - (pCurrCol + pColDiff));
    return (grid[pCurrRow + pRowDiff][pCurrCol + pColDiff].g +
        grid[pCurrRow + pRowDiff][pCurrCol + pColDiff].h);
}
// =====
// 内部类与匿名类
// =====
}

```

此段代码不难理解，其中的主要部分解释如下。

- 设定算法，声明用于保存网格的二维数组 grid，网格中的每个地点都用一个 GridLoc 对象表示。这个数组所代表的网格线正如图 13.7 中所画的那样。
- AStar 类的构造器将 grid 初始化，同时也初始化了一些便于简化计算所用的变量。每次需要寻找路径时，就需要创建一个 AStar 对象。
- 使用 setObstacle() 方法来告知 A\* 算法网格中障碍物的位置。对于网格中每一个需要躲避的障碍物格子，都要用此方法设置一次。
- getPath() 方法是实现 A\* 算法的核心，为了明白其工作原理，首先需要详细讲一下它的返回值及各个参数的含义。
  - Path：返回的 Path 对象正是构造 PathModifier 所需的那种。回想一下，前文说过，可以给 Sprite 对象设定一个 PathModifier，用于指定其移动路径，这正是在本游戏中要用的方法。尽管不能确定 Path 对象所指的路径具体有多长，但是可以肯定的是它会包含两个数组，用于指定每一步移动的横坐标与纵坐标。
  - float pStartX 与 float pStartY：这两个参数用于设定起点的像素坐标。这个起点位置是必须设定的，否则精灵一开始就会跳到 (0,0) 点。这个起点坐标将会预先加入到 Path 对象中。
  - int pTargetCol 与 int pTargetRow：目标位置的列坐标与行坐标。因为 A\* 算法是以单元格为单位的，所以这里要提供的是行列坐标而非像素坐标。
  - float pSpriteWidth 与 float pSpriteHeight：算法所找到的路径是由一系列的网格坐标所组成的，需要将它换算成精灵使用的像素坐标，所以需要知道精灵的大小。
- getPath() 方法一开始初始化一些变量，因为不知道 Path 对象中的路径到底有多长，所以使用 ArrayList<Integer> 这种可以自由增长的数据结构来存储它。
- while 循环会持续执行，直到找到目标点为止。每次执行循环体的时候，它

都会向 Path 对象中增加一个坐标，每次都将根据如下步骤来决定所增加的坐标值。

- 首先初始化一个用于保存 A\* 算法中 F 值（经由  $F=G+H$  这个式子算出）的数组，然后调用 fcomp() 方法，它将会分别算出与当前点相邻的 8 个坐标点的 F 值。如果所要计算的地点上面有障碍物，或者我们已经走过该点，又或者它在当前网格界外，那么 fcomp() 方法将会返回一个非常大的值（5000.0f）用以表示此情况。否则，此方法会把根据该点与起点的距离同该点到终点的 Manhattan 距离相加，并将结果返回。
- 一旦计算了 8 个相邻坐标的值之后，就通过 for 循环来找出其中的最小值。那个值所在的点就是下一步将要移动到的点。现在读者应该清楚为何有时 fcomp() 方法要返回 5000 了。因为我们根本不想走到那种坐标点上。
- 接下来的 switch 语句根据刚算好的移动位置更新下一步的行列坐标，因为这两个值以后还需再次当做参数传给 fcomp()，以算出具有最低值的相邻坐标点。
- 将下一步的行列坐标分别加入相应的 ArrayList 对象中，并将当前地点标注为“已走过”（footprint=true）。
- 前进到下一个坐标点。
- 当 while 循环返回时，完整的路径查找就完成了，我们可以据此创建 Path 对象所需的像素坐标数组，并以此创建 Path 及 PathModifier 对象。初始化一个数组，然后将 ArrayList 中的网格坐标换算之后填入其中，并将创建好的 Path 对象返回给方法的调用者。
- fcomp() 方法非常直白，首先测试参数中所传入的地点是否有障碍物，是否在网格界外，是否已经走过。如果满足以上任何一条，则返回一个非常大的值，这样在选择下一步的移动坐标时就不会考虑它了。如果该坐标是可以合法移动的点，那么只需计算出 g（与起点的距离）和 h（使用 Manhattan 试探法计算出的该点与终点的距离）的值，将其相加得出 f 值，并返回即可。

### 将路径查找增加到 Level1Activity 类中

只需略微改动几处代码，就可以将路径查找算法运用在游戏中的吸血鬼精灵上了。程序清单 13.4 是修改之后的源码。

程序清单 13.4 修改之后的 Level1Activity.java 文件

```
public class Level1Activity extends BaseGameActivity {
    // =====
```

```

// 字段
// =====
private AStar[] aStar = new AStar[10];
private Path[] pathVamp = new Path[10];
. . .
@Override
public void onLoadResources() {
. . .
    for (int i=0; i<10; i++) {
        aStar[i] = new AStar(16, 24,
            CAMERA_WIDTH, CAMERA_HEIGHT);
        aStar[i].setObstacle(12,0);
        aStar[i].setObstacle(13,0);
        aStar[i].setObstacle(14,0);
        aStar[i].setObstacle(15,0);
        aStar[i].setObstacle(8,5);
        aStar[i].setObstacle(8,6);
        aStar[i].setObstacle(8,7);
        aStar[i].setObstacle(9,5);
        aStar[i].setObstacle(9,6);
        aStar[i].setObstacle(9,7);
        aStar[i].setObstacle(10,5);
        aStar[i].setObstacle(10,6);
        aStar[i].setObstacle(10,7);
        aStar[i].setObstacle(6,13);
        aStar[i].setObstacle(7,13);
        aStar[i].setObstacle(8,13);
        aStar[i].setObstacle(6,14);
        aStar[i].setObstacle(7,14);
        aStar[i].setObstacle(8,14);
        aStar[i].setObstacle(6,15);
        aStar[i].setObstacle(7,15);
        aStar[i].setObstacle(8,15);
        aStar[i].setObstacle(10,17);
        aStar[i].setObstacle(11,17);
        aStar[i].setObstacle(12,17);
        aStar[i].setObstacle(10,18);
        aStar[i].setObstacle(11,18);
        aStar[i].setObstacle(12,18);
        aStar[i].setObstacle(10,19);
        aStar[i].setObstacle(11,19);
        aStar[i].setObstacle(12,19);
    }

    private Runnable mStartVamp = new Runnable() {
        public void run() {
            pathVamp[i] = aStar[i].getPath(asprVamp[i].getX(), 1,
                asprVamp[i].getY(), 10, asprVamp[i].getWidth(),
                asprVamp[i].getHeight());

```



```

        asprVamp[i].registerEntityModifier(
            new SequenceEntityModifier (
                new AlphaModifier(5.0f, 0.0f, 1.0f),
                new PathModifier(60.0f, pathVamp[i])
            ));
    }
}

```

以上这段代码看起来也很简单：

- 声明了 10 个吸血鬼精灵所使用的 A\* 算法对象以及路径对象。
- 将每一处障碍物的坐标告知 A\* 算法对象。必须为每个精灵对象使用单独的 A\* 算法对象，不然它们就会按照同一个起止坐标来计算路径并进行移动了。
- 在 mStartVamp 所指的 Runnable 任务中，吸血鬼精灵对象在创建出来之后，立刻查询自己从当前地点到达目标地点的最佳路径。用存放查询结果的 Path 对象构建一个 PathModifier 对象，将它应用于精灵对象上，替换掉早先使用的 MoveModifier 对象，那个修改器只能指示精灵做简单的直线运动。

### 13.3 总结

本章高度概括了游戏中能用到的某些人工智能技术，并着重研究了其中一种，那就是 A\* 路径查找算法。有很多 AI 技术方面的著作可供参考，至于讲解如何在游戏中运用 AI 技术的书，那就更多了。如果读者想要研究游戏中可以用到的人工智能算法，那么本章可以算是一个好的起点。

### 13.4 习题

1. 如果运行本章的游戏代码，就会发现一个问题。不管吸血鬼从哪里出发，路上有何种障碍，它们都能走到 Miss B 家的大门口。然而，因为墓碑和武器库的摆放位置，导致所有吸血鬼最后都会排成一行在屏幕上穿行。

修改 Level1Activity.java，使每个吸血鬼随机向左走一段时间之后再寻找通往 Miss B 家门口的路径，看看用这种方法能不能让吸血鬼在屏幕上分布得稍微稀疏一些。

2. 现在吸血鬼看起来仍然很笨，当武器朝其飞来时，它仍然只会按照预先计算好的路径前行，有时甚至直接撞在武器上。修改 Level1Activity.java，当玩家在屏幕中放置子弹或斧头时，吸血鬼会闪开几步，然后重新计算通往 Miss B 家门的路径。



## 第 14 章

# 计分与碰撞

- 14.1 计分系统设计
- 14.2 AndEngine 的碰撞
- 14.3 开始计算玩家的得分
- 14.4 《墓地》(第 1 关) 场景
- 14.5 《打吸血鬼》
- 14.6 《愤怒的村民》
- 14.7 总结
- 14.8 习题

对于大多数游戏来说，分数是个很重要的因素，因为它可以反映玩家的游戏水平，同时能告诉玩家还有多少提升空间。

计分系统可以做得很复杂，也可以很简单。一个简单的计分系统至少应该包含如下要素：

- ☐ 记录当前的游戏分数
- ☐ 记录一定数量的最高得分
- ☐ 向玩家显示当前得分
- ☐ 显示高分列表

复杂的计分系统可能会包含如下功能：

- ☐ 可能会通过排名系统（Ranking System）和比赛来记录其他（在线）玩家的分数。
- ☐ 记录同一个设备上不同玩家的分数。
- ☐ 记录产生最高分的时间和日期。
- ☐ 以时间（例如解开谜题所花时间）而不是分数来统计分数。
- ☐ 当玩家获得一定分数时，给予奖励。比如说，在玩家达到某个分数之后，可以奖励给玩家一个新武器，这样游戏就会更有趣。
- ☐ 给玩家与游戏没有直接关系的奖品，这种奖品可以是任何东西，可以是奖杯或者一些特殊的隐藏物品。手机游戏大量地使用了成就系统，以便在基本的游戏内容之上，继续扩展游戏的可玩性。

本书关注简单的计分系统，将复杂的计分与成就系统留给读者去自行发挥。

游戏中频繁发生的计分事件一般都是由种种碰撞激发的。在前述的 V3 范例游戏制作过程中，已经编写了大量的碰撞检测代码，不过本章还将讲述 AndEngine 所提供的两种碰撞检测方式，通过它们可以增强游戏的计分功能：

- ☐ 使用 AndEngine 所提供的 `collidesWith()` 方法来检测 `Sprite` 和 `Shape` 对象的碰撞
- ☐ 使用 `Box2D` 物理引擎所提供的碰撞检测功能

## 14.1 计分系统设计

AndEngine 并未提供任何用于计分的 API 或者数据结构，所以必须自己来实现它。当然了，我们会利用 AndEngine 的显示功能以及 Android 系统 API 来构建一个能够与 AndEngine 协同工作的简单计分系统。在第 11 章的习题 3 中，已经做了一点儿与计分有关的工作（这部分的具体细节可以参阅附录中的习题解答），本章将会实现更多的功能。

本游戏的计分系统需求如下：

- ☐ 更新所有小游戏的分数。
- ☐ 记录 5 个最高成绩。
- ☐ 在 Scores 页面，显示每个小游戏的 5 个最高分，按分数由高到低排序。

- 在每个小游戏的场景中显示该游戏的当前分数。
- 将分数存储起来，以便下次运行游戏时还能访问到。

### 14.1.1 更新小游戏取得的分数

本游戏将使用 Android 系统的 SharedPreferences 来保存游戏分数，回想一下，在第 11 章中，也曾使用它来保存音乐与音效的开关值。Android 系统将会为整个应用程序创建一个 SharedPreferences 对象，访问该对象的方法如程序清单 14.1 所示。

程序清单 14.1 读取 SharedPreferences 之中保存的值

---

```
scores = getSharedPreferences("scores", MODE_PRIVATE);
highScores[4] = scores.getInt("Level1-4", 0);
highScores[3] = scores.getInt("Level1-3", 0);
highScores[2] = scores.getInt("Level1-2", 0);
highScores[1] = scores.getInt("Level1-1", 0);
highScores[0] = scores.getInt("Level1-0", 0);
. . .
```

---

上述代码将存放分数的 SharedPreferences 对象命名为“scores”，它包含有键-值对，其中的键是 String 对象，而值则是表示分数的整数。每个小游戏中都会有 5 个键-值对来保存 5 个最高分。getInt() 方法的参数分别是键以及默认值，当键不存在时，默认值将充当该方法的返回值。因此，在上述代码中，如果找不到某个键，则会返回 0。

将分数值放到 SharedPreferences 对象中的方法非常简单，如程序清单 14.2 所示。

程序清单 14.2 向 SharedPreferences 之中写入值

---

```
scores = getSharedPreferences("scores", MODE_PRIVATE);
scoresEditor = scores.edit();
scoresEditor.putInt("Level1-4", highScores[4]);
scoresEditor.commit();
. . .
```

---

像这样使用 SharedPreferences 来存储分数会有风险，如果要在其中保存有价值的数据，则需格外小心。如果玩家取得了手机的根用户权限（即通过刷非官方系统固件而获得了 Linux 中 root 用户的使用权），那么就可以访问并编辑这些经由 SharedPreferences 来存储的数据了。像 V3 游戏这样，仅仅保存分数这种没有实用价值的数据，使用 SharedPreferences 就没有问题。使用 SQLite 也会遇到同样的情况，如果需要保存一些不想让玩家看到的数据，则需要加密。

### 14.1.2 记录 5 个最高分

在每个小游戏开始运行之前，游戏代码将会取得该小游戏的 5 个最高分，如果有需

要，还会在游戏结束时更新它们。在游戏结束时，如果玩家的得分比最高分中的某一个要高，那么分数列表将会更新。程序清单 14.3 展示了某个小游戏结尾部分的处理代码，这段代码检测玩家是否取得了比高分榜上的分数更高的分数。

程序清单 14.3 游戏结尾处用于处理分数的代码

---

```
int[] newHighScores = {0,0,0,0,0};
for (int i=4; i>-1; i--){
    if (thisScore > highScores[i]){
        newHighScores[i] = thisScore;
        for (int j=i-1; j>-1; j--){
            newHighScores[j] = highScores[j+1];
        }
        if (i==4) newHigh = true;
        break;
    } else {
        newHighScores[i] = highScores[i];
    }
}
for (int i=0; i<5; i++) highScores[i] = newHighScores[i];
scoresEditor.putInt("Level1-4", highScores[4]);
scoresEditor.putInt("Level1-3", highScores[3]);
scoresEditor.putInt("Level1-2", highScores[2]);
scoresEditor.putInt("Level1-1", highScores[1]);
scoresEditor.putInt("Level1-0", highScores[0]);
scoresEditor.commit();
. . .
```

---

原有的 5 个最高分保存在 `highScores[]` 整数数组中，小游戏结束时玩家取得的分数保存在 `thisScore` 变量中。代码新建了一个名为 `newHighScores[]` 的数组，用于保存可能是更新之后的高分列表。如果 `highScores[]` 中的某个分数大于等于 `thisScore`，就将其复制到 `newHighScores[]` 中的对应位置中。如果发现 `highScores[]` 中的某个分数小于 `thisScore`，那么就将 `thisScore` 以及 `highScores[]` 中的其余高分都复制到 `newHighScores[]` 中。

一旦获取了新的高分榜之后，就将其复制回 `highScores` 中，并更新 `SharedPreferences` 中的值。注意，在使用 `putInt()` 方法修改值之后，必须调用 `commit()` 方法提交更改，否则修改不会生效。

### 14.1.3 在小游戏场景中显示分数

第 7 章讲述了 `Text` 实体的用法，本章将使用 `ChangeableText` 实体来保存小游戏运行时玩家所得的分数。小游戏的分数将始终显示在屏幕右上角，会用一个私有方法来更

新屏幕上显示的分数以及 thisScore 中所保存的分数值。程序清单 14.4 是一个简单的分数显示范例，其中包含了用于更新的私有方法。

程序清单 14.4 使用 ChangeableText 对象来显示分数

```
// 分数显示
mCurrScore = new ChangeableText(0.75f * CAMERA_WIDTH, 10.0f,
    mFont32, "Score: 0", "Score: XXXXXX".length());
scene.getLastChild().attachChild(mCurrScore);
. . .
mAddScore(BULLET_VAMP_SCORE);
. . .
private void mAddScore(int pAdder){
    thisScore += pAdder;
    mCurrScore.setText("Score: " + thisScore);
}
. . .
```

上述代码将这个可以显示 6 位数字的 ChangeableText 对象放置在屏幕的右上角，它所使用的 mFont32 对象已经由该 Activity 的代码载入了，使用的是 Flubber TrueType 字型。每次需要更新游戏中的分数时，就调用 mAddScore() 方法，它会更新 thisScore 内部变量，也会更新用于显示分数的 mCurrScore 文本对象。

#### 14.1.4 分数页面的显示

该页面不需要太复杂的代码，只是一个用于显示各个小游戏最高分的简单列表而已。为了和游戏整体风格一致，这里使用的也是 Flubber 字型。与第 11 章习题 3 所做的准备工作一样，现在需要实现从 Options 页面进入 Scores 页面所需的代码。图 14.1 显示了包含高分榜的 Scores 页面截图。

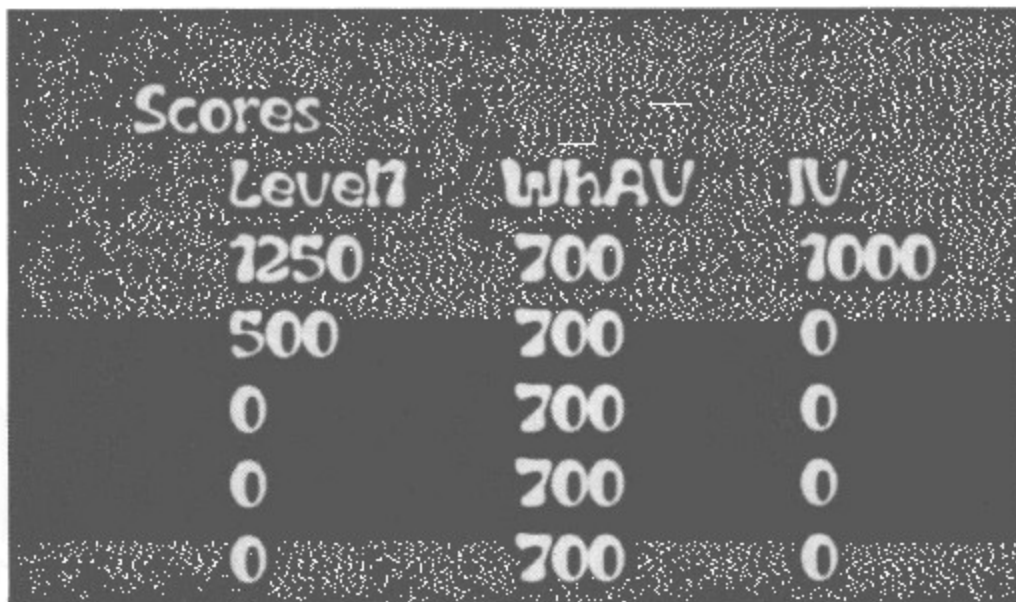


图 14.1 Scores 页面



修改后的 ScoresActivity.java 代码如程序清单 14.5 所示。

程序清单 14.5 加入显示高分榜功能的 ScoresActivity.java 代码

```
package com.pearson.lagp.v3;

+imports

public class ScoresActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    // =====
    // 字段
    // =====

    protected Camera mCamera;

    protected Scene mScoresScene;
    private Text mTitle, mHeaders;
    private Text[] mScoreL = new Text[5];
    private Text[] mScoreW = new Text[5];
    private Text[] mScoreI = new Text[5];
    private Texture mFontTexture;
    private Font mFont;
    private SharedPreferences scores;
    private SharedPreferences.Editor scoresEditor;

    // =====
    // 构造器
    // =====

    // =====
    // Getter 与 Setter 方法
    // =====

    // =====
    // 覆写超类及接口中的方法
    // =====

    @Override
    public Engine onLoadEngine() {
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        scores = getSharedPreferences("scores", MODE_PRIVATE);
        scoresEditor = scores.edit();
        return new Engine(new EngineOptions(true,
```

```

        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
        CAMERA_HEIGHT), this.mCamera));
    }

    @Override
    public void onLoadResources() {
        /* 载入 Font 与 Textures 对象。 */
        this.mFontTexture = new Texture(256, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);

        FontFactory.setAssetBasePath("font/");
        this.mFont = FontFactory.createFromAsset(this.mFontTexture,
        this, "Flubber.ttf", 32, true, Color.RED);
        this.mEngine.getTextureManager().loadTexture(
        this.mFontTexture);
        this.mEngine.getFontManager().loadFont(this.mFont);
    }

    @Override
    public Scene onLoadScene() {
        /* 将摄像头置于背景中央。 */
        final int centerX = (CAMERA_WIDTH) / 2;
        final int centerY = (CAMERA_HEIGHT) / 2;

        this.mScoresScene = new Scene(1);
        /* 增加背景及分数对象。 */
        mScoresScene.setBackground(new ColorBackground(
        0.0f, 0.0f, 0.0f));
        mTitle = new Text( centerX - 200, centerY - 120, mFont,
        "Scores");
        mScoresScene.getLastChild().attachChild(mTitle);
        mHeaders = new Text( centerX - 150, centerY - 80, mFont,
        "Levell    WhAV    IV");
        mScoresScene.getLastChild().attachChild(mHeaders);
        for (int i=0; i<5; i++){
            mScoreL[i] = new Text( centerX - 150,
            (centerY - 40) + (4-i)*40, mFont, "" +
            scores.getInt("Levell-"+i, -1));
            mScoreW[i] = new Text( centerX, (centerY - 40) +
            (4-i)*40, mFont, "" +
            scores.getInt("WhAV-"+i, -1));
            mScoreI[i] = new Text( centerX + 150,
            (centerY - 40) + (4-i)*40, mFont, "" +
            scores.getInt("IV-"+i, -1));
            mScoresScene.getLastChild().attachChild(mScoreL[i]);
            mScoresScene.getLastChild().attachChild(mScoreW[i]);
            mScoresScene.getLastChild().attachChild(mScoreI[i]);
        }
        return this.mScoresScene;
    }
}

```

```

@Override
public void onLoadComplete() {
}
}

```

这段代码很直白，所以就不再做过多的讲解了。需要注意的是，代码中的所有位置都是相对于 480×320 像素的屏幕大小来说的。如果你的游戏比 V3 范例游戏更加灵活，需要自动适应不同大小的屏幕，则需要改进这部分代码。

## 14.2 AndEngine 的碰撞

迄今为止的范例游戏代码，很大程度上忽略了对碰撞的检测，不过现在该好好地处理这个问题，以便正确地更新小游戏中的分数。就像本章开头简述的那样，在 AndEngine 游戏中进行碰撞检测有两种办法，选择哪种办法取决于游戏中是否使用了物理学效果。

- ❑ 如果游戏中没有使用 Box2D 物理引擎，那么可以使用 AndEngine 所提供的碰撞检测机制来检测 Sprite 对象与 Shapes 对象的碰撞。这种方式可以检测两个对象是否正在碰撞，推荐在当前 Scene 对象所注册的 UpdateHandler 中进行碰撞检测。
- ❑ 如果游戏使用了 Box2D，则可以利用其中内嵌的碰撞检测系统，该系统的功能比 AndEngine 更加灵活，也更加高效。开发者可以在定义好的 PhysicsWorld 对象上注册一个 ContactListener 方法，当虚拟环境中的任意两个物体碰撞时，ContactListener 中的 beginContact() 和 endContact() 方法就会被回调。Box2D 甚至可以在不参与物理模拟的物体上进行碰撞检测，不过 V3 游戏并不会这么做（如果读者想要了解更多关于碰撞检测的知识，请参阅第 12 章中引述的 Box2D 用户手册）。

### 14.2.1 AndEngine 的 Shape 碰撞

在 AndEngine 的 Shape 对象（也包含 Sprite 对象）中进行碰撞检测很简单，只需要在 Scene 对象中的物体位置都更新完毕之后，就可以根据此位置来检测 Shape 对象的碰撞。程序清单 14.6 是进行碰撞检测所用的代码范式。

程序清单 14.6 AndEngine 中 Shape 对象之间的碰撞检测

```

scene.registerUpdateHandler(new IUpdateHandler() {
    @Override
    public void reset() { }
}

```

```

@Override
public void onUpdate(final float pSecondsElapsed) {
    if (spriteA.collidesWith(spriteB)){
        // 执行碰撞发生后的逻辑
    }
}
};

```

就是这么简单！开发者只需要在每个所关注的物体上执行检测即可。

### 14.2.2 Box2D 的碰撞

Box2D 的碰撞检测有时会变得非常复杂。在默认情况下，Box2D 在物理模拟的每一帧中会针对所有的碰撞都调用 `ContactListener` 方法。Box2D 同时也有一个碰撞过滤系统，通过指定位掩码，可以过滤掉不重要的碰撞通知。在本书的范例游戏中，不需要用到这些过滤器。

程序清单 14.7 展示了 `ContactListener` 方法。`beginContact()` 与 `endContact()` 回调方法都有一个类型为 `Contact` 的参数，其中含有本次碰撞的所有细节：

- ❑ 参与碰撞的两个固定物，及其属性：
  - ❑ 密度
  - ❑ 摩擦系数
  - ❑ 弹性
- ❑ 与这两个固定物相关联的物体及其属性：
  - ❑ 质量
  - ❑ 密度
  - ❑ 线性速度
  - ❑ 角度
  - ❑ 角速度

简言之，`Contact` 对象包含了有关碰撞的所有信息。同时开发者也可以获取每个物体中自定义的 `UserData` 数据，该数据是一种辨识物体类型的方式。

程序清单 14.7 Box2D 的碰撞监听

```

this.mPhysicsWorld.setContactListener( new ContactListener() {
    @Override
    public void beginContact(Contact contact) {
        Body bodyA = contact.getFixtureA().getBody();
        Body bodyB = contact.getFixtureB().getBody();
    }

    public void endContact(Contact contact) {
    }
}
});

```

程序清单 14.7 中有个问题值得注意：通过 Box2D 的碰撞检测回调，并不能确定 bodyA 与 bodyB 的具体身份。如果需要检测两种类型物体之间的碰撞，则需要使用 UserData（在本章稍后的 IVActivity 类代码中将会看到这种用法）。

## 14.3 开始计算玩家的得分

在写好了查看分数与记录分数的机制之后，需要开始真正计算玩家的得分。虽说每个小游戏的得分方式不同，不过本书假定第 1 关的分数与打吸血鬼游戏的分数相同。我们将浏览每一个小游戏的代码，并找出需要添加计分代码的地方，为其增加计分功能。

## 14.4 《墓地》（第 1 关）场景

游戏的第 1 关是 V3 游戏中最复杂的部分，它需要做出多处修改，以便增加计分功能。现将为了增加计分功能需要做的修改罗列如下。

- 计分方式：需要实现上一节所描述的计分机制，这样就可以根据玩家所杀死的吸血鬼数目来计算得分了。
- 吸血鬼的死亡：玩家已经可以通过触摸来杀死吸血鬼，现在还需要让武器在碰到吸血鬼时能够将它杀掉。当吸血鬼死亡时，这个事件将会激发分数的更新（通过触摸方式击杀吸血鬼可得 50 分，使用子弹射杀可得 100 分，使用斧头可得 200 分，而十字架则是 500 分）。为了增加趣味，可以调整不同武器的得分规则以便寻求平衡：我们规定子弹可以杀掉它运行路径上的所有吸血鬼，而斧头只能杀掉它碰到的第一个吸血鬼，十字架只能原地不动等待吸血鬼踩上来。
- 小游戏的结束：为了统计出游戏的最终得分，必须将小游戏结束。第 1 关的小游戏会在吸血鬼到达 Miss B 的家门（吸血鬼胜利）或所有吸血鬼都被杀死（玩家胜利）时终止。在小游戏结束时，会显示结果画面，玩家可以通过其上的选项来选择是重新玩还是返回主菜单。图 14.2 显示了玩家胜利时的结束场景，图 14.3 则是吸血鬼胜利时的画面。

现在的 Activity 类很大，将近 700 行代码。所以将其分为数小节来讲解。

### 14.4.1 常量和字段

对于 Activity 类中常量与字段的修改如程序清单 14.8 所示。



图 14.2 第1关玩家获胜的结果画面

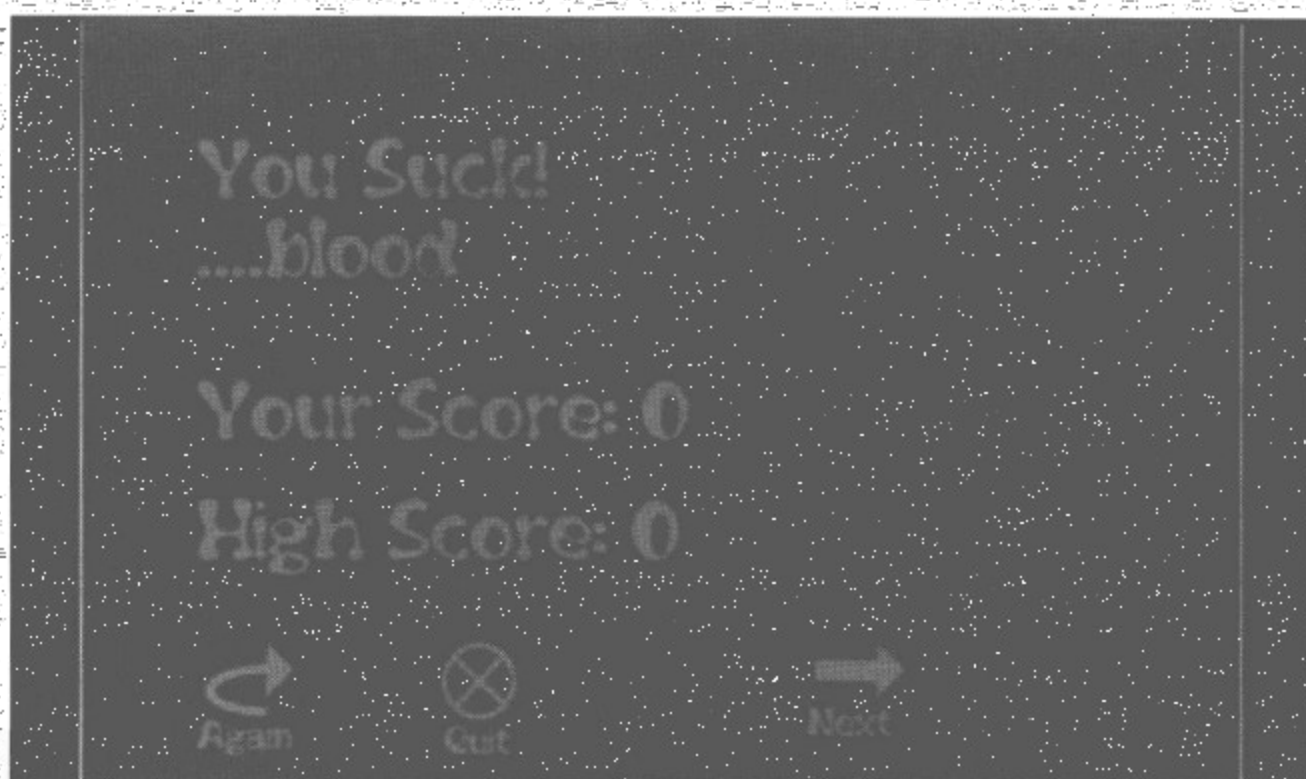


图 14.3 第1关吸血鬼获胜的结果画面

程序清单 14.8 增加了计分功能之后的 Level1Activity.java 代码中的常量与字段部分

```
public class Level1Activity extends BaseGameActivity {
    // =====
    // 常量
    // =====
}
```



```

private static final int TOUCH_VAMP_SCORE = 50;
private static final int BULLET_VAMP_SCORE = 100;
private static final int HATCHET_VAMP_SCORE = 200;
private static final int CROSS_VAMP_SCORE = 500;

private static final int NUKE_BULLET = -1;
private static final int NUKE_HATCHET = NUKE_BULLET+1;
private static final int NUKE_CROSS = NUKE_HATCHET+1;
private static final int NUKE_TOUCH = NUKE_CROSS+1;

private static final int MAX_VAMPS = 10;
private static final int VAMP_RATE = 2000;

private static final boolean PLAYER_WINS = true;
private static final boolean VAMPIRES_WIN = false;

// Miss B 家前门的位置
private static final Rectangle MissBs = new Rectangle(
    35.0f, 195.0f, 15.0f, 35.0f);

// =====
// 六字段
// =====

private Texture mPopUpTexture;

private TextureRegion mEndBackTextureRegion;
private TextureRegion mAgainButtonTextureRegion;
private TextureRegion mQuitButtonTextureRegion;
private TextureRegion mNextButtonTextureRegion;
private TextureRegion mNewHighTextureRegion;
private Texture mFontTexture;
private Font mFont32;
private ChangeableText mCurrScore;

private Sprite endBack, newHigh, againButton, quitButton,
    nextButton;
private int nVamp, nVampsKilled;

private SharedPreferences scores;
private SharedPreferences.Editor scoresEditor;
private int[] highScores = new int[5];
private int thisScore = 0;

```

首先创建一些分值常量，并定义一组用于记录以各种方式击杀吸血鬼数量的标志。我们需要将杀死吸血鬼的代码放到一个独立的方法之中，因为不管它是以何种方式被杀掉的，处理代码都是一样的。该类定义了NUKE\_BULLET、NUKE\_HATCHET等标志，

处理击杀吸血鬼的那个方法可以依据它们来算出正确的得分。

接下来定义了屏幕上同时出现的最大吸血鬼数量，以及它们出现的频率。在上一个版本的 Level1Activity 代码中，这些数值都以硬编码的形式出现。这里所做的修改是为了将来能够通过这些变量来调整游戏难度。第 16 章将会讲解如何改变游戏的难度。

在常量定义区的最后，增加了用于表示玩家获胜还是吸血鬼获胜的 Boolean 变量。同时还定义了一个覆盖 Miss B 家门的 Rectangle 对象，用于判断吸血鬼是否已经到达目标。

该类所定义的字段几乎都是为了实现图 F4.2 及 14.3 所示的弹出式结束场景并显示分数。代码还增加了一个统计被杀吸血鬼数量的整数变量，以判断是否玩家已经杀死了所有的吸血鬼，从而获得游戏的胜利。其余变量则是为了使用名为 scores 的 SharedPreferences 对象而定义的。

#### 14.4.2 onLoadEngine 方法与 onLoadResources 方法

onLoadEngine() 方法所做的唯一修改就是增加了使用 scores 这个 SharedPreferences 对象的代码。程序清单 14.1 中已经列出了这部分修改，现在的代码把最高分直接存入整型数组 highScores 中。

修改之后的 onLoadResources() 方法如程序清单 14.9 所示。

程序清单 14.9 增加了计分功能之后的 Level1Activity.java 代码中的 onLoadResources() 方法

```
@Override
public void onLoadResources() {
    ...
    mPopUpTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mEndBackTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mPopUpTexture, getApplicationContext(),
            "endback.png", 0, 0);
    mAgainButtonTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mPopUpTexture, getApplicationContext(),
            "againbutton.png", 0, 330);
    mQuitButtonTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mPopUpTexture, getApplicationContext(),
            "quitbutton.png", 50, 330);
    mNextButtonTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mPopUpTexture, getApplicationContext(),
            "nextbutton.png", 100, 330);
    mNewHighTextureRegion =
        TextureRegionFactory.createFromAsset(
```

```

        this.mPopUpTexture, getApplicationContext(),
        "newhigh.png", 100, 400);
mEngine.getTextureManager().loadTexture(
    this.mPopUpTexture);

this.mFontTexture = new Texture(256, 512,
    TextureOptions.BILINEAR_PREMULTIPLYALPHA);
FontFactory.setAssetBasePath("font/");
mFont32 = FontFactory.createFromAsset(this.mFontTexture,
    this, "Flubber.ttf", 32, true, Color.RED);
mEngine.getTextureManager().loadTexture(this.mFontTexture);
mEngine.getFontManager().loadFont(this.mFont32);
. . .

```

这部分代码平淡无奇。它使用结束场景所需的位图创建了相应的 Texture 对象，同时还创建了用于显示分数的 Font 对象。

### 14.4.3 onLoadScene 方法

与前面几个例子一样，一旦写好 onLoadScene() 方法，游戏就会变得有趣起来。该方法增加了用于使武器在碰到吸血鬼时将其杀掉的代码，同时也创建并加入了用于显示分数的各种变量，还创建了结束场景所需的 Sprite 对象。完整的修改如程序清单 14.10 所示。

程序清单 14.10 增加了计分功能之后的 Level1Activity.java 代码中的 onLoadScene() 方法

```

@Override
public Scene onLoadScene() {
. . .
    scene.registerUpdateHandler(new IUpdateHandler() {
        @Override
        public void reset() { }

        @Override
        public void onUpdate(final float pSecondsElapsed) {
            for (int i=0; i<nVamp; i++){
                if (asprVamp[i].collidesWith(bullet)){
                    mNukeVamp(i, NUKE_BULLET);
                }
                if (asprVamp[i].collidesWith(hatchet)){
                    mNukeVamp(i, NUKE_HATCHET);
                }
                if (asprVamp[i].collidesWith(cross)){
                    mNukeVamp(i, NUKE_CROSS);
                }
                if (asprVamp[i].collidesWith(MissBs)){
                    // 游戏结束，吸血鬼获胜
                    mGameOver(VAMPIRES_WIN);
                }
            }
        }
    });
}

```

```

    }
    if ((touchActive) &&
        (asprVamp[i].collidesWith(touchRect))) {
        mNukeVamp(i, NUKE_TOUCH);
    }
}
});

// 显示分数
mCurrScore = new ChangeableText(0.75f * CAMERA_WIDTH,
    10.0f, mFont32, "Score: 0",
    "Score: XXXXXX".length());
scene.getLastChild().attachChild(mCurrScore);

// 创建结束场景所需的 Sprite 对象, 不过现在并未将其加入场景
endBack = new Sprite(
    (CAMERA_WIDTH - mEndBackTextureRegion.getWidth()) / 2,
    (CAMERA_HEIGHT - mEndBackTextureRegion.getHeight()) / 2,
    mEndBackTextureRegion);
newHigh = new Sprite(0.0f, 0.0f, mNewHighTextureRegion);
againButton = new Sprite(0.0f, 0.0f,
    mAgainButtonTextureRegion) {
    @Override
    public boolean onAreaTouched(final TouchEvent
        pAreaTouchEvent, final float pTouchAreaLocalX,
        final float pTouchAreaLocalY) {
        switch(pAreaTouchEvent.getAction()) {
            case TouchEvent.ACTION_DOWN:
                // 待定
        }
        return true;
    }
};

nextButton = new Sprite(0.0f, 0.0f,
    mNextButtonTextureRegion) {
    @Override
    public boolean onAreaTouched(final TouchEvent
        pAreaTouchEvent, final float pTouchAreaLocalX,
        final float pTouchAreaLocalY) {
        switch(pAreaTouchEvent.getAction()) {
            case TouchEvent.ACTION_DOWN:
                // 待定
        }
        return true;
    }
};

quitButton = new Sprite(0.0f, 0.0f,
    mQuitButtonTextureRegion) {
    @Override
    public boolean onAreaTouched(final TouchEvent

```

```

        pAreaTouchEvent, final float pTouchAreaLocalX,
        final float pTouchAreaLocalY) {
            switch(pAreaTouchEvent.getAction()) {
                case TouchEvent.ACTION_DOWN:
                    // 待定
            }
            return true;
        }
    };
    return scene;
}

```

该段代码加入了一个 UpdateHandler 对象，用于在每次更新时检测碰撞。代码循环检查每个吸血鬼对象，看其是否与子弹、斧头、十字架、Miss B 的家门或者玩家的触摸点相碰。如果是前三种或最后一种情况，那么表示这个吸血鬼被干掉了，代码将会调用稍后定义的 mNukeVamp() 方法。如果吸血鬼到达了 Miss B 的家门口，那么将结束游戏并判定吸血鬼获胜，此时还要调用稍后定义的 mGameOver() 方法。

其后的代码创建了用于显示分数的 ChangeableText 对象，还有表示结束场景中按钮的 Sprite 对象。此时这些按钮并不起作用，把这部分代码留到第 16 章再写。

#### 14.4.4 mStartVamp 任务

回想一下前面的代码，mStartVamp() 是用于生成吸血鬼的 Runnable 任务。这里所要做的唯一修改就是改变玩家碰到吸血鬼之后的处理代码。为了改变游戏逻辑，需要修改 onAreaTouched() 方法。程序清单 14.11 列出了修改之后的方法代码，该方法覆写了 AnimatedSprite 类中的那个方法。

程序清单 14.11 增加了计分功能之后的 Level1Activity.java 代码中的 mStartVamp() 任务

```

private Runnable mStartVamp = new Runnable() {
    public void run() {
        . . .
        asprVamp[i] = new AnimatedSprite(CAMERA_WIDTH - 30.0f,
            startY, mScrumTextureRegion.clone()) {
            @Override
            public boolean onAreaTouched(
                final TouchEvent pAreaTouchEvent,
                final float pTouchAreaLocalX,
                final float pTouchAreaLocalY) {
                switch(pAreaTouchEvent.getAction()) {
                    case TouchEvent.ACTION_DOWN:
                        /* 附近有吸血鬼吗? */
                        touchRect = new Rectangle (
                            pAreaTouchEvent.getX(),
                            pAreaTouchEvent.getY(), 20.0f, 20.0f);

```



```

        touchActive = true;
        break;
    case TouchEvent.ACTION_UP:
        touchActive = false;
    }
    return true;
}
};

```

在早前版本的 `onAreaTouch()` 方法中，是通过比较触摸点与精灵位置来进行碰撞检测的。这里修改一下，以便将触摸与几种武器的碰撞检测统一起来，使得碰撞发生后都能统一调用 `mNukeVamp()` 方法，而且，这么做也会简化游戏代码。

## 14.5 《打吸血鬼》

《打吸血鬼》游戏比上一节中所说的墓地攻防游戏要简单，不过还是需要增加计分机制，使《打吸血鬼》游戏能够正常结束。

- 计分方式：本书将重用上一节中的大部分代码，并按照玩家成功关闭棺材的个数来计分。同时规定如果某个棺材在打开之后一段时间内没有被玩家关闭，那么将会扣分。
- 棺材的关闭：代码将设置两个时间常数，以便在第 16 章通过修改它们来调整游戏难度。第一个时间常数是 `OPEN_RATE`，表示两个棺材打开的间隔时间。第二个常数是 `OPEN_TIME`，表示棺材在这段时间内保持打开，直到玩家通过触摸将其关闭为止。还有一个常数叫做 `OPENS_PER_GAME`，表示游戏中保持打开的最大棺材数。
- 小游戏的结束：当打开并关闭（不论是因为玩家触摸而关闭还是因为超时而关闭）的棺材数达到 `OPENS_PER_GAME` 的值时，结束小游戏。这里并没有真正的输赢，所以使用图 14.2 中表示玩家获胜的结果画面就好。

这里还是将代码分为不同的小节来讲解，其中相似的地方请读者参阅早先的 `Level1Activity` 类代码。

### 14.5.1 常量和字段

除了 `Level1Activity` 中为了显示结果画面而增加的字段之外，`WAVActivity` 类将会再引入一些用于计分的常量与字段，如程序清单 14.12 所示。

程序清单 14.12 增加了计分功能之后的 `WAVActivity.java` 代码中的常量与字段

```

...
private static final int CLOSE_COFFIN_SCORE = 100;

```



```
private static final int OPEN_RATE = 4000;
private static final int OPENS_PER_GAME = 10;
private static final int STAY_OPEN = 2000;

private int mNumClosed = 0;
private ArrayList<Integer> openCoffins = new ArrayList<Integer>();
```

常量的意义上一节已经讲过了，这里引入了两个用于记录打开的棺材数的变量。名为 openCoffins 的 ArrayList 变量是一个先进先出（first in, first out, FIFO）列表，用于保存已经打开的棺材编号。

对于 onLoadEngine() 与 onLoadResources() 方法所做的代码变更就不再讲述了，因为它们与 Level1Activity 类中所做的修改非常相似，包括从 SharedPreferences 中取得最高分、载入结果画面所需的 Texture 对象等。

## 14.5.2 onLoadScene 方法

onLoadScene() 方法是执行计分逻辑的地方，让我们通过程序清单 14.13 看看如何使用 ArrayList 来保存打开的棺材编号。

程序清单 14.13 增加了计分功能之后的 WAVActivity.java 代码中的 onLoadScene() 方法

```
@Override
public Scene onLoadScene() {
    scene.setOnSceneTouchListener(new IOnSceneTouchListener() {
        @Override
        public boolean onSceneTouchEvent(
            final Scene pScene,
            final TouchEvent pSceneTouchEvent) {
            switch(pSceneTouchEvent.getAction()) {
                case TouchEvent.ACTION_DOWN:
                    /* 获得触摸点所对应的瓦片对象。 */
                    tmxTile = tmxLayer.getTMXTileAt(
                        pSceneTouchEvent.getX(),
                        pSceneTouchEvent.getY());
                    if((tmxTile != null) &&
                        (tmxTile.getGlobalTileID() ==
                         mOpenCoffinGID)) {
                        mAddScore(CLOSE_COFFIN_SCORE);
                        tmxTile.setGlobalTileID(
                            mWAVTMXMap, mCoffinGID);
                    }
                    break;
                case TouchEvent.ACTION_UP:
                    break;
            }
        }
    });
}
```

```

        return true;
    }
    });
    . . .

```

早先的 WAVActivity.java 代码中已经有这个名为 onSceneTouchEvent() 的覆写方法了，这里实际上只增加了一句代码（为了读者能够理解代码的上下文，程序清单中还是列出了完整的代码）：

```
mAddScore(CLOSE_COFFIN_SCORE);
```

### 14.5.3 openCoffin 和 closeCoffin 方法

现在看一下用于打开及关闭棺材所用的代码，修改之后的完整代码在程序清单 14.14 中。

程序清单 14.14 增加了计分功能之后的 WAVActivity.java 代码中的  
openCoffin() 与 closeCoffin() 方法

```

private Runnable openCoffin = new Runnable() {
    public void run() {
        int openThis = gen.nextInt(coffinPtr);
        int tileRow = coffins[openThis]/15;
        int tileCol = coffins[openThis] % 15;
        tmxTile = tmxLayer.getTMXTileAt(tileCol*32 + 16,
        tileRow*32 + 16);
        tmxTile.setGlobalTileID(mWAVTMXMap, mOpenCoffinGID);
        openCoffins.add(openThis);
        int openTime = gen.nextInt(OPEN_RATE);
        mHandler.postDelayed(openCoffin, openTime);
        mHandler.postDelayed(closeCoffin, openTime+STAY_OPEN);
    }
};

private Runnable closeCoffin = new Runnable() {
    public void run() {
        int closeThis = openCoffins.get(0);
        openCoffins.remove(0);
        int tileRow = coffins[closeThis]/15;
        int tileCol = coffins[closeThis] % 15;
        tmxTile = tmxLayer.getTMXTileAt(tileCol*32 + 16,
        tileRow*32 + 16);
        tmxTile.setGlobalTileID(mWAVTMXMap, mCoffinGID);
        if (++mNumClosed > OPENS_PER_GAME) mGameOver(PLAYER_WINS);
    }
};

```

openCoffin() 所指的 Runnable 代码方法与原来的大致相同，不过这个修改之后的代码会安排用于关闭棺材的 closeCoffin() 任务在经过 STAY\_OPEN 所定的时间之后

执行。

closeCoffin() 任务的代码是新写的, 它将从 openCoffins 中取得第一个棺材的编号 (就算它已经因为玩家的触摸而关闭)。我们知道列表中的第一个棺材就是接下来将要被关闭的那个, 所以这里是唯一可以判断游戏是否结束的地方。

WAVActivity.java 代码其余部分的修改都和上一节中针对 Level1Activity.java 所做的修改一样。

## 14.6 《愤怒的村民》

对于《愤怒的村民》这个游戏, 还是需要引入计分机制, 并定义游戏的结束:

- 计分的方式: 玩家通过使吸血鬼头像坠地而获得分数。每一次成功地将头像打落到地上可以获得 200 分。
- 游戏的结束: 在玩家成功地使吸血鬼头像坠地 (玩家获胜) 或者发射了一定数量的小木桩之后 (吸血鬼获胜), 游戏都会结束。

### 14.6.1 常量和字段

IV 小游戏需要增加的常量与字段如程序清单 14.15 所示。

程序清单 14.15 增加了计分功能之后的 IVActivity.java 代码中的常量与字段

```
// =====
// 常量
// =====

private static final int VAMPIRE_FLOORED = 200;
private static final boolean PLAYER_WINS = true;
private static final boolean VAMPIRES_WIN = false;
private static final int MAX_STAKES = 5;

// =====
// 字段
// =====

private Texture mPopUpTexture;
private TextureRegion mEndBackTextureRegion;
private TextureRegion mAgainButtonTextureRegion;
private TextureRegion mQuitButtonTextureRegion;
private TextureRegion mNextButtonTextureRegion;
private TextureRegion mNewHighTextureRegion;

private Texture mFontTexture;
private Font mFont32;
```

```

private ChangeableText mCurrScore;
private Sprite endBack, newHigh, againButton, quitButton,
    nextButton;

private int numStakes = 0;

private int numHeads = 0;
private ArrayList<Body> deadHeads = new ArrayList<Body>();
private SharedPreferences scores;
private SharedPreferences.Editor scoresEditor;
private int[] highScores = new int[5];
private int thisScore = 0;

```

上述代码中最值得一提的字段就是 `ArrayList<Body> deadHeads`。它用来保存已经坠地的吸血鬼头像数。注意，吸血鬼头像在坠地之还会弹起，从而多次与地面相碰撞，然而代码只需要考虑首次碰撞即可。

`onLoadEngine()` 方法与 `onLoadResources()` 方法的修改与 `Level1Activity.java` 和 `WAVActivity.java` 中的大致相同，这里就不再详述了。

## 14.6.2 onLoadScene 方法

回想一下，`onLoadScene()` 方法是小游戏中使用关卡加载程序来载入关卡的地方。游戏需要知道已经加载的吸血鬼头像数量，以及已经落地的物体数量，代码通过使用 ID 标签来实现这个功能。吸血鬼头像的 ID 是“vamp”，而地板的 ID 标签则是“floor”。程序清单 14.16 与 14.17 列出了需要做的改动。

程序清单 14.16 增加了计分功能之后 `IVActivity.java` 代码中 `onLoadScene()` 方法的第 1 部分

```

@Override
public Scene onLoadScene() {
    ...
    bkLoader.registerEntityLoader(TAG_BODY,
        new IBKEntityLoader() {
            @Override
            public void onLoadEntity(final String pEntityName,
                final Attributes pAttributes,
                final String pValue) {
                if (mShape.equals(TAG_SHAPE_VALUE_SQUARE)) {
                    ...
                    final Body mBody =
                        PhysicsFactory.createBoxBody(
                            mPhysicsWorld, bodyShape,
                            mBodyType,
                            PhysicsFactory.createFixtureDef(
                                mDensity, mElasticity,
                                mFriction));
                    mBody.setUserData(mID);
                }
            }
        }
    );
}

```

```

    . . .
        } else if (mShape.equals(
            TAG_SHAPE_VALUE_CIRCLE)) {
    . . .

    final Body mBody =
        PhysicsFactory.createBoxBody(
            mPhysicsWorld, bodyShape,
            mBodyType,
            PhysicsFactory.createFixtureDef(
                mDensity, mElasticity,
                mFriction));
        mBody.setUserData(mID);
    . . .
        } else if (mShape.equals(
    . . .

```

上述 onLoadScene() 方法的代码根据 XML 关卡文件中的数据来创建物体。这里只需要把存储在 mID 之中的 ID 值设定到每个物体的 UserData 中，以便在物体碰撞时能很容易地获取到它。

onLoadScene() 方法的另一处修改出现在靠后一些的地方，这部分代码用于解析需要载入的 ID 标签，如程序清单 14.17 所示。只要在解析中看到一个以“vamp”为值的 ID 标签，那么就将吸血鬼头像的数量加 1。

程序清单 14.17 增加了计分功能之后 IActivity.java 代码中 onLoadScene() 方法的第 2 部分

```

        bkLoader.registerEntityLoader(TAG_PHYSICSANDID,
            new IBKEntityLoader() {
                @Override
                public void onLoadEntity(final String pEntityName,
    . . .

                    mID = trimQuotes(physTokens[3]);
                    if (mID.equals("vamp")) numHeads++;
                }

```

### 14.6.3 onLoadComplete 方法

IV 小游戏的另一处改动则是扩充了 onLoadComplete() 方法代码中对 beginContact() 方法的覆写代码，修改之后的代码在吸血鬼头像落地时会增加玩家的分数。修改之后的代码如程序清单 14.18 所示。

程序清单 14.18 增加了计分功能之后 IActivity.java 代码中的 onLoadComplete() 方法

```

@Override
public void onLoadComplete() {

    this.mPhysicsWorld.setContactListener(

```

```

new ContactListener() {
    @Override
    public void beginContact(Contact contact) {
        Body bodyA = contact.getFixtureA().getBody();
        Body bodyB = contact.getFixtureB().getBody();
        String idA = (String)bodyA.getUserData();
        String idB = (String)bodyB.getUserData();
        if ((idA.equals("vamp")) && (idB.equals("floor"))) {
            if (!deadHeads.contains(bodyA)) {
                deadHeads.add(bodyA);
            }
            mAddScore(VAMPIRE_FLOORED);
            if (deadHeads.size() == numHeads) {
                mGameOver(PLAYER_WINS);
            }
        }
        if ((idB.equals("vamp")) && (idA.equals("floor"))) {
            if (!deadHeads.contains(bodyB)) {
                deadHeads.add(bodyB);
            }
            mAddScore(VAMPIRE_FLOORED);
            if (deadHeads.size() == numHeads) {
                mGameOver(PLAYER_WINS);
            }
        }
        public void endContact(Contact contact) {
        }
    }
};
}

```

当发生物理碰撞时，这段代码检测参与碰撞的两个物体是否一个是吸血鬼头像，而另一个是地板。当这种碰撞发生后，玩家的分数将会增加，同时代码检查是否所有的吸血鬼头像都已经落地。正如上一节中提到的那样，我们将每一个落地的吸血鬼头像加入到 `deadHeads` 中，这样就可以避免计算同一个吸血鬼头像多次弹地的情况了。

#### 14.6.4 addStake 方法

还有一个修改，它不同于所有小游戏中为了实现计分功能而增加的代码，此处修改用于保存已经发射的小木桩数量，并在最后一个木桩发射出去之后宣布游戏结束。该修改是在 `addStake()` 方法中增加了一行代码，如程序清单 14.19 所示。

程序清单 14.19 增加了计分功能之后 `IVActivity.java` 代码中的 `addStake()` 方法

```

private void addStake(final float pX, final float pY, float velX,
    float velY) {
    /* 如果玩家用完了所有的小木桩，那么游戏就结束了。 */
    if (numStakes++ > MAX_STAKES) mGameOver(VAMPIRES_WIN);
    . . .
}

```



做完这个修改之后，所有的碰撞与计分功能就实现完毕了，现在 V3 游戏的可玩性将有进一步的提升。在第 15 章介绍完一些 AndEngine 的扩展包之后，我们将在第 16 章彻底完成 V3 范例游戏的制作。

## 14.7 总结

本章讲解了两个互相关联的功能，那就是计分与碰撞检测。在 AndEngine 中有两种检测碰撞的方式，具体应该使用哪种取决于游戏中是否用到了 Box2D 物理引擎。本章还利用 AndEngine 所提供的绘制元素以及 Android 系统所提供的 SharedPreferences 存储 API 来构建了一套记录并显示玩家得分的机制。

## 14.8 习题

1. 读者可能注意到了，在本章下载代码中的“assets/mfx”文件夹下有一个名为 oof.ogg 的声音文件，修改 IVActivity.java 文件，使吸血鬼头像落地时，播放“oof”音效。
2. 很多游戏在玩家获得新的最高分时都会有明显的提示。修改三个小游戏的结果屏幕，当玩家分数超过所有高分榜上已有分数时，出现“New High Score”字样。该文本所用的图像已经包含在下载代码的 assets/gfx/scoring 文件夹下了。图 14.4 是一个带有最高分记录提示语的范例画面。

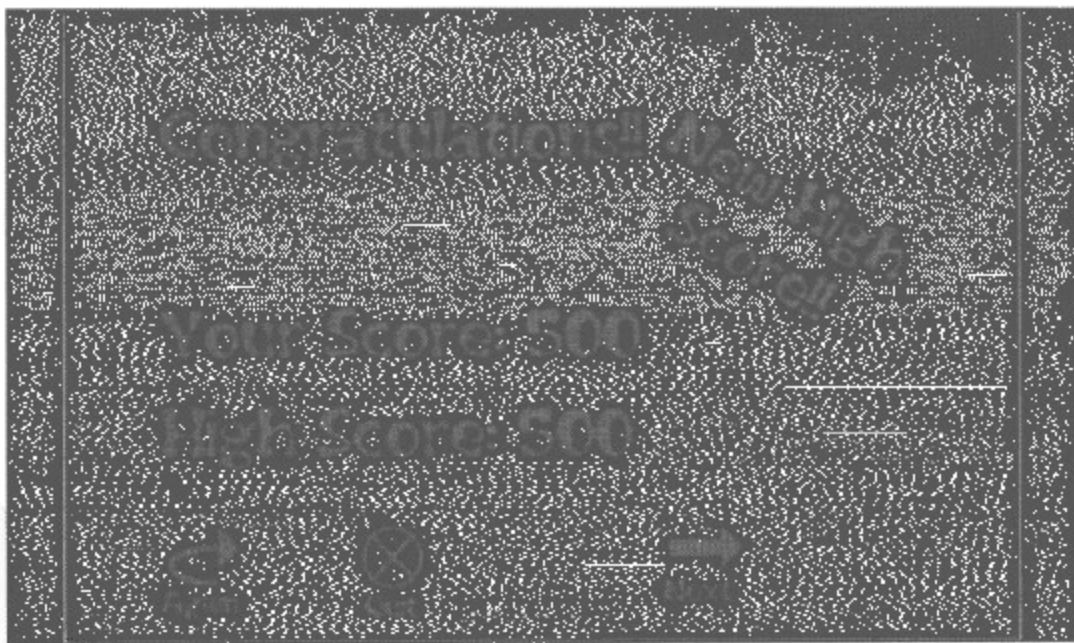


图 14.4 带有创造最高分记录提示语的游戏结果画面



## 第 15 章

# 多媒体扩展包

- 15.1 下载多媒体扩展包
- 15.2 动态壁纸
- 15.3 MOD 格式音乐
- 15.4 多人游戏
- 15.5 AndEngine 的多点触摸
- 15.6 增强现实游戏
- 15.7 总结
- 15.8 习题

AndEngine 是一个不断增长、不断演进的 Android 游戏引擎。当新功能开发出来后，它们将会以扩展包的形式引入到游戏开发中来。这些扩展包并不是 AndEngine 基本代码的一部分，不过它们可以通过下载而得到。所有扩展包的源代码都可以在以下网址获得：

<http://code.google.com/p/andengine/>

当本书写成时，AndEngine 共有 7 个扩展包，其中的某些已经在前面的代码中用到了：

- ❑ 动态壁纸：使开发者可以使用 AndEngine 来创建 Android 系统的动态壁纸
- ❑ MOD 文件播放器：开发者可以使用 XMP 播放器来播放 MOD 以及其他格式的音乐文件
- ❑ 多人游戏：在 IP 连接基础上，使用特制的多人游戏网络连接协议
- ❑ 多点触摸：利用支持多点触摸的 Android 设备所提供的功能
- ❑ Box2D 物理引擎：用于制作物理学游戏的引擎（第 12 章已经讨论过）
- ❑ 增强现实游戏：让开发者可以将 AndEngine 的 Scene 对象叠放在手机的摄像头预览框内
- ❑ 使用 SVG 图像创建 TextureRegion：支持可缩放的矢量图（第 5 章中已经讲过）

针对每一个未用到过的扩展包，本章都将使用一节的内容来讲解它。同时还会用 5 个短小的范例程序来示范每个扩展包的基本功能。如果读者感兴趣的话，可以深入研究它们。

## 15.1 下载多媒体扩展包

AndEngine 的每一个扩展包都有源码库。本书可供下载的代码中包含了每一个扩展包所对应的 .jar 文件，这些扩展包的版本都是本书写成时所发布的版本。当读者阅读本书时，更新的 .jar 文件可以在源码库的 lib 文件夹下找到。另外，也可以将整个源码库复制到开发电脑中，并使用第 12 章中所介绍的构建方式来自自己构建 .jar 文件。

每个扩展包的源码库网址如下：

<http://code.google.com/p/andengineivewallpaperextension/>

<http://code.google.com/p/andenginemodplayerextension/>

<http://code.google.com/p/andenginemultiplayerextension/>

<http://code.google.com/p/andenginemultitouchextension/>

<http://code.google.com/p/andenginephysicsbox2dextension/>

<http://code.google.com/p/andengineaugmentedrealityextension/>

<http://code.google.com/p/andenginesvgtextureregionextension/>

## 15.2 动态壁纸

AndEngine 的动态壁纸 (Live Wallpaper) 扩展包可以帮助我们利用 AndEngine 提供的所有游戏制作相关功能来创建 Android 系统的动态壁纸。开发者可以创建并显示精灵、图形、动画、修改器、粒子效果、文本、瓦片地图以及其他所有游戏制作中可以使用的东西。可以将这些对象制作成一张壁纸，充当 Android 系统主屏幕的背景。

可以从如下网址下载一份制作动态壁纸的 Activity 范例代码：

<http://code.google.com/p/andenginelifewallpaperextension/>

该壁纸模拟了一根燃烧的香烟，大量的烟气从烟头中冒出来。不论手机屏幕朝什么方向倾斜，烟总是向“上”冒。要运行这个范例程序，先将它下载下来，然后使用 Eclipse 将下载的文件创建为一个新的 Android 项目。创建这个项目，并运行它。然后通过 Android 手机主屏幕上的“Setting”菜单项，依次点击“Personalize”、“Home wallpaper”及“Live wallpapers”菜单项<sup>①</sup>，这样就会弹出一个用于切换手机壁纸的对话框。其中“Cigarette Live-Wallpaper”将作为诸多选项之一，出现在列表之中。图 15.1 显示了正在运行范例动态壁纸的主屏幕。



图 15.1 动态壁纸范例

### 15.2.1 Android 动态壁纸

Android 系统从 2.1 版本开始，就支持动态壁纸了，从我们的角度来看，这意味着运行 2.1 之前版本的 Android 手机将不能加载并使用动态壁纸。好在旧的手机或系统很快会被淘汰掉，所以这只是个临时性的问题而已。

动态壁纸是作为一个服务运行在 Android 系统后台的。如果读者想要创建动态壁纸的话，请参阅如下网址的文章：

<http://android-developers.blogspot.com/2010/02/live-wallpapers.html>

就像该文指出的那样，动态壁纸非常耗电，因为它们需要持续地运行。尤其需要注意的是，开发者必须在动态壁纸被另一个 Activity 挡住时停止其运行，否则，当另一个 Activity 占据了屏幕，动态壁纸程序依然在浪费着手机电池去绘制那些没人能看得见的图像。

<sup>①</sup> 在简体中文系统界面下依次是“设置”、“个性化设置”、“首页壁纸”及“动态壁纸”。——译者注

### 15.2.2 创建 V3 的 Android 动态壁纸

你可能希望以自己游戏中的元素来创建动态壁纸。作为示范，本书将使用 V3 游戏中的元素来创建一张动态壁纸，如图 15.2 所示，在这个壁纸中的小场景内，吸血鬼将会走过一片漆黑的墓地。



图 15.2 由 V3 中的元素所制作的动态壁纸

程序清单 15.1 列出了 V3LiveWallpaper.java 中的重要代码，完整的源文件可以在本章的下载代码中找到。

#### 程序清单 15.1 V3LiveWallpaper.java 代码节选

```

private Random gen;

// =====
// 覆写超类与接口中的方法
// =====

@Override
public org.anddev.andengine.engine.Engine onLoadEngine() {
    mHandler = new Handler();
    gen = new Random();
    return new org.anddev.andengine.engine.Engine(
        new EngineOptions(true, this.mScreenOrientation,
            new FillResolutionPolicy(), new Camera(0, 0,
                CAMERA_WIDTH, CAMERA_HEIGHT)));
}

@Override
public void onLoadResources() {
    TextureRegionFactory.setAssetBasePath("gfx/Wallpaper/");
    mBackgroundTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mBackgroundTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mBackgroundTexture, getApplicationContext(),
            "V3Wallpaper.png", 0, 0);
    mEngine.getTextureManager().loadTexture(
        this.mBackgroundTexture);
    mScrumTexture = new Texture(512, 256,
        TextureOptions.DEFAULT);
    mScrumTextureRegion =
        TextureRegionFactory.createTiledFromAsset(
            this.mScrumTexture, getApplicationContext(),
            "scrum_tiled.png", 0, 0, 8, 4);
    mEngine.getTextureManager().loadTexture(
        this.mScrumTexture);
    this.getEngine().getTextureManager().loadTexture(
        this.mScrumTexture);
}

@Override
public Scene onLoadScene() {
    final Scene scene = new Scene(1);

    // 载入背景
    Sprite bg = new Sprite(0, 0, mBackgroundTextureRegion);
    scene.getLastChild().attachChild(bg);

    // 加入第一个吸血鬼 (其他的也会随之加入)
    nVamp = 0;
    mHandler.postDelayed(mStartVamp, 3000);
}

```



```

scene.registerUpdateHandler(new IUpdateHandler() {
    @Override
    public void reset() { }

    @Override
    public void onUpdate(final float pSecondsElapsed) {
        for (int i=0; i<nVamp; i++){
            if (asprVamp[i].getX() < 30.0f){
                // 将吸血鬼移回屏幕右方
                float startY =
                    gen.nextFloat()*
                    (CAMERA_HEIGHT - 50.0f);
                asprVamp[i].clearEntityModifiers();
                asprVamp[i].registerEntityModifier(
                    new MoveModifier(40.0f,
                        CAMERA_WIDTH - 30.0f, 0.0f,
                        startY, 340.0f)
                );
            }
        }
    }
});

return scene;
}

// =====
// 方法
// =====
private Runnable mStartVamp = new Runnable() {
    public void run() {
        int i = nVamp;
        Scene scene = LiveWallpaperService.this.mEngine.getScene();
        float startY = gen.nextFloat()*(CAMERA_HEIGHT - 50.0f);
        asprVamp[i] = new AnimatedSprite(CAMERA_WIDTH - 30.0f,
            startY, mScrumTextureRegion.clone());
        nVamp++;
        scene.registerTouchArea(asprVamp[i]);
        final long[] frameDurations = new long[26];
        Arrays.fill(frameDurations, 500);
        asprVamp[i].animate(frameDurations, 0, 25, true);
        asprVamp[i].registerEntityModifier(
            new SequenceEntityModifier(
                new AlphaModifier(5.0f, 0.0f, 1.0f),
                new MoveModifier(60.0f,
                    asprVamp[i].getX(), 0.0f, startY,
                    340.f)
            )
        );
        scene.getLastChild().attachChild(asprVamp[i]);
    }
};

```

```

        if (nVamp < MAX_VAMPS) {
            mHandler.postDelayed(mStartVamp, VAMP_RATE);
        }
    }

    @Override
    public void onGamePaused() {
        mHandler.removeCallbacks(mStartVamp);
    }

    @Override
    public void onGameResumed() {
        mHandler.postDelayed(mStartVamp, VAMP_RATE);
    }
}

```

如果读者熟悉本书迄今为止的代码，那么将会看到，程序清单 15.1 中所列的代码大部分是照搬或稍加修改了 `Level1Activity.java` 中的代码，这部分代码被放在一个继承了 `BaseLiveWallpaperService` 的类当中。动态壁纸不需要检测碰撞或创建路径，只需要让吸血鬼从右向左移动即可。当它们到达屏幕左边时，`UpdateHandler` 会将其重新放置在屏幕的右边。

在所修改的这部分代码中，大部分都是为了适应不同的屏幕方向而改动的。与游戏中不同，壁纸通常是纵向（portrait orientation）显示的，然而对于大多数手机及运行 Honeycomb 或更新版本的 Android 平板电脑来说，都需要能够在屏幕方向发生改变时正确地在纵向与横向壁纸之间切换。调整绘图坐标以适应屏幕方向的改变，并不需要修改很多代码。

```

<service
    android:name="LiveWallpaperService"
    android:enabled="true"
    android:icon="@drawable/icon"
    android:label="@string/service_name"
    android:permission=
        "android.permission.BIND_WALLPAPER">
    <intent-filter android:priority="1" >
        <action android:name=
            "android.service.wallpaper.WallpaperService" />
    </intent-filter>
    <meta-data
        android:name="android.service.wallpaper"
        android:resource="@xml/wallpaper" />
    </service>
</application>

</manifest>

```

---

## 15.3 MOD 格式音乐

在听说 AndEngine 的 MOD 播放器扩展包之前，笔者并不熟悉 MOD 音乐格式，后来才发现，在数码艺术家的领域里面，有这么一种亚文化，该群体中的人专门创作一些能够用计算机来演示与播放的非商业性质音乐样品（Demo）。这些作曲者力求制作出最具艺术性且最能利用运行设备机能的音乐样品来。他们钟爱的音频格式之一就是 MOD，此格式是由 20 世纪 80 年代风靡一时的 Amiga 电脑<sup>⊖</sup>上的同名格式派生而来。制作与播放这些音乐样品的程序叫做“tracker”或“modplayer”，这些程序也可以在 Android 手机上运行。

### 15.3.1 搜寻 MOD 格式的音乐

很多作曲者都会在网上社区发布作品，有一个流行的网站叫做 Demoscene，它的网址是 [www.demoscene.info](http://www.demoscene.info)。相关的 MOD 文件存档可以在如下网址中找到：

<http://modarchive.org>

可以找到共计上百兆字节的音乐样本，它们都可以用合适的 tracker 来播放，此外很多老游戏的音乐都可以找到 MOD 或其他扩展包所支持的文件格式。

如果要在商业游戏中包含这些文件则需小心，大部分 MOD 文件的许可都声明可以允许以个人目的对其再度使用与分发，然而对于商业性的使用（例如包含在游戏中），则需要获得原作者的明确授权。有些 MOD 文件在使用 XMP 之类的播放器（下一节将

---

⊖ Amiga（非正式译名为阿米加）是 Amiga 公司开发的个人电脑产品系列。——译者注

讲到) 进行播放时, 可以显示出作者的联系方式来, 另外一些则不能, 还有一些音乐文件的信息已经过时了。关于从网上下载 MOD 文件的使用授权问题, 可以在下列网址找到详细的说明:

<http://modarchive.org/index.php?faq-licensing>

### 15.3.2 XMP MOD 播放器

AndEngine 所提供的 MOD 扩展包是利用名为 XMP 的播放器来播放 MOD 文件的。XMP 可以在 Android Market 上找到, 读者不妨将其安装到手机上, 用其播放一些 MOD 音乐文件看看效果。图 15.3 显示了正在播放音乐文件的 XMP 播放器界面。屏幕中间的条状图形是以动画形式显示的音乐频率。

XMP 不仅仅是一个 MOD 文件播放器, 它可以播放超过 90 种音频文件格式, 其中的很多格式最初都是在电子游戏中使用的。XMP 自己进行解码与播放的工作, AndEngine 扩展包可以让游戏代码通过 Native Development Kit 来访问 XMP 播放器, 并向其下达指令。并不需要在目标手机上安装 XMP 播放器, 因为它的代码已经包含在扩展包的 .jar 库文件中了。虽说网上有大量的 MOD 文件可以免费下载, 但是这并不意味着可以在游戏中合法地使用它们, 像其他有知识产权的物品一样, 需要阅读文件所附的许可协议来确定是否可以在商业游戏中使用它。

XMP 播放器有一个地方很有意思, 那就是它要求所播放的声音文件必须存放在 Android 手机的 SD 存储卡上。如果游戏项目的 assets 文件夹下有一个 .mod 音乐文件, 那么如何将其放在 SD 卡上, 以便 XMP 播放器能找到它呢? 幸好 Nicolas 提供了一些可用于解决此种情况的文件操作工具函数, 并附上了一个使用 MOD 文件的例子, 以展示如何使用这些函数。范例程序的源文件可以在如下网址找到:

<http://code.google.com/p/andengineexamples/source/browse/src/org/anddev/andengine/examples/ModPlayerExample.java>

程序清单 15.3 将范例代码中移动文件的相关代码做了简化。

程序清单 15.3 将文件移动至 SD 存储卡

```
private final ModPlayer mModPlayer = ModPlayer.getInstance();
```

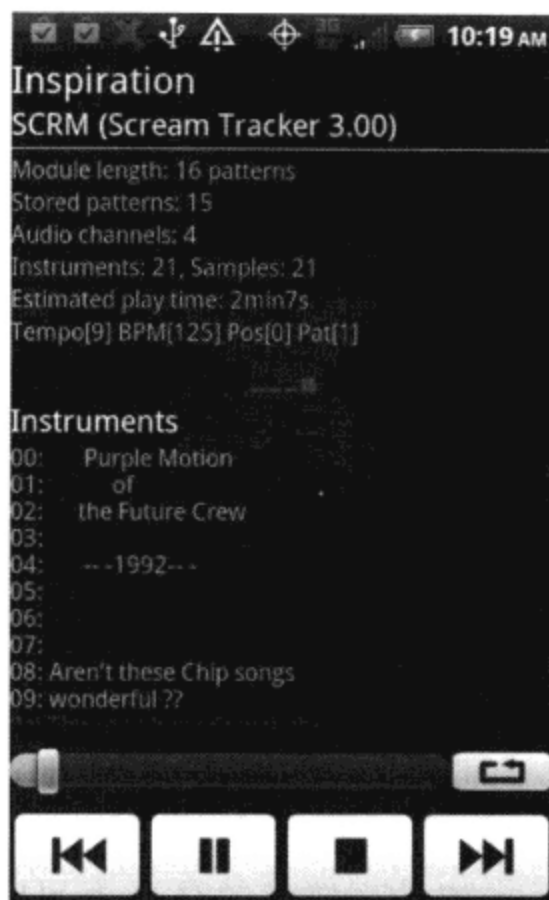


图 15.3 XMP 播放器界面

```

    . . .
    if (FileUtils.isFileExistingOnExternalStorage(this,
        "mfx/8bit.mod")) {
        this.startPlayingMod();
    } else {
        this.doAsync(R.titleResourceID, R.messageResourceID,
            new Callable<Void>() {
                @Override
                public Void call() throws Exception {
                    FileUtils.ensureDirectoriesExistOnExternalStorage(
                        ModPlayerExample.this, "mfx/");
                    FileUtils.copyToExternalStorage(
                        ModPlayerExample.this, "mfx/8bit.mod",
                        "mfx/8bit.mod");
                    return null;
                }
            }, new Callback<Void>() {
                @Override
                public void onCallback(final Void pCallbackValue) {
                    ModPlayerExample.this.startPlayingMod();
                }
            });
    . . .
    // =====
    // 方法
    // =====

    private void startPlayingMod() {
        this.mModPlayer.play( FileUtils.getAbsolutePathOnExternalStorage(
            this, "mfx/8bit.mod"));
    }
    . . .

```

如果文件已经存在于 SD 卡中，那么只需要调用 `startPlayingMod()` 方法，通过 `ModPlayer` 对象将文件全名传递给 XMP 播放器即可。否则，将会执行一个相对于游戏逻辑的异步操作，操作代码使用工具函数确保目标文件夹已经存在，然后从 `assets/mfx` 目录中将文件复制到目标位置。当复制操作结束后，将会调用 `startPlayingMod()` 方法，该方法的代码会将要播放的音乐文件名传递给 XMP 播放器。

## 15.4 多人游戏

另一个 `AndEngine` 扩展包提供了多人游戏中不同设备之间的基本通信能力，该扩展包所用的方法是使用 `stream socket`（流式套接字）。在设备之间创建“客户端/服务器”关系。如果读者并不熟悉起源于 TCP/IP 通信的套接字技术，那么不妨将其想象为某设备上以数字命名的虚拟端口。套接字的用法之一就是用来交换数据流，交换双方

一致协定使用某套固定的端口号来发送与接收消息，例如收发电子邮件常用的 POP3 与 SMTP 协议，就使用了“广为人知”的固定端口号 110 及 25。

使用此扩展包时，其中一台 Android 手机需要扮演服务器的角色，而另一台则被当成为客户端。两台设备可以通过 Wi-Fi、WAN（包括 2G、3G、EDGE、GPRS 等方式）甚至蓝牙来自由地交换信息。

Nicolas 写了一个程序，很好地演示了这个扩展包在多个设备之间传递信息的能力，读者可以在如下网址找到它：

<http://code.google.com/p/android-engineer-examples/source/browse/trunk/engineserver/>



户端手机的屏幕上，也会在同一位置出现同样的脸谱。

## 15.5 AndEngine 的多点触摸

AndEngine 的多点触摸扩展包可以让开发者在游戏中使用多点触摸手势，当然了，使用这项技术的前提是游戏所运行的 Android 手机本身支持多点触摸。在这项硬需求上，没有变通的办法能够绕过去，运行 2.0（代号 Éclair）或更新版本系统的手机大多都支持多点触摸。

此扩展包极大地简化了多点触摸技术的使用。在第 8 章讨论触摸输入方式的时候，我们讲到，Android 系统在每个独立的触摸点上都会同时发出以 ACTION\_POINTER\_DOWN、ACTION\_POINTER\_MOVE 或 ACTION\_POINTER\_UP 为参数的触摸事件。程序可以从事件对象中获取如下信息：

- ❑ int getPointerCount(): 返回触摸点的个数（触摸屏幕的手指数）
- ❑ float getX(int pointerIndex): 类似于单点触摸情况下的 getX() 方法
- ❑ float getY(int pointerIndex): 类似于单点触摸情况下的 getY() 方法

触摸指针<sup>⊖</sup>的索引坐标在不同事件中并不保证是一致的，例如，在某个事件中索引号为 1 的那个触摸指针在另一个触摸事件中所具有的索引就未必是 1。为了解决这个问题，Android 系统提供了指针 ID 用于在不同的触摸事件中维持指针身份的连续性，同时还在事件对象中提供了一些其他信息。不过如果开发者使用 AndEngine 扩展包来处理多点触摸的话，则不必担心这些问题。

程序清单 15.4 是由 Nicolas Gramlich 所写的 MultiTouchExample.java 范例代码改编而来。

程序清单 15.4 多点触摸范例代码，第 1 部分

```

    . . .
    @Override
    public Engine onLoadEngine() {
    . . .
        try {
            if (MultiTouch.isSupported(this)) {
                engine.setTouchController(
                    new MultiTouchController());
                if (MultiTouch.isSupportedDistinct(this)) {
                    mMultiTouchDistinct = true;
                } else {
                    mMultiTouchDistinct = false;
                }
            }
        }
    }

```

⊖ 即手指。——译者注

```

    } else {
        Toast.makeText(this,
            "Sorry your device does NOT support " +
            "MultiTouch!\n\n(Falling back to " +
            "SingleTouch.)", Toast.LENGTH_LONG).show();
    }
} catch (final MultiTouchException e) {
    Toast.makeText(this, "Sorry your Android " +
        "Version does NOT support MultiTouch!\n\n" +
        "(Falling back to SingleTouch.)",
        Toast.LENGTH_LONG).show();
}

return engine;
}

```

在运行程序之前，首先确认多点触摸扩展包所用的库文件 `andenginemultitouch-extension.jar` 已经像载入其他扩展包那样，被正确地加载到开发项目中了。`onLoadEngine()` 方法中的 `try-catch` 语句测试当前运行程序的手机与 Android 系统是否支持多点触摸。现在 Android 手机上有两种支持多点触摸技术的方式：

- ❑ 一种方式是仅支持由两个手指所做出的手势，例如通过挤压或拉伸屏幕来执行的缩放操作。这种情况下系统并不会记录每一个手指的动作。
- ❑ 另一种支持方式则是将每个手指的动作都独立记录下来。

程序清单 15.5 继续列出剩余部分的代码，其中匿名 `Sprite` 类的 `onAreaTouch()` 方法代码与早先单点触摸情况下的代码很相似。唯一的区别就在于这里多定义了一个名为 `mGrabbed` 的 `boolean` 型字段，它通过 `ACTION_DOWN` 事件类型来记录该精灵是否已经被“抓住”。如果需要使用多点触摸来同时移动多个精灵的话，那么使用该扩展包比使用 Android 系统的指针索引与 ID 要简单得多。

程序清单 15.5 多点触摸范例代码，第 2 部分

```

    final Sprite sprite = new Sprite(pX, pY,
        this.mTextureRegion) {
        boolean mGrabbed = false;

        @Override
        public boolean onAreaTouched(final TouchEvent
            pSceneTouchEvent,
            final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            switch(pSceneTouchEvent.getAction()) {
                case TouchEvent.ACTION_DOWN:

```

```
        this.mGrabbed = true;
        break;
    case TouchEvent.ACTION_MOVE:
        if(this.mGrabbed) {
            this.setPosition(
                pSceneTouchEvent.getX() - Card.CARD_WIDTH / 2,
                pSceneTouchEvent.getY() - Card.CARD_HEIGHT / 2);
        }
        break;
    case TouchEvent.ACTION_UP:
        if(this.mGrabbed) {
            this.mGrabbed = false;
            this.setScale(1.0f);
        }
        break;
    }
    return true;
}
```

---

## 15.6 增强现实游戏

“增强现实”（augmented reality）一词在手机游戏中具有不同的含义。本书所指的意思是将电脑生成的图像叠加在手机摄像头的取景窗口所看到的场景上。图 15.5 显示了一幅简单的增强现实画面，图中的场景为吸血鬼正穿行于笔者家的后院中。



图 15.5 增强现实场景演示：穿行于笔者家后院中的吸血鬼

将 augmentedrealityextension.jar 引入到 Android 游戏项目之后, 就会发现这个 And-Engine 扩展包所增加的两个新类:

□ BaseAugmentedRealityGameActivity: 它继承自 BaseGameActivity 类, 内部组合了一个用于显示手机摄像头预览画面的 SurfaceView 对象。

□ CameraPreviewSurfaceView: 此类用于创建上面提到的那个 SurfaceView 对象。

此扩展包的使用相当简单, 程序清单 15.6 列出了制作吸血鬼穿越后院效果所需的关键代码。

程序清单 15.6 增强现实场景范例代码

```
package com.pearson.lagp.vinb;

...

public class VampiresInBackyard extends BaseAugmentedRealityGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;
    private static final int VAMP_RATE = 2000;
    private static final int MAX_VAMPS = 10;

    // =====
    // 字段
    // =====

    private Camera mCamera;
    private Handler mHandler;
    private Texture mScrumTexture;
    private TiledTextureRegion mScrumTextureRegion;

    private AnimatedSprite[] asprVamp = new AnimatedSprite[10];
    private int nVamp;

    private Random gen;

    // =====
    // 构造器
    // =====

    // =====
    // Getter 与 Setter 方法
    // =====

    // =====
    // 覆写超类及接口中的方法
    // =====
}
```

```
// =====

@Override
public Engine onLoadEngine() {
    Toast.makeText(this, "If you don't see a vampire moving " +
        "over the screen, try starting this while already being in " +
        "Landscape orientation!!", Toast.LENGTH_LONG).show();
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    mHandler = new Handler();
    gen = new Random();
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
}

@Override
public void onLoadResources() {
    TextureRegionFactory.setAssetBasePath("gfx/VinB/");
    mScrumTexture = new Texture(512, 256,
        TextureOptions.DEFAULT);
    mScrumTextureRegion =
        TextureRegionFactory.createTiledFromAsset(
            this.mScrumTexture,
            getApplicationContext(),
            "scrum_tiled.png", 0, 0, 8, 4);
    mEngine.getTextureManager().loadTexture(
        this.mScrumTexture);
    this.getEngine().getTextureManager().loadTexture(
        this.mScrumTexture);
}

@Override
public Scene onLoadScene() {
    final Scene scene = new Scene(1);
    scene.setBackground(
        new ColorBackground(0.0f, 0.0f, 0.0f, 0.0f));

    // 加入第一个吸血鬼 (其他的也会随之加入)
    nVamp = 0;
    mHandler.postDelayed(mStartVamp, 3000);

    scene.registerUpdateHandler(new IUpdateHandler() {
        @Override
        public void reset() { }

        @Override
        public void onUpdate(final float pSecondsElapsed) {
            for (int i=0; i<nVamp; i++){
                if (asprVamp[i].getX() < 30.0f){

```

```

        // 将吸血鬼重新放置在屏幕右方
        float startY = gen.nextFloat()*
(CAMERA_HEIGHT - 50.0f);
        asprVamp[i].clearEntityModifiers();
        asprVamp[i].registerEntityModifier(
            new MoveModifier(40.0f,
                CAMERA_WIDTH - 30.0f, 0.0f,
                startY, 340.0f)
        );
    }
}
});
return scene;
}

// =====
// 方法
// =====
private Runnable mStartVamp = new Runnable() {
    public void run() {
        int i = nVamp;
        Scene scene = VampiresInBackyard.this.mEngine.getScene();
        float startY = gen.nextFloat()*(CAMERA_HEIGHT - 50.0f);
        asprVamp[i] = new AnimatedSprite(CAMERA_WIDTH - 30.0f,
            startY, mScrumTextureRegion.clone());
        nVamp++;
        scene.registerTouchArea(asprVamp[i]);
        final long[] frameDurations = new long[26];
        Arrays.fill(frameDurations, 500);
        asprVamp[i].animate(frameDurations, 0, 25, true);
        asprVamp[i].registerEntityModifier(
            new SequenceEntityModifier (
                new AlphaModifier(5.0f, 0.0f, 1.0f),
                new MoveModifier(40.0f,
                    CAMERA_WIDTH - 30.0f, 0.0f, startY, 340.f)
            ));
        scene.getLastChild().attachChild(asprVamp[i]);
        if (nVamp < MAX_VAMPS){
            mHandler.postDelayed(mStartVamp, VAMP_RATE);
        }
    }
};
}

```

上述代码中的大部分内容都是从 V3LiveWallpaper.java 中复制并粘贴过来的, 同时 V3LiveWallpaper.java 的很多代码又是从 LevelActivity.java 中复制而来的。当开发者让自己的类继承自 BaseAugmentedRealityGameActivity 而非 BaseGameActivity 时, 就可



以自动将手机摄像头的预览画面叠加到游戏场景之上了。可以将任意的图像放置在预览画面之上，以创建自己的增强现实应用程序。

## 15.7 总结

本章复习了一遍 AndEngine 所支持的扩展包，碰巧的是，它们与多媒体技术都有这样或那样的关联。毫无疑问，随着 AndEngine 的持续发展壮大，会出现越来越多的扩展包，从而使游戏变得更加有趣，也更具交互性。

通常会在 AndEngine 的论坛中发布有关新扩展包的通知：

<http://www.andengine.org/forums/extensions/>

扩展包所用的源代码与二进制文件可以在 AndEngine 的 github 站点找到：

<http://code.google.com/p/andengine/>

## 15.8 习题

1. 修改 V3LiveWallpaper.java 文件，使吸血鬼到达屏幕左边界之后自动爆炸，效果如图 15.6 所示。

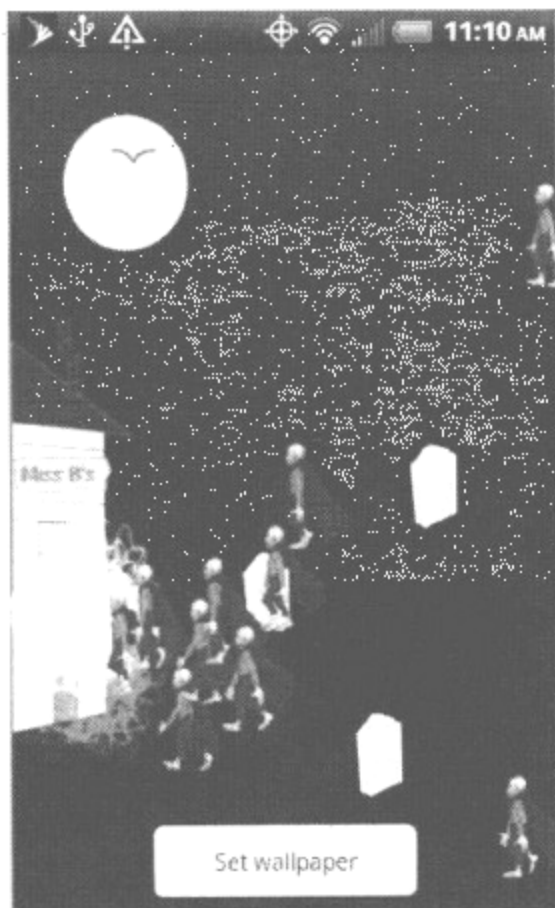


图 15.6 带有爆炸效果的 V3Live-Wallpaper 程序

2. 现在 V3 范例游戏中的《打吸血鬼》小游戏并没有播放背景音乐。修改 WAVActivity.java, 选择一首 MOD 格式的背景音乐来播放。

3. 修改 VampiresInBackyard.java 的代码, 在屏幕顶端显示当前手机所指的罗盘方向<sup>①</sup>。修改之后的运行效果如图 15.7 所示。



图 15.7 增强现实场景演示：穿行于后院中的吸血鬼（屏幕顶端显示罗盘方向）

<sup>①</sup> 以正北方向为 0 度，角度按顺时针递增。——译者注



## 第 16 章

# 游戏集成

- 16.1 困难度调节
- 16.2 游戏结束画面的代码
- 16.3 第 1 关：主游戏
- 16.4 《打吸血鬼》
- 16.5 《愤怒的村民》
- 16.6 选项菜单
- 16.7 总结
- 16.8 习题

到这里，本书已经将 AndEngine 游戏引擎的基本元素全部涵盖了。我们讨论并示范了如下主题：

- ☐ 基本的游戏逻辑循环
- ☐ 创建并显示菜单
- ☐ 创建场景并在场景间切换
- ☐ 创建 Sprite 对象并在其上附加修改器
- ☐ 制作带有动画效果的精灵
- ☐ 使用某种字型来绘制文本
- ☐ 获取用户输入
- ☐ 使用瓦片地图来当游戏桌布
- ☐ 构建并显示粒子效果
- ☐ 播放音乐及音效
- ☐ 使用 Box2D 物理引擎
- ☐ 运用人工智能技术
- ☐ 记录玩家的分数以便衡量其游戏水平
- ☐ 多媒体扩展包

在上述过程中，我们通过编写大量的代码，制作了一个名为 V3 的游戏，它的全称是 *Vampires Versus Virgins*（即少女大战吸血鬼）。然而，游戏制作过程中还是留下了一些小问题尚待解决。现在我们将这些问题一并处理，使 V3 游戏更具系统性和可玩性。

本章将讨论如下问题：

- ☐ 困难度调节：让每个小游戏都能够自由调节困难度，让玩家觉得游戏具有一定的挑战性，同时又有能力完成它。每个小游戏都有可供调节的参数，而且这些参数需要在游戏结束之后保存起来，以便下次还能使用。
- ☐ 游戏的结束画面：在第 14 章实现计分功能时，我们为每个小游戏创建了结束场景，并在游戏中显示玩家当前获得的分数，还实现了显示 5 个最高分的功能。当时曾提到，结果画面上有三个按钮，它们的功能留待以后实现。现在，我们就来实现这些按钮的功能：
  - ☐ Again：表示玩家想重玩当前这个小游戏。
  - ☐ Quit：表示玩家想彻底退出游戏。
  - ☐ Next：表示玩家想继续玩下一个小游戏。

本章将困难度和结束画面分别用一节来讲述，然后用三节的内容分别讲述对三个小游戏所做的修改。

## 16.1 困难度调节

困难度的调节如果做得好，可以让游戏更吸引玩家；如果做不好，则可能会令玩家对游戏望而生畏。如果游戏过于简单，玩家可能很快就感到厌倦，不想再玩它了；如果游戏过于困难，使玩家产生挫折感，那么玩家也会放弃这款游戏。所以困难度的调节要恰到好处才行。

更麻烦的是，玩家会在游戏过程中变得逐步熟练，所以困难度还应随着游戏的进行而改变。不管用哪种方案调节困难度，都需要将其存储起来，以便下次运行游戏时还能继续使用这些数值。一旦玩家熟悉某款游戏之后，就不想每次打开游戏都要从简单难度玩起了。

### 16.1.1 困难度参数的保存

每个小游戏的困难度参数都需要保存起来，以便下次运行时还能访问到。对于复杂的游戏来说，合适的做法也许是将其保存在 SQLite 数据库中，不过对于 V3 这种仅有几个参数的游戏来说，有点儿小题大做。

所以，还是向咱们的老朋友 SharedPreferences 求助吧。新建一个名为“difficulty”的配置文件，每个小游戏都将困难度参数存于其中，Android 系统会维护这些数据，以便每次运行时都能访问到它们。

程序清单 16.1 演示了如何在每个小游戏中使用 SharedPreferences。

程序清单 16.1 使用 SharedPreferences 来存储困难度参数的通用代码

```

...
difficulty = getSharedPreferences("difficulty", MODE_PRIVATE);
diffEditor = difficulty.edit();
mParam1 = difficulty.getInt("GAMELET.PARAM1", 1000);
mParam2 = difficulty.getInt("GAMELET.PARAM2", 1);
mWins = difficulty.getInt("GAMELET.WINS", 0);
mPlays = difficulty.getInt("GAMELET.PLAYS", 0);
...
private void mSaveDifficulty() {
    diffEditor.putInt("GAMELET.PARAM1", mParam1);
    diffEditor.putInt("GAMELET.PARAM2", mParam2);
    diffEditor.putInt("GAMELET.WINS", mWins);
    diffEditor.putInt("GAMELET.PLAYS", mPlays);
}
...

```

在小游戏的开始处，也就是 onLoadEngine() 方法中，将困难度参数存入 SharedPreferences 之中。上述代码中的 GAMELET 将用每个小游戏的具体名称来替换，PARAMn 将用每个困难度参数的具体名称来替换。在游戏结束的时候，调用 mSaveDifficulty() 方

法将这些有可能已经更新过的参数再写回 SharedPreferences。

### 16.1.2 困难度参数的设定

有了保存困难度参数的机制之后，接下来要考虑的就是如何来设定困难度了。这个问题看起来很简单，然而将你的全部职业生涯都耗在上面，也未必能够将它做好。要调整好困难度，需要请玩家来实测游戏。可以根据他们的游戏效果来调整游戏的困难度，在收集与分析了这些数据之后，可以通过多种手段来调整游戏的难度：

- 邀请尽可能多的目标玩家参与游戏测试，并收集他们的游戏结果。将结果数据进行相关性比较，并根据这些反馈数据来设定一个让大部分玩家都觉得满意的困难度。
- 在游戏进行过程中调整困难度。虽说仍然要进行初始困难度的设定，不过在游戏运行过程中可以根据玩家的输赢情况来动态地调整困难度。V3 就将使用这种方式。
- 根据一定范围内的玩家反馈数据，来远程设定游戏的困难度。几乎所有的 Android 手机都能上网，所以开发者可以使用网络连接来收集玩家的游戏数据，并据此设定游戏困难度。数据收集与远程设置需要玩家授予游戏程序访问与修改手机数据的权限，不过这项技术非常强大，尤其是在社交游戏中。

为了演示我们所采用的这套困难度调整方案，游戏需要将玩家进行游戏的次数与获胜的次数记入 SharedPreferences 中。在游戏结束时，可以根据这些信息以及玩家是否在本轮游戏中取胜，来调整这个小游戏下一次运行时的困难度。读者也许还会考虑使用诸如“最近一次连续游戏失败的次数”（recent losses in a row）等数据来调整困难度，不过为了演示困难度的调整，使用输赢次数就足够了。说句玩笑话，有些游戏可能会收集重力加速计的反馈数据，看看玩家因为游戏受挫而把手机摔了多少次。

程序清单 16.1 中的代码将游戏次数与获胜次数保存在 SharedPreferences 中，很显然，失败次数可以通过将两者相减而得出。这里可以使用比 int 更大的数据类型，不过就算每秒钟玩一次游戏，也可以记录 68 年的游戏数据，所以这里用整型变量足够了。

每个小游戏都将有一个名为 mIncreaseDifficulty() 的方法，用于增加游戏困难度。提升游戏困难度的逻辑可以做得很复杂，不过为了演示范例游戏，本书会将它做得非常简单。每一个小游戏中用于调整困难度的具体代码将在本章稍后的各节中分别讲述。

## 16.2 游戏结束画面的代码

在玩家按下 Again 按钮之后，游戏需要重新开始。有两种实现方式：

- 将还未执行的 Runnable 任务都取消掉，重新初始化各种用于记录游戏当前状态的计数器，并且调用重启游戏的方法。
- 向系统投递一个用于重新执行当前 Activity 的任务，然后调用 finish() 方法，使当前的 Activity 结束执行。



第一种方式也许可以很好地利用手机资源。贴图、粒子效果与关卡信息都不需要重新载入，许多其他类型的对象也不需要重新创建，代码只需要将各个对象的状态都正确地初始化即可。第二种方式则更加简单，因为重新初始化的工作会由 Runnable 任务来完成。

本章的范例代码将使用第二种方式，在末尾的习题中，将要求读者使用第一种方式来改写。可以参阅附录中的习题解答来查看第一种方式的实现代码。

程序清单 16.2 列出了所有 V3 小游戏都将使用的处理游戏结束的代码。

程序清单 16.2 用于处理游戏结束的通用代码

```

    againButton = new Sprite(0.0f, 0.0f, mAgainButtonTextureRegion){
        @Override
        public boolean onAreaTouched(final TouchEvent pAreaTouchEvent,
            final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            switch(pAreaTouchEvent.getAction()) {
            case TouchEvent.ACTION_DOWN:
                mEndCleanup();
                mHandler.post(mPlayThis);
                finish();
                break;
            }
            return true;
        }
    };

    nextButton = new Sprite(0.0f, 0.0f, mNextButtonTextureRegion){
        @Override
        public boolean onAreaTouched(final TouchEvent pAreaTouchEvent,
            final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            switch(pAreaTouchEvent.getAction()) {
            case TouchEvent.ACTION_DOWN:
                mEndCleanup();
                mHandler.post(mPlayNext);
                finish();
                break;
            }
            return true;
        }
    };

    quitButton = new Sprite(0.0f, 0.0f, mQuitButtonTextureRegion){
        @Override
        public boolean onAreaTouched(final TouchEvent pAreaTouchEvent,
            final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            switch(pAreaTouchEvent.getAction()) {
            case TouchEvent.ACTION_DOWN:

```

```

        mEndCleanup();
        finish();
        break;
    }
    return true;
}

};

...
private void mEndCleanup() {
    mPlays++;
    if (mPlayerWon) {
        mIncreaseDifficulty();
        mWins++;
    }
    mSaveDifficulty();
}

private Runnable mPlayThis = new Runnable() {
    public void run() {
        Intent myIntent = new Intent(ThisActivity.this,
            ThisActivity.class);
        ThisActivity.this.startActivity(myIntent);
    }
};

private Runnable mPlayNext = new Runnable() {
    public void run() {
        Intent myIntent = new Intent(ThisActivity.this,
            NextActivity.class);
        ThisActivity.this.startActivity(myIntent);
    }
};

```

各按钮中的 `onAreaTouched()` 方法处理 `ACTION_DOWN` 分支所用的代码是相似的：

- ❑ 某些用于处理游戏结束的代码已经提取到 `mEndCleanup()` 方法中了，尽管也可以将它们放在 `mGameOver()` 方法中，不过这种方式使得代码更加简洁。
- ❑ 不论是要重新运行当前 Activity 还是要启动下一个游戏的 Activity，都将执行代码封装在 `Runnable` 任务中，并投递给系统。

- ❑ 调用 `finish()` 方法以结束当前 Activity 的执行。

`mEndCleanup()` 方法做了三件事：

- ❑ 将玩家运行小游戏的次数加 1。
- ❑ 如果玩家本轮游戏获胜，那么就调用 `mIncreaseDifficulty()` 方法增加困难度（调节困难度的具体方式请阅上一节的内容），并且将玩家获胜次数加 1。
- ❑ 将困难度参数写回 `SharedPreferences` 中。

名为 `mPlayThis` 的 `Runnable` 任务将会重启当前的 Activity（程序清单 16.2 中的

ThisActivity 将使用具体小游戏 Activity 的名称来替换), 而名为 mPlayNext 的 Runnable 任务则会启动下一个将要运行的小游戏。

## 16.3 第 1 关: 主游戏

现在的主游戏(也就是游戏第 1 关)是将两个困难度以常量形式保存的, 需要修改代码, 将其变为可以调整数值的变量, 以便同 SharedPreferences 中的数值关联起来。同时还需要再增加一个用于干扰玩家的参数。

- ❑ mMaxVamps: 它就是原来名为 MAX\_VAMPS 的那个常量。它表示游戏中某一时间屏幕上出现的吸血鬼总数的最大值。
- ❑ mVampRate: 它是原来名为 VAMP\_RATE 的那个常量, 用于指定生成两个吸血鬼的最大时间间隔。
- ❑ mDistract: 这个新加入的变量用于表示是否在游戏中开启干扰效果。干扰效果可以做得很复杂, 不过 V3 游戏的干扰效果就是每隔一段时间, 会从 Miss B 家窗口伸出一个正在求救的女孩动画头像而已。干扰效果如图 16.1 所示。

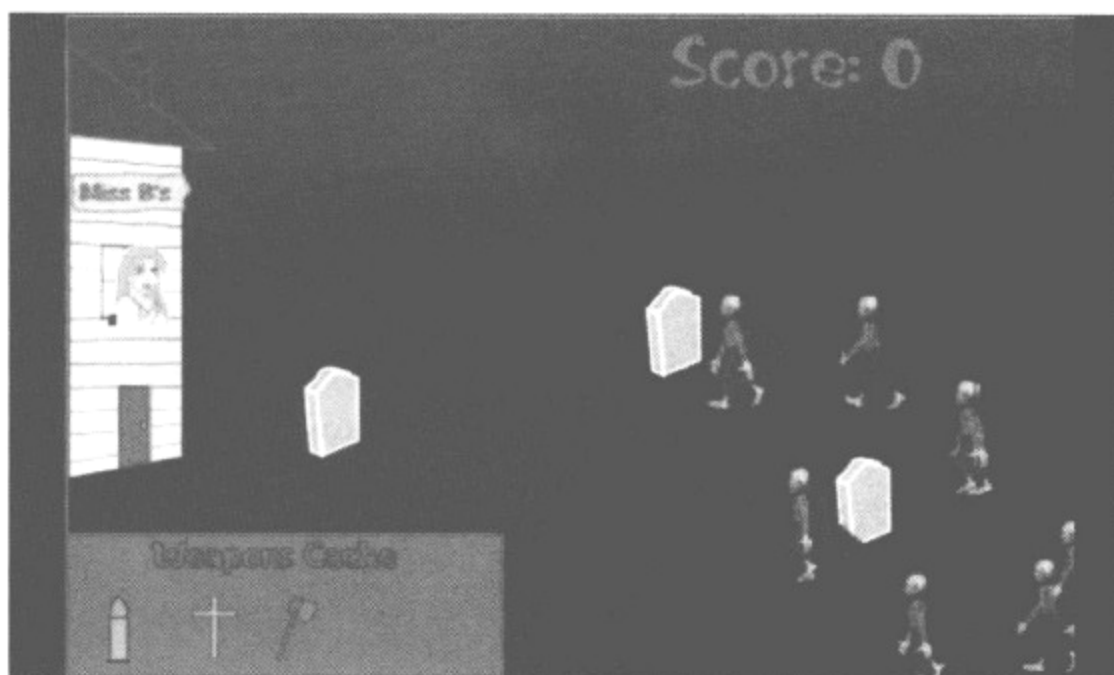


图 16.1 游戏第一关中从 Miss B 家窗口伸出少女头像以干扰玩家

程序清单 16.3 列出了 Level1Activity.java 代码中的几处重要修改。

程序清单 16.3 加入了游戏结束处理与困难度调节功能的 Level1Activity.java 代码

```
package com.pearson.lagp.v3;

+imports

public class Level1Activity extends BaseGameActivity {
```

```

// =====
// 字段
// =====

private Texture mSarahTexture;

private TiledTextureRegion mSarahTextureRegion;

private AnimatedSprite[] asprVamp = new AnimatedSprite[40];
private AnimatedSprite asprSarah;

private Sound mSaveMeSound;

private SharedPreferences difficulty;
private SharedPreferences.Editor diffEditor;

private AStar[] aStar = new AStar[40];
private Path[] pathVamp = new Path[40];
private int mWins, mPlays;
private int mMaxVamps;
private int mVampRate;
private boolean mDistract;
private boolean mPlayerWon;
private boolean mActivityVisible = true;

@Override
public Engine onLoadEngine() {

    difficulty = getSharedPreferences("difficulty",
        MODE_PRIVATE);
    diffEditor = difficulty.edit();
    mMaxVamps = difficulty.getInt("Lv11.MAX_VAMPS", 10);
    mVampRate = difficulty.getInt("Lv11.VAMP_RATE", 4000);
    mDistract = difficulty.getBoolean("Lv11.DISTRACT", true);
    mWins = difficulty.getInt("Lv11.WINS", 0);
    mPlays = difficulty.getInt("Lv11.PLAYS", 0);

@Override
public void onLoadResources() {

    mSarahTexture = new Texture(256, 64,
        TextureOptions.DEFAULT);
    mSarahTextureRegion =
        TextureRegionFactory.createTiledFromAsset(
this.mSarahTexture, getApplicationContext(),
        "sarahanim.png", 0, 0, 6, 1);
    mEngine.getTextureManager().loadTexture(
this.mSarahTexture);

```

```

    . . .
    SoundFactory.setAssetBasePath("mfx/");
    try {
    . . .
        this.mSaveMeSound = SoundFactory.createSoundFromAsset(
            this.mEngine.getSoundManager(),
            getApplicationContext(), "saveme.ogg");
    . . .
    @Override
    public Scene onLoadScene() {
    . . .

        // 如果开启了干扰效果, 那么就启动第一个用于干扰的动画
        if (mDistract) {
            mHandler.postDelayed(mStartSarah, 5000);
        }

    . . .
    // 结束画面的按钮处理和程序清单 16.2 一样
    . . .

        asprSarah = new AnimatedSprite(15.0f, 90.0f,
            mSarahTextureRegion);
        asprSarah.setVisible(false);
        scene.getLastChild().attachChild(asprSarah);
        return scene;
    }

    . . .
    @Override
    public void onGamePaused() {
    . . .
        mSaveMeSound.stop();
        mActivityVisible = false;
    }

    @Override
    public void onGameResumed() {
        super.onGameResumed();
        mActivityVisible = true;
    }

    . . .
    private Runnable mStartSarah = new Runnable() {
        public void run() {
            final long[] frameDurations = new long[6];
            Arrays.fill(frameDurations, 200);
            asprSarah.setVisible(true);
            asprSarah.animate(frameDurations, 0, 5, false);
            playSound(mSaveMeSound);
            mHandler.postDelayed(mStartSarah,
                (long) (gen.nextFloat() * 7000.0f + 8000.0f));
        }
    };

```

```

        mHandler.postDelayed(mEndSarah, 2000);
    }
}

private Runnable mEndSarah = new Runnable() {
    public void run() {
        asprSarah.setVisible(false);
    }
}

private void playSound (Sound mSound) {
    if ((audioOptions.getBoolean("effectsOn", false)) &&
        (mActivityVisible)) {
        mSound.play();
    }
}

private void mIncreaseDifficulty() {
    // 让游戏变得稍微困难一点
    if (mMaxVamps < 40) mMaxVamps += 5;
    if (mVampRate > 500) mVampRate -= 500;
    if (mWins > 5) mDistract = true;
}

private void mSaveDifficulty() {
    diffEditor.putInt("Lvl1.MAX_VAMPS", mMaxVamps);
    diffEditor.putInt("Lvl1.VAMP_RATE", mVampRate);
    diffEditor.putBoolean("Lvl1-DISTRACT", mDistract);
    diffEditor.putInt("Lvl1.WINS", mWins);
    diffEditor.putInt("Lvl1.PLAYS", mPlays);
}

private void mEndCleanup() {
    mPlays++;
    if (mPlayerWon) {
        mIncreaseDifficulty();
        mWins++;
    }
    mSaveDifficulty();
}

private Runnable mPlayThis = new Runnable() {
    public void run() {
        Intent myIntent = new Intent(Level1Activity.this,
            Level1Activity.class);
        Level1Activity.this.startActivity(myIntent);
        finish();
    }
}

```



```

    };

    private Runnable mPlayNext = new Runnable() {
        public void run() {
            Intent myIntent = new Intent(Level1Activity.this,
                WAVActivity.class);
            Level1Activity.this.startActivity(myIntent);
            finish();
        }
    };
}

```

像上一节所述，这段代码从 `SharedPreferences` 对象中载入困难度参数，并于稍后将其写回。按钮逻辑的处理代码和程序清单 16.2 中的一样，所以这里不再重复。尽管方法参数不同，不过方法名 `mIncreaseDifficulty()` 与 `mSaveDifficulty()` 在所有小游戏的代码中都是一样的。

如果读者还不知道的话，我来告诉你，那个正在求救的少女名叫 Sarah，笔者在 Content Paradise 网站花 3.99 美元购买了她头像的使用权。

这段代码没有什么新的内容，像加载吸血鬼精灵那样，将 Sarah 的头像载入到 `TiledTextureRegion` 对象中，并且像加载其他音效那样，将表示呼救声的音效载入到游戏中来。如果游戏运行时开启了干扰效果，那么就投递一个 `Runnable` 任务，以显示第一个 Sarah 动画头像，在等待一段时间之后，该任务代码会再次显示动画效果。在创建 Sarah 精灵的时候，将其设定为不可见，然后根据是否需要向玩家显示干扰效果而切换其可见性与动画效果。

在 `onGamePaused()` 与 `onGameResumed()` 方法中，有一处小修改值得注意。这两个方法所修改的那个 `Boolean` 变量用于表示当前游戏的 `Activity` 是否可见，`playSound()` 方法将根据这个值来决定是否需要播放 Sarah 的呼救声。如果没有这个标志变量，那么就算当前游戏已经被切换至后台，Sarah 还是会一直呼救。

另一个需要提一下的地方就是代码将保存吸血鬼相关信息的数组长度扩容为 40，这也是 `mMaxVamps` 所能取的最大值。其实应该将这个值提取为常量，这样整个小游戏中只需修改一处代码即可一次性地改变这个值了。

名为 `mPlayThis` 与 `mPlayNext` 的 `Runnable` 任务的代码已经在程序清单 16.3 中列出了，这里所写的版本比程序清单 16.2 中的那个更为具体。`mPlayThis` 任务会将当前的 `Activity` 重启，而 `mPlayNext` 任务则会按照如下顺序来切换下一个将要运行的小游戏：

- 《墓地》（游戏第一关）
- 《打吸血鬼》（Whack-A-Vampire）
- 《愤怒的村民》（Irate Villagers）

## 16.4 《打吸血鬼》

WAV 游戏中有三个常量与游戏难度参数有关，这一次还是将它们都修改为变量，这样就可以调整其数值，并将其写入名为“difficulty”的 SharedPreferences 配置信息中以便保存：

- ❑ **mOpenRate**：它就是原来名为 OPEN\_RATE 的常量。该参数指定了两个打开的棺材之间所经的最长时间间隔。增加这个参数的值将使游戏变得简单，而减少它的值则会使游戏更加困难。
- ❑ **mStayOpen**：它是原来名为 STAY\_OPEN 的那个常量，该值指定了棺材保持打开状态的最大时间。同上一个参数一样，增加它游戏则变得简单，而减少它游戏则变得困难。
- ❑ **mOpenPerGame**：它是原来名为 OPENS\_PER\_GAME 的那个常量，它表示该轮游戏中棺材所打开的总次数。在一轮游戏中棺材打开的次数增多意味着游戏更加困难，因为玩家必须在更长的时间段内集中注意力以进行游戏，反之，如果打开棺材的次数变少，则游戏会变得简单。

程序清单 16.4 列出了加入游戏结束处理与难度调节功能之后的 WAVActivity.java 代码。

程序清单 16.4 加入了游戏结束处理与难度调节功能的 WAVActivity.java 代码

```
package com.pearson.lagp.v3;

+imports

public class WAVActivity extends BaseGameActivity {
    . . .

    // =====
    // 字段
    // =====

    . . .

    private int mOpenRate;
    private int mOpensPerGame;
    private int mStayOpen;
    private int mWins, mPlays;
    private boolean mPlayerWon;

    . . .

    // =====
    // 覆写超类与接口中的方法
    // =====

    @Override
```

```

public Engine onLoadEngine() {
    . . .
    difficulty = getSharedPreferences("difficulty",
        MODE_PRIVATE);
    diffEditor = difficulty.edit();
    mOpenRate = difficulty.getInt("WhAV.OPEN_RATE", 4000);
    mStayOpen = difficulty.getInt("WhAV.STAY_OPEN", 2000);
    mOpensPerGame = difficulty.getInt("WhAV.OPENS_PER_GAME",
        10);
    mWins = difficulty.getInt("WhAV.WINS", 0);
    mPlays = difficulty.getInt("WhAV.PLAYS", 0);
    . . .
    // 结束画面的按钮处理和程序清单 16.2 一样
    . . .
    private Runnable openCoffin = new Runnable() {
        public void run() {
            . . .
            mHandler.postDelayed(openCoffin, openTime);
            mHandler.postDelayed(closeCoffin, openTime+mStayOpen);
        }
    };

    private Runnable closeCoffin = new Runnable() {
        public void run() {
            . . .
            if (++mNumClosed > mOpensPerGame)
                mGameOver(PLOYER_WINS);
        }
    };

    private void mGameOver(boolean pWin){
        . . .
        if (pWin){
            mPlayerWon = true;
            scene.setChildScene(mCreateEndScene(newTop,
                "Congratulations!!"), false, true, true);
        } else {
            mPlayerWon = false;
            scene.setChildScene(mCreateEndScene(false,
                "You Suck! \n. . . .blood"));
        }
    }

    . . .
    private void mIncreaseDifficulty() {
        // 让游戏变得稍微困难一点
        if (mOpenRate > 1000) mOpenRate -= 1000;
        if (mStayOpen > 500) mStayOpen -= 200;
        if (mOpensPerGame < 50) mOpensPerGame += 10;
    }
}

```

```

private void mSaveDifficulty() {
    diffEditor.putInt("WhAV.OPEN_RATE", mOpenRate);
    diffEditor.putInt("WhAV.STAY_OPEN", mStayOpen);
    diffEditor.putInt("WhAV.OPENS_PER_GAME", mOpensPerGame);
    diffEditor.putInt("WhAV.WINS", mWins);
    diffEditor.putInt("WhAV.PLAYS", mPlays);
}

...
}

```

## 16.5 《愤怒的村民》

在类似《愤怒的村民》这种物理学游戏中，关卡的困难度通常取决于关卡的设计。所以说，能够随着游戏的运行而调整的困难度参数就会稍微少一些。仍然可以调整的两个参数分别是：

- mMaxStakes：这个变量用于指定玩家击打杂草堆中的吸血鬼头像使之落地所能用的最大木桩数。在早前版本的代码中，它以名为 MAX\_STAKES 的常量表示。
- mMaxTime：原来版本的游戏，对玩家完成任务所花时间没有限制，不过现在我们通过限制玩家过关的时间来给其一点压力。

程序清单 16.5 列出了加入游戏结束处理与困难度调节功能之后的 IVActivity.java 代码。

程序清单 16.5 加入了游戏结束处理与困难度调节功能的 IVActivity.java 代码

```

package com.pearson.lagp.v3;

+imports

public class IVActivity extends BaseGameActivity implements
    IOnSceneTouchListener, BKConstants {
    // =====
    // 常量
    // =====
    ...
    private static final int NUM_LEVELS = 4;
    ...
    // =====

```

```

// 字段
// =====

...

private ChangeableText mCurrScore, mTimeLeftTxt;
private float mTimeLeft;

...

private boolean mPlayerWon;
private int mWins, mPlays;
private int mMaxStakes;
private int mMaxTime;
private boolean [] mLevelComplete = new boolean[NUM_LEVELS];
private int mCurrentLevel;
private Handler mHandler;

...

@Override
public Engine onLoadEngine() {

...

    difficulty = getSharedPreferences("difficulty",
        MODE_PRIVATE);
    diffEditor = difficulty.edit();
    mMaxStakes = difficulty.getInt("IV.MAX_STAKES", 10);
    mMaxTime = difficulty.getInt("IV.MAX_TIME", 90);
    mTimeLeft = mMaxTime;
    for (int i=0; i<NUM_LEVELS; i++){
        mLevelComplete[i] =
            difficulty.getBoolean("IV.LEVEL"+i, false);
    }
    mWins = difficulty.getInt("IV.WINS", 0);
    mPlays = difficulty.getInt("IV.PLAYS", 0);

...

}

@Override
public void onLoadResources() {

...

    // 显示完成本关所剩时间
    mTimeLeftTxt = new ChangeableText(0.2f*CAMERA_WIDTH, 10.0f,
        mFont32, "Secs: 0", "Secs: XXX".length());
    mScene.getLastChild().attachChild(mTimeLeftTxt);

...

    mScene.registerUpdateHandler(new IUpdateHandler() {

```

```

@Override
public void reset() { }

@Override
public void onUpdate(final float pSecondsElapsed) {
    mTimeLeft -= pSecondsElapsed;
    if (mTimeLeft < 0){
        mGameOver(false);
    }
    mTimeLeftTxt.setText("Time: " +
        (int)mTimeLeft);
}
});

return mScene;
}

...

private void mIncreaseDifficulty() {
    // 让游戏变得稍微困难一点
    if (mMaxStakes > 1) mMaxStakes -= 1;
    if (mMaxTime > 20) mMaxTime -= 10;
}

private void mSaveDifficulty() {
    diffEditor.putInt("IV.MAX_STAKES", mMaxStakes);
    diffEditor.putInt("IV.MAX_TIME", mMaxTime);
    for (int i=0; i<NUM_LEVELS; i++){
        diffEditor.putBoolean("IV.LEVEL"+i, mLevelComplete[i]);
    }
    diffEditor.putInt("IV.WINS", mWins);
    diffEditor.putInt("IV.PLAYS", mPlays);
    diffEditor.commit();
}
}

```

除了加入困难度参数之外,《愤怒的村民》游戏中又有了两个新功能:

- ❑ 游戏可以记录玩家所通过的关卡,并在运行时根据关卡的完成情况来决定下一关载入的关卡。这些关卡文件以“iv0.lvl”、“iv1.lvl”的方式来命名。游戏的最大关卡数保存在名为 NUM\_LEVELS 的常量中。
- ❑ 屏幕上显示了倒数计时。如果玩家在时间耗尽之前未能完成游戏,那么游戏就会结束并判定玩家失败。每关所限定的游戏时间作为一个困难度参数,会根据上一



次游戏结束之后的情况来进行调整。

第一个功能是通过一个 Boolean 数组来实现的，它记录下了玩家是否已将某一关成功地完成，并将其值写入到保存困难度参数的 SharedPreferences 中。当游戏启动时，会先查看这个数组以决定接下来要加载的究竟是哪一关。当玩家成功地通过某个关卡时，游戏代码会修改相应的 Boolean 值，并将其提交到 SharedPreferences 中。

时间显示功能的实现代码与显示分数的代码类似，只不过现在使用了一个新建的 UpdateHandler 对象来更新时间。为了编程方便，每个 UpdateHandler 对象在更新时都可以通过回调方法的参数得知这次更新距离上一次的时间间隔，以便由此推算出玩家完成关卡所剩的时间。如果时间耗尽，那么就以 false 为参数调用 mGameOver() 方法，表示玩家在本关游戏中失败了。

## 16.6 选项菜单

现在既然已经将三个小游戏串联起来了，那么按理说就可以删除 Option 菜单中的“Whack-A-Vampire”与“Irate Villagers”菜单项了。然而，调试游戏的时候，为了方便，不想一关一关地玩下去，而是需要能够直接运行某个小游戏，所以我们就不修改选项菜单的内容了。

## 16.7 总结

现在 V3 游戏基本上做好了，虽然还需要做一些清理工作使代码更加整洁，更易维护，不过从功能上讲，游戏已经可以邀请玩家进行实际测试了。

本章所讲的技术并不局限于 AndEngine。将游戏难度参数化并对其进行调整的这套机制是使用纯 Java 代码配合 Android 系统 API 来做的。而重新运行游戏与继续运行下一个游戏这种功能则是完全使用 Android 系统的 startActivity() 方法来构建的。

现在游戏差不多就可以发布了。在第 17 章中，将会讨论与游戏的测试和发布相关的工作，并正式让 V3 游戏公诸于世。

## 16.8 习题

1. 在 Level1Activity 的 onPause() 方法中，代码将 mActivityVisible 这个 Boolean 变量的值设为 false，这样在 V3 游戏于后台运行时，playSound() 方法就不会再播放声音了。还有一种稍微好一些的方法，那就是取消所有还没有运行的 Runnable 任务，稍后在 onResume() 方法中再恢复它们。这样做可以节省用于投递和运行 Runnable 任务所

花的手机 CPU 资源。修改 `Level1Activity.java` 的代码,使得 V3 游戏在后台运行时,所有的 `Runnable` 任务都停止执行。

2. 修改 `OptionsActivity.java` 文件,使得选项菜单中的“Whack-A-Vampire”与“Irate Villagers”菜单项不再出现,而是在启动画面的右边多出两个隐藏按钮,分别用于直接启动这两个小游戏。

3. 修改 `MainMenuActivity.java` 文件,当用户选择主菜单中的“Help”菜单项时,游戏会切换到显示帮助信息的场景中去。



## 第 17 章

# 测试与发行

- 17.1 应用程序商业模式
- 17.2 测试与发行准备工作
- 17.3 发行游戏
- 17.4 推广游戏
- 17.5 总结

V3 范例游戏的功能部分几乎都做好了（笔者希望读者自己的游戏也是如此），现在可以将其发布了。不过 V3 游戏还有一个功能待实现，那就是需要支持游戏内付费（in-app purchase）。在增加了这个功能之后，我们就需要对游戏进行彻底的测试了，在测试过程中尽可能多地找出 Bug。当玩家将游戏下载下来并试玩之后，他们将会赞叹游戏这款游戏非常好玩，并且非常流畅。如果总是出现“强制关闭”这种错误的话，玩家可能就不会再玩这款游戏了。

## 17.1 应用程序商业模式

在深入讲述每种赢利方式的具体技术实现细节之前，花几分钟时间看看如何为游戏做市场规划。稍后将有一节内容来讲游戏的推广问题，这里先来看看如何从自己制作优秀手机游戏的努力中获得回报。

表 17.1 列出了几种常见的商业模式，并总结了其主要的优缺点。没有哪个模式是对所有人都适用的，所以说要根据每种商业模式的优劣来决定自己所用的方案（或者提出自己独特的商业模式）。就像表 17.1 中所述，有些模式需要在游戏中加入代码，另外一些则不需要。

表 17.1 Android 游戏的商业模式

商业模式	优点	开销
免费	可以获得最为广泛的受众，可以用来提振自己的名声	低。只有发行游戏所需的开销
手机广告	每次用户点击广告之后，都可以获得一定的赢利	中等。需要在游戏中加入展示广告的代码，广告也会占据屏幕空间
免费增值（Freemium）： 免费应用程序 + 程序内付费	可以通过免费的基本版应用程序获得非常广泛的受众，同时通过收费插件获取一定的赢利	中等。需要加入用于管理程序内付费功能的代码
付费应用	当用户下载并安装游戏时，即可立即获得赢利	低。只有发行游戏所需的开销

表 17.1 并没有将所有商业模式全部列出来，因为每天都有许多新的商业模式出现。不过，现在的大部分应用程序都使用了上表中的一种或几种商业模式。举例来说，经常会见到这种游戏，它的基本版是免费的，而增强版则需要付费下载。

本书所讲的 V3 游戏并不是用来赚钱的，所以我们选择以免费下载的形式发布。如果读者对在游戏中植入广告比较感兴趣，那么可以查阅本章末尾列出的众多手机广告公司。以下网址概述了如何将 Google 所提供的 AdMob 广告服务集成入游戏：

<http://code.google.com/mobile/ads/docs/android/fundamentals.html>

如果读者对程序内付费感兴趣，那么也可以参阅 Google 在 Android SDK 中所举的

一个范例应用程序及其概述，网址如下：

[http://developer.android.com/guide/market/billing/billing\\_overview.html](http://developer.android.com/guide/market/billing/billing_overview.html)

## 17.2 测试与发行准备工作

有一个无法回避的事实，那就是：测试工作是单调乏味的。实际上，很多软件开发者都将测试看成一件很烦人的事情。测试人员需要扮演用户的角色，一遍又一遍地输入相同的命令，以找出错误，并在问题得到修正之后再次测试程序以确保运行效果无误。这样做很快人就使人厌倦了。

当然，软件开发也能够为我们带来很多乐趣。本书的开头曾提到，正如解决谜题一样，对我们大多数人来说，软件开发也是很有趣的，因为它给了我们解决问题的快乐感。测试的过程就是去发掘这些谜题，而且针对测试的规划，本身就是个有趣的问题。

我知道此刻读者在想什么，你会认为一次又一次地重复测试过程，依然是非常令人讨厌的，我也不想在这个问题上同你争论，不过测试对一款游戏的成功来说，绝对是个关键问题。在将游戏交给玩家进行测试之前，必须尽可能多地找出 Bug。与此同时，在正式发布游戏之前，也必须使邀请来的游戏测试者尽可能多地找出 Bug。

Google 的开发者网站提供了一张在发行应用程序之前进行准备工作所需的核对表：

<http://developer.android.com/guide/publishing/preparing.html>

本书将核对表的内容根据游戏制作的需要转述如下，并在转述的过程中考虑到了如何在 Android Market 之外的应用程序商店中发布游戏。

### 17.2.1 在实际设备上测试游戏

这一点怎么强调都不过分：您需要在用户所用的手机上实测游戏。Android SDK 附带的模拟器确实很好，不过在 PC 或 Mac 电脑上使用鼠标来模拟触摸手势显得并不真实。如果你手边没有 Android 手机，而且想用模拟器做单元测试，这从理论上来讲是没问题的，不过笔者从来都不这样做，而是将一部用于开发的 Android 手机一直连接在我的开发电脑上，并且在它上面测试与调试。

然而，仅仅有一部 Android 手机也是不够的，各种 Android 手机的配置千差万别。它们运行着不同版本的操作系统；有着不同类型的处理器；有些有图形处理器，有些没有；有些具有高分辨率的大屏幕，有些则是低分辨率的小屏幕；有些只支持单点触摸，另一些则具有先进的多点触摸支持。您需要在尽可能多的手机上进行游戏测试。与 iOS 或 Windows Phone 7 这样的平台相比，有一点好处就是在将应用程序提交到软件商店之前，可以通过电脑反复地将程序上传到手机进行测试。

我们大多数人都没有足够的资金将所有型号的 Android 手机都买到手。不过可

以找一些拥有 Android 手机的朋友，并邀请他们来做游戏的测试玩家。当然了，需要将游戏的 APK 安装文件发给他们，不过安装文件并不需要签名，只要他们在手机上打开了相应的选项（选择主画面设置菜单中的“Settings”菜单项，并依次选择“Application”中的“Unknown Sources”菜单项<sup>①</sup>），同时通过 adb 程序向手机上传 APK 安装文件即可。

### 17.2.2 考虑加入终端用户许可协议

由于法律原因，为了在发行时确定开发者的责任，应该在游戏中包含终端用户许可协议（End User License Agreement, EULA）。下载并安装游戏的用户将会看到这个协议，同时协议下方有两个用于表示接受或拒绝的按钮。笔者不是律师，也不打算提供法律咨询，不过读者可以在如下网址找到一份 Android 应用程序所用的范例 EULA 文件：

<http://code.google.com/p/apps-for-android/source/browse/trunk/DivideAndConquer/assets/EULA?r=93>

你也可以在本章的下载代码中找到少女大战吸血鬼游戏中所用的 EULA 文件。

需要在游戏中加入一些代码，以便在第一次启动游戏时显示 EULA 文件。玩家可以选择接受或者拒绝。如果玩家接受协议的话，游戏将会开始运行，并且不会再次询问玩家了。如果玩家拒绝，游戏就会立即退出。图 17.1 显示了 EULA 协议对话框。程序清单 17.1 中列出了用于显示该协议的代码，这段代码同时会在 SharedPreferences 对象中记录游戏是否已经显示过该协议。



图 17.1 EULA 协议显示对话框

程序清单 17.1 加入 showEULA 方法之后的 StartActivity.java 代码

```
@Override
```

① 以上英文菜单项在简体中文界面中分别是“设置”、“应用程序”与“未知源”。——译者注



```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    showEULA();
}
...
public void showEULA() {
    eula = getSharedPreferences("eula", MODE_PRIVATE);
    boolean eulaShown = eula.getBoolean("shown", false);
    if(eulaShown == false){
        String title = this.getString(R.string.app_name);
        String message = this.getString(R.string.updates) + "\n\n" +
            getString(R.string.eula);

        AlertDialog.Builder builder = new AlertDialog.Builder(this)
            .setTitle(title)
            .setCancelable(false)
            .setMessage(message)
            .setPositiveButton(getString(R.string.accept),
                new Dialog.OnClickListener() {
                    @Override
                    public void onClick(
                        DialogInterface dialogInterface, int i) {
                        // 将协议标记为已阅读。
                        eulaEditor = eula.edit();
                        eulaEditor.putBoolean("shown", true);
                        eulaEditor.commit();
                        dialogInterface.dismiss();
                    }
                })
            .setNegativeButton(getString(R.string.decline),
                new Dialog.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog,
                        int which) {
                        finish();
                    }
                })
            );
        builder.create().show();
    }
}

```

`onCreate()` 方法在调用 `super.onCreate()` 方法之后，紧接着调用了 `showEULA()` 方法，该方法使用 `AlertDialog` 类创建了一个包含更新信息与 EULA 协议文本的对话框，并将其显示出来。更新信息与协议文本都包含在 `strings.xml` 文件之中。代码在 `AlertDialog` 对象上以 `false` 为参数调用 `setCancelable()` 方法，以防止用户通过手机上的 `back` 键绕过该对话框。

### 17.2.3 向 Manifest 文件加入图标与标签

需要为 V3 游戏设定一个用于在应用程序启动器 (Application Launcher) 菜单中显示的图标。我们就使用第 15 章中创建动态壁纸所用的那个图标吧。这里创建了三个不同大小的图标, 以便在三种具有不同类型 dpi 的 Android 设备上使用。图标文件都以 icon.png 命名, 并分别放置于 res/ 文件夹下的三个资源文件夹中, 如表 17.2 所示。三个图标的样子如图 17.2 所示。



图 17.2 V3 游戏图标

表 17.2 V3 游戏所用的图标文件

文件夹	图标大小
drawable-hdpi	128 像素 × 128 像素
drawable-mdpi	64 像素 × 64 像素
drawable-ldpi	32 像素 × 32 像素

### 17.2.4 关闭记录与调试功能

V3 游戏代码并未大量地使用记录与调试功能 (至少本书中所开发的 V3 游戏版本是这样的)。如果读者在开发游戏的过程中确实使用了大量的记录与调试语句的话, 那么最好将其套在条件语句块内, 这样就可以很轻松地开关这些语句了。不应该在玩家面前显示调试语句, 而且也不应该浪费磁盘空间去记录一些玩家根本不会看的信息。

在下载下来的 V3 代码中, 读者也许注意到有一个地方确实需要明确地关闭调试功能, 那就是在 AndroidManifest.xml 文件中。笔者开发游戏所用的这部 Android 手机需要打开 manifest 文件中的 debuggable 属性, 方能通过 gdb 调试器来调试游戏。现在为了关闭调试功能, 需要修改 AndroidManifest.xml 文件中的一行, 修改后的该行内容如下:

```
<application android:icon="@drawable/icon" android:label="@string/app_name" android:debuggable="false">
```

### 17.2.5 在游戏中增加版本号

也许读者并不热衷于版本号这个概念, 不过对于 Android 应用程序及游戏来说, 版本号是至关重要的。尽管 Android 系统并不需要访问版本号信息, 不过 Android Market 将利用这个信息来告诉用户是否有更新版本的程序可供安装。

游戏的 manifest 文件中应该定义两个标识符:

- android:versionCode: 发行游戏时所填写的版本号, 它是一个单调递增的整数。建议开始发行游戏时, 将其设置为 1, 在以后每次发布重大更新或微小更新时, 再将其值递增 1。

- `android:versionName`：一个显示给用户的字符串，用于描述此次发行的版本。可以使用任意的版本号命名规则，不过开发者最常用的还是以“major.minor”（主版本号.次版本号）的格式来告诉用户当前程序的主版本号和次版本号分别是什么。V3 游戏 manifest 文件中有关版本的部分如程序清单 17.2 所示。

程序清单 17.2 V3 游戏的 Manifest 文件中与版本相关的部分

---

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.pearson.lagp.v3"
    android:versionCode="1"
    android:versionName="1.0">
```

---

```

[no]: yes

Generating 2,048 bit RSA key pair and self-signed certificate
(SHA1withRSA) with a validity of 10,000 days
    for: CN=Rick Rogers, OU=Portmobile Software, O=Portmobile Software,
L=Harvard, ST=MA, C=US
Enter key password for <V3_alias>
    (RETURN if same as keystore password):
[Storing V3-release-key.keystore]

```

---

运行 Keytool 工具之后，所生成的密钥会保存在密钥库（keystore）中，密钥与密钥库都是加密过的。别名（alias）用来在密钥库中引用某个特定的密钥，当稍后对最终的 .apk 文件进行签名时将会用到它。请不要忘记密码，也不要忘了创建密钥时所用的别名。建议在游戏的所有发行版本中都使用相同的加密证书，这样就可以一直使用此处指定的密码与别名了。“validity” 参数后面所跟的天数用于指定生成密钥的有效期。Google 要求所有私有密钥的有效期至少持续到 2033 年的 10 月 22 日。

### 17.2.7 编译与签名最终的 .apk 文件

将密钥保存至密钥库之后，只需经过几个对话框，就可以对一个 .apk 文件进行签明了。用鼠标右键点击游戏对应的 Android 项目，在弹出的菜单中选择“Android Tools”子菜单下的“Export Signed apk”菜单项，并在其后弹出的一系列对话框中按照如下信息填写相关的栏位：

- ☐ 密钥库的位置：Keytool 工具将在当前工作目录下创建密钥库。在上一小节的例子当中，V3-release-key.keystore 这个密钥库创建在 C:\Users\rick 文件夹下。
- ☐ 密码：对话框中需要输入两次密码，一次是密钥库的密码，一次是密钥的密码。
- ☐ 别名：上一小节的例子使用 V3\_alias 作为密钥的别名。对话框列出了密钥库中所有的别名，用户可以在其中进行选择。
- ☐ 目标文件：该选项用于指定签名后的 .apk 文件所存放的位置。

现在可以使用 adb 程序将 .apk 文件安装到 Android 手机上了。比如说，当前电脑只有一部通过 USB 连接的 Android 手机，那么就可以使用如下命令来进行安装：

```
adb -d install v3.apk
```

执行命令之后，游戏就会被安装到手机上。如果修改了游戏代码，需要重新安装签名后的程序，则应先将其卸载：

```
adb -d uninstall com.pearson.v3
```

### 17.2.8 测试最终的 .apk 文件

从签名 .apk 文件到正式在 Android Market 发布，这个过程中很有可能需要修改游

戏的代码。因为游戏马上就要公开发行了，所以非常有必要再仔细检查最后一遍，以确保所有的东西都能正常工作。将 .apk 文件安装在你所能找到的所有 Android 手机上，确认经过先前的测试过程之后，所有手机都能正确地运行游戏。如果找到了新问题，那么就将其修复，然后重复上一小节的签名过程即可，因为这次私有密钥已经创建好了，我们可以直接使用它。

## 17.3 发行游戏

可以发布 Android 应用程序的软件商店在不断地增多。其中两个最为流行的软件下载站点分别是：

- ❑ Android Market: Google 公司官方的 Android 软件商店

- ❑ Amazon App Store: Amazon 公司新推出的软件商店，目前仅在美国可以访问

在本书写成之时，Android Market 是最为普及的软件商店。几乎所有的 Android 手机都会预装 Android Market 程序。所以如果要发行游戏的话，应该首先考虑它。

许多 Android 手机用户也愿意从其他诸如 Amazon 等软件商店中下载应用与游戏。不管是由于购买软件很方便，还是由于“Free App of the Day”促销政策很实惠，Amazon 软件商店确实变得越来越流行了。不过用户毕竟还是要自己将 Amazon App Store 程序安装在其手机上——这可能是非官方软件商店所无法避免的缺陷。

从开发者的角度来看，在这两个软件商店上发售游戏都很简单（具体细节请看下面两个小节）。不过开发者所付的费用并不同，Android Market 一次性收取 25 美元“以鼓励高质量产品的开发”，Amazon App Store 则要从第二年开始，收取 99 美元的年费。

### 17.3.1 Android Market

Google 公司宣称，每天有超过 50 万台 Android 设备被激活，并且正在使用的 Android 设备数已达 100 万台。这些手机上几乎都装有 Android Market 程序，用户可以轻松地通过它来下载应用与游戏。开发者可以通过它将产品推向大量的受众，所以只要付出 25 美元即可在 Android Market 上发布游戏，这实在是令人难以置信的低价了。

关于 Android Market 的简介可以在如下网址查阅：

<https://market.android.com/publish/signup>

该文档告诉开发者，要想在 Android Market 上发布软件，必须完成以下三件事：

- ❑ 创建开发者简介。

- ❑ 向 Google 付注册费。

- ☐ 同意 Android Market 开发者发行协议 (Android Market Developer Distribution Agreement)。

开发者简介非常简单,甚至不需要填写名字与地址(不过付注册费时需要填写这些信息)。此处需要提供电子邮件地址与电话号码,以便 Android Market 可以联系到你,Google 承诺不会将此信息提供他人。

25 美元的注册费只能通过 Google Checkout 来支付,它支持所有普通的信用卡。

Android Market 开发者发行协议是一个必须遵守的法律文件。笔者不是律师,也不会提供法律咨询。读者在签署协议之前最好咨询懂法律知识的相关人士,不过话说回来,如果要在 Android Market 上发布产品,则必须同意此协议才行。

完成了上述三个步骤之后,几分钟内你就会被授权为一名正式的 Android Market 产品开发者了,同时你也可以通过开发者控制台来存取账户信息。现在,你已经可以上传新的应用与游戏了,此外还可以管理已经上传的产品。

在上传签名之后的 .apk 文件时,需要提供应用程序描述信息、一些运行时的截图、程序图标,还可以提供一些诸如宣传片之类的影音信息。Google 一直在不停地改变要求开发者填写的信息,而且很有可能继续修改,不过现在 Android Market 上的游戏只需提供下列信息即可:

- ☐ 该版本的游戏所用的语言。
- ☐ 游戏名称(最长 30 个字符)。
- ☐ 关于游戏的描述(最多 4000 个英文字符)。
- ☐ 游戏最近版本所做的修订(最多 500 个英文字符)。
- ☐ 一些宣传性的文字(最多 80 个英文字符)。
- ☐ 在“应用程序”(Application)与“游戏”(Game)中做出选择。该选项决定了 Android Market 将把应用程序展示在哪个类别下面。
- ☐ 假如选择的是“游戏”,那么有 6 个子类型可选:
  - ☐ Arcade & Action
  - ☐ Brain & Puzzle (益智类游戏<sup>①</sup>)
  - ☐ Cards & Casino (纸牌游戏)
  - ☐ Casual (休闲游戏)
  - ☐ Racing (比赛)
  - ☐ Sports Games (体育游戏)
- ☐ 选择游戏的内容分级:
  - ☐ High maturity (心智成熟度 - 高)
  - ☐ Medium maturity (心智成熟度 - 中)

① 括号中为简体中文界面下的名称,下同。——译者注



- ☐ Low maturity (心智成熟度 - 低)
- ☐ Everyone (所有人)
- ☐ 定价。
- ☐ 游戏所发行的国家。
- ☐ 联系信息, 例如网站、电子邮件、电话号码等。
- ☐ 两个需要开发者同意的条款:

1. 承诺游戏符合 Android 内容准则 (Android Content Guidelines), 这份文件的内容还是相当合理的。不过需要重申, 笔者不是律师, 所以请自己确认其内容 (<http://www.android.com/us/developer-content-policy.html>)。

2. 承诺愿意承担美国法律中规定的有关游戏出口所应负的责任。

当填写好上述信息之后, 按下 “Publish” 按钮, 你的游戏就与广大用户见面了。

如果游戏是收费的或者包含程序内付费的机制, 那么还需要有 Google Merchant 账号。注册该账户并不难 (在 “Edit Application” 页面中有用于注册账户的链接), 不过还是会牵涉到其他一些法律合同, 所以最好请专业人士来帮你核查一番。

### 17.3.2 Amazon App Store

Amazon App Store 是一个发布 Android 应用的新型平台, 它的缺点是其潜在的游戏玩家数量相对 Android Market 来说要少很多。

它的好处是其上的应用与游戏数量相对来说要少一些, 这意味着新游戏能够更容易被玩家注意到。另一个好处是 Amazon 对其所接收的应用内容比较灵活。Google 的开发人员计划政策 (Developer Program Policy) 明确禁止包含裸露与现金赌博的内容, 而 Amazon 则只要求在内容分级信息中注明即可。

在 Amazon 上发布应用程序的过程与 Android Market 的类似:

- ☐ 访问 Amazon Appstore Developer Portal 页面, 此页面的网址很长, 而且可能在读者阅读本文时已经修改。所以请在网上直接搜索 “Amazon app store developer portal”, 这样就能找到它的网址了。
- ☐ 使用已有的 Amazon 账户登录, 或者新建一个账户。
- ☐ 需要同意一份法律协议。
- ☐ 如果计划在游戏中收费, 可以输入银行账户信息, 这样 Amazon 就会将钱打入这个账户中。
- ☐ 可以在 “MyApps” 页面上传应用程序。与 Android Market 一样, 所需填写的信息也是一直在改变, 不过现在要求填写的是如下信息:
  - ☐ 名称
  - ☐ 程序支持的设备布局: 可以是手机或者平板电脑, 也可以两者都支持
  - ☐ 如果需要像 big boy 相关产品那样记录应用程序的销售信息, 那么可以指定一

个应用程序 SKU (Stock Keeping Unit) 编码<sup>①</sup>

- ☐ 程序所支持的语言 (默认为英语)
- ☐ 可选填的联系信息, 包括电子邮件、电话、网址等
- ☐ 包含以下条目的商业信息区 (Merchandising Section):
  - ☐ 类别 (例如 “游戏”) 与子类别 (现有 17 个可供选择的子类别)
  - ☐ 一些关键词, 可以帮助用户搜到游戏
  - ☐ 游戏描述 (最多 4000 字)
  - ☐ 价格 (也可以是免费的)
  - ☐ 一系列日期, 包括首发 (original release) 日期, 在商店中可见 (visibility) 的截止日期, 程序可以使用 (availability) 的截止日期以及程序下架 (discontinuation) 的日期
- ☐ 含有如下选项的内容分级区 (Content Rating Section):
  - ☐ 广告 (Advertisements)
  - ☐ 文化偏见 (Culture Intolerance)
  - ☐ 动态内容 (Dynamic Content)
  - ☐ 裸露 (Nudity)
  - ☐ 与性有关的内容 (Sexual Content)
  - ☐ 酒精、烟草与毒品 (Alcohol, Tobacco, or Drugs)
  - ☐ 专为儿童设计
  - ☐ 赌博
  - ☐ 脏话
  - ☐ 暴力
- ☐ 多媒体上传区 (Upload Multimedia)

(promotion)这个词容易让人联想到狂欢节上招徕生意的小贩,或者是“万灵油”的推销员,不过为了让人们知道游戏,必须对其进行推广。

有很多推广手机游戏的手段,也有很多讲述此问题的专著。本节将会讲一些基本问题,不过还是强烈推荐读者尽可能地花些时间来阅读这方面的书籍与文章。其他手机平台上所用的推广理念原则上也适用于 Android 平台上的游戏。而且笔者还建议在推销游戏时要跳出固有的思维模式,想象一下当前有什么好主意可以将游戏推广到潜在的玩家之中。

### 17.4.1 App Store 推广

前面讲述 Android Market 与 Amazon App Store 的游戏提交时就曾提到,这两个软件商店都提供了一些推广手段,可以使玩家非常容易地找到并下载你的游戏。现在就来详细地看看这些推广手段。

#### 热门游戏 (Feature Apps)

两个软件商店都为其用户提供了热门游戏列表。很显然,能够进入某个热门列表对游戏来说好处很大,不过这意味着游戏必须进入如下列表之一:“热门付费游戏”(Top Paid)、“热门免费游戏”(Top Free)、“总收入最多游戏”(Top Grossing)、“最新热门付费游戏”(Top New Paid)及“最新热门免费游戏”(Top New Free)。Android Market 也有名为热门应用程序(其中也包含游戏)(Trending apps)与热门付费游戏(Best-Selling Games)的排行榜。此外还有一些并非像热门排行榜这种以数量来排序的推荐列表,这些列表中的程序都是由网站的编辑所选定的。开发者所能做的就是尽量让游戏引人注目,并且最好是能够吸引一名网站编辑来玩,以便其能够将游戏放入某个推荐列表当中。这种排行榜是没有办法通过给钱而挤进去的。

#### 游戏名称

花些时间来想一个能够让玩家记住的好名字。V3 游戏并不想成为那种一鸣惊人的杰作,不过如果真打算那么做的话,那么名称中包含类似“virgin”与“vampire”这样易于记住的名字能够让游戏卖得更好。最好的游戏名字是那些带有感情色彩的,能够吸引潜在玩家注意并且令人难以忘记的名字。Android Market 中的游戏名称最长 30 个字符,所以要充分利用这些空间。就像测试游戏那样,请朋友们帮忙在几个预定的游戏名称中挑选一个好一些的。

#### 关键词

供使用的分析工具，不过读者不要被这个困难拦住，Google AdWords 提供了免费的关键词工具，可以通过以下网址访问：

[https://adwords.google.com/o/Targeting/Explorer?\\_\\_u=10000000000&\\_\\_c=10000000000&ideaRequestType=KEYWORD\\_IDEAS#search.none](https://adwords.google.com/o/Targeting/Explorer?__u=10000000000&__c=10000000000&ideaRequestType=KEYWORD_IDEAS#search.none)

该工具所统计的关键词是针对网页搜索而非手机游戏搜索的，不过可以通过它来看看当前流行的关键词，以利于游戏推广。

### 游戏截图与图标

可能需要创建一个颇具吸引力的图标以使玩家下载你的游戏。你的游戏如果有幸出现在软件商店的推荐列表之中，那么程序图标将会显示在排行榜页面上。所以一定要制作好游戏图标。

上传游戏时所附的截图一定要将游戏最有趣的一面展示出来，潜在的玩家看到截图时会说“哇！这游戏看起来很有趣，我要下载下来玩玩”。上传截图的顺序也很重要，因为并非所有截图都能显示在游戏首页上，首先要展示那些最引人注目的截图。

### 游戏描述

软件商店中的游戏描述是真正需要做自我宣传的地方。广告写手所用的技巧之一就是关注于某个词汇，逐个句子地去描述它。举例来说，在广告文案中，第一句的唯一目的就是吸引读者去阅读第二句。反复锤炼第一句的措辞，让任何人读了它都忍不住要读接下来的那句。当读完第二句后，他们就会迫不及待想读完整个第一段，而这第一段话又让他们忍不住想要读完整篇文案。在描述语的最后，要向潜在的玩家发出号召，请他们下载此游戏。

### 内容分级

读者可能并不认为内容分级也是推广手段之一，不过它确实是。你需要确定游戏的目标玩家，而内容分级则应该反映出这部分玩家的需求。软件商店都会提供搜索筛选器，要让目标玩家通过筛选器找到你的游戏，而不是将它过滤掉。

### 玩家的评价

由于某些原因，潜在的玩家会相当重视网站上陌生人所发的评论。你需要确保游戏会受到好评，所以一定要请众多测试玩家将游戏彻底测试好，才可以正式发布。这些测试玩家应该可以在软件商店中随意发表对于游戏的评价，而且你也应该鼓励他们这么做。对于偶尔出现的差评，不要过于担心，只要大多数评价者都对游戏有正面印象就好。

## 价格

定价是一门艺术。就算读完了所有讲述定价的书，也未必能为游戏定出合适的价格来。让游戏免费通常是一个不错的选择，除了下载付费之外，还有多种可以通过游戏赚钱的方式。如果你决定要将游戏做成付费的，可以考虑提供一个“精简版”（lite version）以供免费下载，这样可以让游戏面对更多的玩家。

除了免费的游戏外，似乎手机游戏的定价都在 0.99 ~ 4.99 美元之间，低价的游戏通常比高价的游戏销量好。用户在软件商店中搜索游戏时，可以根据价格来筛选，所以，一定要定个适当的价格，以免游戏被用户筛选掉。

要注意的是，不论游戏定价多少，都不可能拿到 100% 的钱。现在的 Android Market 协议约定 Google 将每次游戏下载费的 70% 分给开发者，而 Amazon App Store 的协议则约定，开发者的收入，是游戏销售总额的 70% 与推荐列表中销售总额的 20%（因为推荐列表中的价格可能会打折）中较多的那一个。如果你一开始是将游戏免费发布的，那么稍后就不能将其改为付费的了，不过可以发布一个功能受限的免费版与一个全功能的付费版。

### 17.4.2 游戏评论网站推广

手机游戏评论网站通常是让游戏获得关注的好地方。像前文提到的那样，每个软件商店都有评论区，不过此外还有一些专注于游戏评论的网站，这种网站各自都有一套提交游戏的过程。游戏一旦被人评论了，就会有一个新的玩家群体注意到它的存在，这会促使其将该游戏下载下来玩。

这里列举一些专注于手机游戏评论的网站：

- ☐ <http://www.pocketgamer.co.uk/>
- ☐ <http://www.gamespot.com/mobile/index.html>
- ☐ <http://www.pocket-arcade.com/>

### 17.4.3 手机广告

本章前面说过，手机广告是游戏赢利的一种方式，此外它还是一种推广游戏的有效手段。不过，在游戏推广过程中，你需要花钱来发布广告。比方说，可以付给 Google 少量的广告费，如果用户真的在 Google 中同时搜索“virgins”与“vampires”这两个词，那么搜索结果中将会展示一个小幅广告，告诉用户有一款 Android 游戏同时包含这两个内容。

以下是一些专注于手机用户的广告服务提供商：

- ☐ AdMob ([www.admob.com](http://www.admob.com))：最大的手机广告提供商，为 Google 公司所有
- ☐ inMobi ([www.inmobi.com](http://www.inmobi.com))：Android 手机平台中 AdMob 所面对的强劲竞争对手



- JumpTap ([www.jumptap.com](http://www.jumptap.com)): 宣称通过精心选择目标手机用户而使广告投放者的回报最大化
  - Millennial Media ([www.millennialmedia.com](http://www.millennialmedia.com)): AdMob 面对的另一个强劲竞争对手
- 所有这些广告服务都是跨平台的 (支持 Android、iOS、Windows Phone 7 等平台), 不过你可以在发布广告时选择自己的目标玩家。

#### 17.4.4 □ 口碑营销

每一个游戏开发者都梦想其游戏能够像病毒那样迅速传播: 人们会在饮水机旁接水时或者吃午餐时谈起你的游戏, 吸引他人的关注, 并告诉他们这款游戏有多么好玩。没有什么方法能确保一定可以达到这种效果, 不过关键是需要让那些乐于向他人介绍新鲜事物的玩家接触到你的游戏, 并保证他们在玩的时候能够享受到游戏的乐趣。

将新做好的游戏告诉身边的每个人, 并且要反复地说给他们听, 以免你的新闻淹没在繁杂的日常生活琐事当中。

#### 17.4.5 社交网络推广

如果游戏还没有 Facebook 页面的话, 就应该为其建立一个。为什么不呢? Facebook 也是一个人们能够找到你游戏的地方, 他们会在上面分享玩游戏的体会。Twitter 是另一个让游戏引人注意的方式, 尤其是发布更新版本时, 可以通过 Twitter 消息促使大家下载并尝鲜。

### 17.5 总结

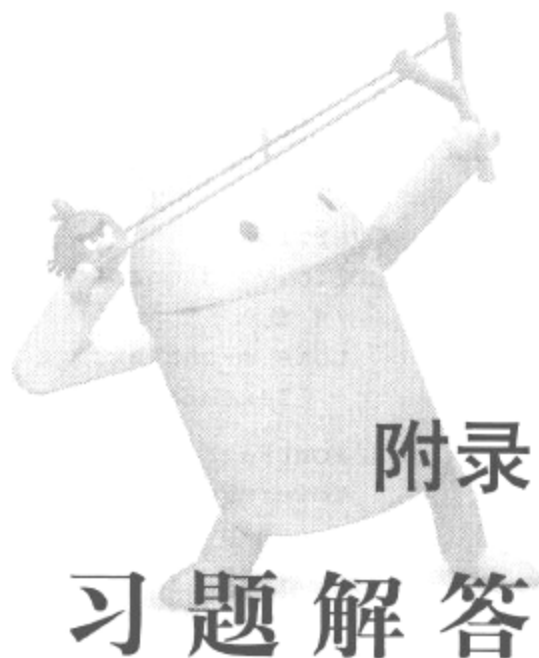
如果游戏制作团队中有一位 MBA (Master of Business Administration, 工商管理硕士) 的话 (也许读者本身就拥有 MBA 学位), 那么这个人一定会告诉你营销学中的“4P”概念:

- 产品 (Product): 必须要有满足顾客需要的产品。在理想情况下, 你所制作的游戏可以吸引玩家, 并让其享受到游戏的快乐, 帮他们打发时间, 甚至具有一点教育意义。
- 价格 (Price): 产品必须有一个使顾客愿意购买它的定价。如前所述, 定价是个很棘手的问题, 而且不单单局限于首次购买游戏时的定价。在这个问题上能够产生很多有创造性的想法。
- 地点 (Place): 营销学意义上的地点指的是顾客购买产品的市场环境。就手机游戏而言, 很显然, 已经有许多由 Google 或 Amazon 这样的大公司所运营的软件商店了。



□ 推广 (Promotion): 顾客必须注意到你的游戏, 而且要他们要意识到这款游戏很有趣, 值得下载下来玩。推广游戏的方式很多, 可以通过软件商店本身, 也可以通过游戏评论站点, 还可以使用手机广告、口碑营销或社交网络等手段, 不管使用哪种方式进行推广, 都要确保游戏能够吸引尽可能多的玩家注意。

祝贺你! 如果读者将本书从头到尾完整地读了一遍, 并详细地研究了书中的范例代码与习题, 那么你就掌握了使用 AndEngine 制作并发行 Android 手机游戏所需的全部知识。现在就来制作你梦想之中的那个游戏吧, 将其发布给大家, 不过别忘了, 最重要的是, 要享受整个游戏制作过程喔!



# 附录 习题解答

## 第 1 章

本章的习题没有标准答案，这些习题就是请读者试用一下本章所讲到的图形制作工具与音乐制作工具，并策划一下你想要做的游戏。如果在完成这些习题的过程中遇到了困难，请参阅本章正文，这样应该就能顺利地找到答案了。

## 第 2 章

本章的习题没有标准答案，它们要求读者使用本章所讲的工具来制作或者查找一些图片与音乐资源，并继续规划一下要做的游戏。如果在完成这些习题的过程中遇到了困难，请参阅本章正文，这样应该就能顺利地找到答案了。

## 第 3 章

1. 在 `onLoadScene()` 方法的代码中，静态菜单是在显示其他物体之前加入场景的，所以此时并不会触发任何动画效果。而弹出式的菜单场景则是在主场景对象已经显示出来之后才加入的，所以它的动画效果能够正常执行，使得菜单能够从左方滑入屏幕。

2. 菜单颜色的设定可以有两种方式：一种方式是在 `onLoadResources()` 方法载入字型所用的 `TextureRegions` 对象时设定，另一种是通过 `ColorMenuItemDecorator` 对象来设定。因为 `MainMenuActivity.java` 的代码就是使用 `ColorMenuItemDecorator` 来设定菜单项颜色的，所以这里就采用第二种方式，将各个菜单项以红、白、蓝等颜色显示出来。

## 修改后的 MenuActivity.java 代码, 它将以多种颜色显示各个菜单项

```

@Override
public void onLoadResources() {
    /* 载入字型与贴图对象。 */
    this.mFontTexture = new Texture(256, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    FontFactory.setAssetBasePath("font/");
    this.mFont = FontFactory.createFromAsset(
        this.mFontTexture, this, "Flubber.ttf", 32,
        true, Color.RED);
    this.mEngine.getTextureManager().loadTexture(
        this.mFontTexture);
    this.mEngine.getFontManager().loadFont(this.mFont);
}

protected void createStaticMenuScene() {
    this.mStaticMenuScene = new MenuScene(this.mCamera);

    final IMenuItem playMenuItem =
        new ColorMenuItemDecorator(
            new TextMenuItem(MENU_PLAY, mFont, "Play Game"),
            0.5f, 0.5f, 0.5f, 1.0f, 1.0f, 1.0f);
    playMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mStaticMenuScene.addMenuItem(playMenuItem);

    final IMenuItem scoresMenuItem =
        new ColorMenuItemDecorator(
            new TextMenuItem(MENU_SCORES, mFont, "Scores"),
            0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f);
    scoresMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mStaticMenuScene.addMenuItem(scoresMenuItem);

    final IMenuItem optionsMenuItem =
        new ColorMenuItemDecorator(
            new TextMenuItem(MENU_OPTIONS, mFont, "Options"),
            0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f);
    optionsMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mStaticMenuScene.addMenuItem(optionsMenuItem);

    final IMenuItem helpMenuItem =
        new ColorMenuItemDecorator(
            new TextMenuItem(MENU_HELP, mFont, "Help"), 0.5f,
            0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    helpMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mStaticMenuScene.addMenuItem(helpMenuItem);
}

```

载入字型对象的代码不变，而创建 ColorMenuItemDecorator 所用的构造器参数则进行了微调，调整之后，“Play Game”菜单项将以白色显示，“Scores”变成蓝色，“Options”变为绿色，而“Help”则是红色的。

3. ScaleMenuItemDecorator 的用法与 ColorMenuDecorator 很相似。以下代码片段展示了如何在主菜单中的某个菜单项上使用此修饰器。

修改后的 MenuActivity.java 代码，用户选中某个菜单项时它会突然变大

```

. . .
protected void createStaticMenuScene() {
    this.mStaticMenuScene = new MenuScene(this.mCamera);

    final IMenuItem playMenuItem = new ScaleMenuItemDecorator(
        new TextMenuItem(MENU_PLAY, mFont, "Play Game"),
        1.2f, 1.0f);
    playMenuItem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE_MINUS_SRC_ALPHA);
    this.mStaticMenuScene.addMenuItem(playMenuItem);
. . .

```

ScaleMenuItemDecorator 构造器的最后两个参数分别指定了菜单项在选中状态与未选中状态下的缩放比例。在上述代码中，playMenuItem 这个菜单项在选中状态会被放大 20% 显示，而在未选中状态则保持原有大小。

## 第 4 章

1. 下列程序清单所列的这个简短的 Activity 演示了如何使用不同的修改器。完整的 Android 项目可以在下载代码中名叫 Modifier 的项目文件夹下找到。

用于演示各种 Modifier 用法的 StartActivity.java 代码

```

package com.pearson.lagp.modex;

import org.anddev.andengine.engine.Engine;
import org.anddev.andengine.engine.camera.Camera;
import org.anddev.andengine.engine.options.EngineOptions;
import org.anddev.andengine.engine.options.EngineOptions
    .ScreenOrientation;
import org.anddev.andengine.engine.options.resolutionpolicy
    .RatioResolutionPolicy;
import org.anddev.andengine.entity.modifier.ScaleModifier;
import org.anddev.andengine.entity.scene.Scene;
import org.anddev.andengine.entity.sprite.Sprite;
import org.anddev.andengine.entity.util.FPSLogger;
import org.anddev.andengine.opengl.texture.Texture;
import org.anddev.andengine.opengl.texture.TextureOptions;
import org.anddev.andengine.opengl.texture.region.TextureRegion;

```

```

import org.anddev.andengine.opengl.texture.region.TextureRegionFactory;
import org.anddev.andengine.ui.activity.BaseGameActivity;

import android.os.Handler;

public class StartActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    // =====
    // 字段
    // =====

    private Camera mCamera;
    private Texture mTexture;
    private TextureRegion mFaceTextureRegion;
    private Handler mHandler;

    // =====
    // 构造器
    // =====

    // =====
    // Getter 与 Setter 方法
    // =====

    // =====
    // 覆写超类与接口中的方法
    // =====

    @Override
    public Engine onLoadEngine() {
        mHandler = new Handler();
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH, CAMERA_HEIGHT);
        return new Engine(new EngineOptions(true, ScreenOrientation
        .LANDSCAPE, new RatioResolutionPolicy(CAMERA_WIDTH, CAMERA_HEIGHT), this
        .mCamera).setNeedsMusic(true));
    }

    @Override
    public void onLoadResources() {
        TextureRegionFactory.setAssetBasePath("gfx/");
        this.mTexture = new Texture(512, 1024,
            TextureOptions.BILINEAR_PREMULTIPLYALPHA);
        this.mFaceTextureRegion = TextureRegionFactory
        .createFromAsset( this.mTexture, this,
            "mathead.png", 0, 0);
    }
}

```

```

        this.mEngine.getTextureManager().loadTexture(this.mTexture);
    }

    @Override
    public Scene onLoadScene() {
        this.mEngine.registerUpdateHandler(new FPSLogger());

        final Scene scene = new Scene(1);

        /* 将摄像头对准头像中央。 */
        final int centerX = (CAMERA_WIDTH - this.mFaceTextureRegion
            .getWidth()) / 2;
        final int centerY = (CAMERA_HEIGHT - this.mFaceTextureRegion
            .getHeight()) / 2;

        /* 创建用于显示头像的精灵对象并将其加入到场景中。 */
        final Sprite face = new Sprite(centerX, centerY, this
            .mFaceTextureRegion);

        face.registerEntityModifier(new ScaleModifier(10.0f, 0.0f, 1.0f));

        scene.getLastChild().attachChild(face);
        return scene;
    }

    @Override
    public void onLoadComplete() {
    }

    // =====
    // 方法
    // =====
    // =====
    // 匿名类及内部类
    // =====
}

```

2. 本题中所要求的效果需要结合 `SequenceEntityModifiers` 与 `ParallelEntityModifiers` 来做。下面列出了其中一种组合方式，这段代码可以使用习题 1 解答中所附的程序框架来测试。这段代码也可以在下载的 `Modifier` 项目中找到。

#### 将各种 Modifier 组合起来

```

        face.registerEntityModifier(new ParallelEntityModifier(
            new MoveModifier(3.0f, 0.0f, CAMERA_WIDTH/2,
                0.0f, CAMERA_HEIGHT/2),
            new SequenceEntityModifier(
                new ColorModifier(2.0f, 1.0f,

```



```

        0.0f, 1.0f, 0.0f,
        1.0f, 1.0f),
        new ColorModifier(2.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        1.0f, 0.0f),
        new ColorModifier(2.0f, 1.0f,
        0.0f, 0.0f, 1.0f,
        0.0f, 0.0f)
    ),
    new SequenceEntityModifier(
        new ScaleModifier(1.0f, 1.0f,
        1.0f),
        new ScaleModifier(2.0f, 1.0f,
        0.5f),
        new ScaleModifier(2.0f, 0.5f,
        1.0f)
    ),
    new SequenceEntityModifier(
        new DelayModifier(2.0f),
        new RotationModifier(2.0f,
        0.0f, 720.0f)
    )
);
...

```

3. 只需要在 onLoadScene() 方法中加入一行代码即可。当创建了 mMainScene 对象之后, 调用如下方法:

```
mMainScene.setScaleCenter(centerX, centerY);
```

4. 有多种方式可以实现这个缓动函数, EaseWiggle.java 中的代码是其中一种实现方式。

#### EaseWiggle.java

```

public class EaseWiggle implements IEaseFunction {
    private static EaseWiggle INSTANCE;

    // =====
    // 构造器
    // =====

    private EaseWiggle() {
    }

    public static EaseWiggle getInstance() {
        if(INSTANCE == null) {
            INSTANCE = new EaseLinear();
        }
    }
}

```

```

        return INSTANCE;
    }

    // =====
    // 覆写超类与接口中的方法
    // =====

    @Override
    public static float getPercentageDone(final float pSecondsElapsed,
        final float pDuration, final float pMinValue,
        final float pMaxValue) {
        return (float) (pMaxValue * pSecondsElapsed / pDuration +
            pMinValue + 4.0f * Math.sin(Math.PI *
            pSecondsElapsed * 10.0f/pDuration));
    }
}

```

---

## 第 5 章

### 1. 旋转红星所用的一种办法如下所示:

#### StartActivity.java

---

```

package com.pearson.lagp.v3;

+imports

public class StarActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;
    private String tag = "SpriteTestActivity";

    // =====
    // 字段
    // =====

    protected Camera mCamera;

    protected Scene mMainScene;

    // =====
    // 构造器
    // =====

    // =====

```

```

// Getter 与 Setter 方法
// =====

// =====
// 覆写超类与接口中的方法
// =====

@Override
public Engine onLoadEngine() {
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
}

@Override
public void onLoadResources() {
}

@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    final Scene scene = new Scene(1);
    scene.setBackground(new ColorBackground(0.0f, 0.0f, 0.0f));

    /* 将摄像头置于屏幕中央。 */
    final int centerX = CAMERA_WIDTH / 2;
    final int centerY = CAMERA_HEIGHT / 2;

    /* 绘制星状图案。 */
    Line star1 = new Line(centerX-100, centerY-40, centerX+100,
        centerY-40, 3.0f);
    star1.setColor(1.0f, 0.0f, 0.0f);
    Line star2 = new Line(200, 0, 40, 125, 3.0f);
    star2.setColor(1.0f, 0.0f, 0.0f);
    Line star3 = new Line(-160, 125, -100, -75, 3.0f);
    star3.setColor(1.0f, 0.0f, 0.0f);
    Line star4 = new Line(-100, -75, -30, 125, 3.0f);
    star4.setColor(1.0f, 0.0f, 0.0f);
    Line star5 = new Line(-30, 125, -200, 0, 3.0f);
    star5.setColor(1.0f, 0.0f, 0.0f);
    star1.attachChild(star2);
    star1.getLastChild().attachChild(star3);
    star1.getLastChild().attachChild(star4);
    star1.getLastChild().attachChild(star5);
    star1.setRotationCenter(100, 40);
    star1.registerEntityModifier(new RotationModifier(5.0f,
        0.0f, 360.0f));
}

```

```

        scene.getLastChild().attachChild(star1);

        return scene;
    }

    @Override
    public void onLoadComplete() {
    }
}

```

请注意，Line 对象与 setRotationCenter 方法所使用的坐标都是相对于 star1 对象来说的。

2. 唯一需要修改代码的地方就是 SpriteTestActivity.java 中的 onLoadResources() 方法（当然还需要将 SVG 扩展包的 .jar 文件与 SVG 图形文件引入到项目中）。下面列出了修改后的代码，也可以在本章所附代码的 V305SVG 文件夹中找到它。

#### StarActivity.java

```

...
@Override
public void onLoadResources() {
    /* 载入 Textures 对象。 */
    TextureRegionFactory.setAssetBasePath("gfx/SpriteTest/");
    mTestTexture = new Texture(512, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mHatchetTextureSource= new SVGAssetTextureSource(
        this, "svg/hatchet40.svg", 1.0f);
    this.mMadMatTextureSource= new SVGAssetTextureSource(this,
        "svg/Mat.svg", 1.0f);

    mHatchetTextureRegion =
        TextureRegionFactory.createFromSource(mTestTexture,
            mHatchetTextureSource, 0,0);
    mMadMatTextureRegion =
        TextureRegionFactory.createFromSource(mTestTexture,
            mMadMatTextureSource, 50,0);
    this.mEngine.getTextureManager().loadTexture(
        this.mTestTexture);
}
...

```

3. 这道习题每位读者都会有自己的解决办法，并没有统一的答案。

## 第 6 章

1. 要让蝙蝠飞来飞去，需要修改 StartActivity.java 中的 onLoadScene() 方法，以下

这段代码列出了修改之后的代码。

为了改变蝙蝠的移动方式，而对 StartActivity.java 中的 onLoadScene() 方法进行修改

```
@Override
public Scene onLoadScene() {
    this.mEngine.registerUpdateHandler(new FPSLogger());

    final Scene scene = new Scene(1);

    /* 将摄像头置于启动画面中央。 */
    final int centerX = (CAMERA_WIDTH -
        this.mSplashTextureRegion.getWidth()) / 2;
    final int centerY = (CAMERA_HEIGHT -
        this.mSplashTextureRegion.getHeight()) / 2;

    /* 创建表示背景精灵对象并将其加入场景中。 */
    final Sprite splash = new Sprite(centerX, centerY,
        this.mSplashTextureRegion);
    scene.getLastChild().attachChild(splash);

    /* 创建代表蝙蝠动画的精灵对象并将其加入场景中。 */
    final AnimatedSprite bat = new AnimatedSprite(350, 100,
        this.mBatTextureRegion);
    bat.animate(100, true);
    bat.registerEntityModifier(new LoopEntityModifier(
        new SequenceEntityModifier(
            new MoveXModifier(2.0f, bat.getX(),
                bat.getX() - 60),
            new MoveXModifier(2.0f, bat.getX() - 60,
                bat.getX()))));
    scene.getLastChild().attachChild(bat);
    return scene;
}
```

以上代码使用 LoopEntityModifier 来重复蝙蝠的整套动作，并使用了两个前后相连的 MoveXModifiers 来执行每次循环中蝙蝠的移动。如果想让蝙蝠在墓碑后面飞行，则需要将墓碑的图样从背景图中抠出来，作为一个独立的 Layer 对象，放置在代表蝙蝠动画的 AnimatedSprite 对象之上。

2. 这道习题的解法视每位读者的具体情况而定，没有统一答案。
3. 这道习题的解法视每位读者的具体情况而定，没有统一答案。
4. 这道题的解法如下。AndEngine 允许开发者向引擎注册一个 UpdateHandler 对象，每次屏幕刷新的时候，这个对象之中的逻辑代码都会执行一次。onLoadScene() 方法在将 Scene 对象返回给引擎之前，注册了一个 UpdateHandler 对象。

Level1Activity.java 代码中的 onLoadScene() 方法, 修改后的程序  
可以让吸血鬼在到达 Miss B 家门口时自动消失

```

. . .
@Override
public Scene onLoadScene() {
. . .
    scene.registerUpdateHandler(new IUpdateHandler() {
        @Override
        public void reset() { }

        @Override
        public void onUpdate(final float pSecondsElapsed) {
            for (int i=0; i<nVamp; i++){
                if (asprVamp[i].getX() < 35.0f){
                    asprVamp[i].setVisible(false);
                }
            }
        }
    });
. . .

```

代码检查每个吸血鬼的坐标, 如果发现它已经到达 Miss B 的家门口了, 就让它消失。

## 第 7 章

1. 只需要改动一行代码即可。在 onLoadResources() 方法中找到 StrokeFont 的构造器, 只需要将其最后一个参数 (也就是 boolean pStrokeOnly) 改为 false 即可:

```

this.mStrokeFont = new StrokeFont(this.mStrokeFontTexture,
    Typeface.create(Typeface.DEFAULT, Typeface.BOLD), 32, true,
    Color.RED, 2.0f, Color.WHITE, false);

```

2. 因为 Texture 对象无法容纳整个字型, 所以导致 “i” 这个字符消失了。因为字型文件中的图像是以矩阵形式保存的, 所以很难预测哪个字符会被切掉。如果游戏中发现有字符不见了, 首先要检查的就是 Texture 对象的大小。针对本题中出现的这种情况, 将其大小定为 512×512 就可以解决了。

3. 要增加 Help 选项, 需要添加一个 MENU\_OPTION 常量, 同时需要修改 createOptionsMenuScene() 与 onClickMenuItem() 方法的代码。修改后的代码如下所示:

OptionsActivity.java 代码, 修改后的程序多支持了一个名为 Help 的选项

```

package com.pearson.lagp.v3;

+imports

public class OptionsActivity extends BaseGameActivity implements

```



```

IOnMenuItemClickListener {
    // =====
    // 常量
    // =====
    . . .
    protected static final int MENU_MUSIC = 0;
    protected static final int MENU_EFFECTS = MENU_MUSIC + 1;
    protected static final int MENU_HELP = MENU_EFFECTS + 1;

    // =====
    // 字段
    // =====
    . . .
    private TextMenuItem mHelp;
    private IMenuItem helpMenuItem;

    . . .

    // =====
    // 构造器
    // =====

    // =====
    // Getter 与 Setter 方法
    // =====

    // =====
    // 覆写超类与接口中的方法
    // =====

    . . .
    @Override
    public void onLoadResources() {
        mHelp = new TextMenuItem(MENU_EFFECTS, mFont, "Help");
    }

    . . .
    @Override
    public boolean onMenuItemClicked(final MenuScene pMenuScene, final
    IMenuItem pMenuItem, final float pMenuItemLocalX, final float
    pMenuItemLocalY) {
        switch(pMenuItem.getID()) {
            . . .
            case MENU_HELP:
                Toast.makeText(MainMenuActivity.this,
                    "Help selected", Toast.LENGTH_SHORT).show();
                return true;
            default:
                return false;
        }
    }
}

```

```
// =====
// 方法
// =====

protected void createOptionsMenuScene() {
    . . .
    helpMenuItem = new ColorMenuItemDecorator( mHelp, 0.5f,
        0.5f, 0.5f, 1.0f, 0.0f, 0.0f);
    this.mOptionsMenuScene.addMenuItem(helpMenuItem);

    this.mOptionsMenuScene.buildAnimations();

    this.mOptionsMenuScene.setBackgroundEnabled(false);
    this.mOptionsMenuScene.setOnMenuItemClickListener(this);
}
. . .
```

4. 下面列出了修改后的 OptionsActivity.java 代码, 实际上笔者在调试代码中就是这么做的, 这么做可以记录下诸如翻转 Boolean 标志这样非常容易出错的操作。

修改后的 OptionsActivity.java 代码, 它将会在 Boolean 变量的值发生改变时弹出桌面通知

```
Toast.makeText(context, text, duration).show();

. . .
@Override
public boolean onOptionsItemSelected(final MenuScene pMenuScene, final
IMenuItem pItem, final float pItemLocalX, final float
pMenuItemLocalY) {
    switch(pMenuItem.getID()) {
        case MENU_MUSIC:
            if (isMusicOn) {
                isMusicOn = false;
                Toast.makeText(this,
                    "Music turned off",
                    LENGTH_SHORT).show();
            } else {
                isMusicOn = true;
                Toast.makeText(this,
                    "Music turned on",
                    LENGTH_SHORT).show();
            }
            createOptionsMenuScene();
            mMainScene.clearChildScene();
            mMainScene.setChildScene(mOptionsMenuScene);
            return true;
        case MENU_EFFECTS:
            if (isEffectsOn) {
                isEffectsOn = false;
```

```

        Toast.makeText(this,
            "Effects turned off",
            LENGTH_SHORT).show();
    } else {
        isEffectsOn = true;
        Toast.makeText(this,
            "Effects turned on",
            LENGTH_SHORT).show();
    }
    createOptionsMenuScene();
    mMainScene.clearChildScene();
    mMainScene.setChildScene(mOptionsMenuScene);
    return true;
default:
    return false;
}
}

```

## 第 8 章

### 1. 下面列出了一种解决办法:

#### DctrlMove.java

```

package com.pearson.lagp.example;

import android.app.Activity;

public class DCtrlMove extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 720;
    private static final int CAMERA_HEIGHT = 480;

    // =====
    // 字段
    // =====

    private Camera mCamera;
    private Texture mTexture;
    private TextureRegion mTextureRegion;
    private float centerX, centerY;
    private Sprite face;
    private Texture mOnScreenControlTexture;
    private TextureRegion mOnScreenControlBaseTextureRegion;
    private TextureRegion mOnScreenControlKnobTextureRegion;
}

```

```

// =====
// 构造器
// =====

// =====
// Getter 与 Setter 方法
// =====

// =====
// 覆写超类与接口中的方法
// =====

@Override
public Engine onLoadEngine() {
    this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
        CAMERA_HEIGHT);
    centerX = CAMERA_WIDTH/2;
    centerY = CAMERA_HEIGHT/2;
    return new Engine(new EngineOptions(true,
        ScreenOrientation.LANDSCAPE,
        new RatioResolutionPolicy(CAMERA_WIDTH,
        CAMERA_HEIGHT), this.mCamera));
}

@Override
public void onLoadResources() {
    this.mTexture = new Texture(256, 256,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    mTextureRegion = TextureRegionFactory.createFromAsset(
        mTexture, getApplicationContext(),
        "gfx/mathead.png", 0, 50); // 32x32

    this.mEngine.getTextureManager().loadTexture(
        this.mTexture);
    this.mOnScreenControlTexture = new Texture(256, 128,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    this.mOnScreenControlBaseTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mOnScreenControlTexture, this,
            "gfx/onscreen_control_base.png", 0, 0);
    this.mOnScreenControlKnobTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mOnScreenControlTexture, this,
            "gfx/onscreen_control_knob.png", 128, 0);
    this.mEngine.getTextureManager().loadTextures(
        this.mTexture, this.mOnScreenControlTexture);
}

@Override
public Scene onLoadScene() {

```

```

final Scene scene = new Scene(1);
scene.setBackground(new ColorBackground(0.1f, 0.6f, 0.9f));

face = new Sprite(centerX, centerY, mTextureRegion);
scene.getLastChild().attachChild(face);
final DigitalOnScreenControl digitalOnScreenControl =
    new DigitalOnScreenControl(0,
        CAMERA_HEIGHT -
        this.mOnScreenControlBaseTextureRegion.getHeight(),
        this.mCamera,
        this.mOnScreenControlBaseTextureRegion,
        this.mOnScreenControlKnobTextureRegion, 0.1f,
        new IOnScreenControlListener() {
            @Override
            public void onControlChange(
                final BaseOnScreenControl pBaseOnScreenControl,
                final float pValueX, final float pValueY) {
                face.setPosition(face.getX() + pValueX *
                    10, face.getY() + pValueY * 10);
            }
        });
digitalOnScreenControl.getControlBase().setBlendFunction(
    GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
digitalOnScreenControl.getControlBase().setAlpha(0.5f);
digitalOnScreenControl.getControlBase().setScaleCenter(0,
    128);
digitalOnScreenControl.getControlBase().setScale(1.25f);
digitalOnScreenControl.getControlKnob().setScale(1.25f);
digitalOnScreenControl.refreshControlKnobPosition();

scene.setChildScene(digitalOnScreenControl);

return scene;
}

@Override
public void onLoadComplete() {
}
}

```

2. 本章所附代码中的 ACtrlMove.java 与上题中的 DCtrlMove.java 非常相似，以下只列出 ACtrlMove.java 代码与 DCtrlMove.java 有区别的地方：

#### ACtrlMove.java

```

...
public class ACtrlMove extends BaseGameActivity {
    ...
    @Override
    public Scene onLoadScene() {

```

```

    . . .
    final AnalogOnScreenControl analogOnScreenControl =
        new AnalogOnScreenControl(0, CAMERA_HEIGHT -
            this.mOnScreenControlBaseTextureRegion.getHeight(),
            this.mCamera,
            this.mOnScreenControlBaseTextureRegion,
            this.mOnScreenControlKnobTextureRegion, 0.1f, 200,
            new IAnalogOnScreenControlListener() {
                @Override
                public void onControlChange(
                    final BaseOnScreenControl pBaseOnScreenControl,
                    final float pValueX, final float pValueY) {
                    face.setPosition(face.getX()+pValueX * 20,
                        face.getY()+pValueY * 20);
                }

                @Override
                public void onControlClick(
                    final AnalogOnScreenControl
                        pAnalogOnScreenControl) {
                    face.registerEntityModifier(
                        new SequenceEntityModifier(
                            new ScaleModifier(0.25f, 1, 1.5f),
                            new ScaleModifier(0.25f, 1.5f, 1)));
                }
            });
    analogOnScreenControl.getControlBase().setBlendFunction(
        GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
    analogOnScreenControl.getControlBase().setAlpha(0.5f);
    analogOnScreenControl.getControlBase().setScaleCenter(0,
        128);
    analogOnScreenControl.getControlBase().setScale(1.25f);
    analogOnScreenControl.getControlKnob().setScale(1.25f);
    analogOnScreenControl.refreshControlKnobPosition();

    scene.setChildScene(analogOnScreenControl);

    return scene;
}
    . . .

```

3. 下面所列的 AccelColor.java 代码通过 AndEngine 扩展包来调用 Android 系统中与加速计相关的 API, 以便检测手机方向的改变。如果读者对这段代码的意思不是很清楚的话, 笔者就来简单讲解一下: 当手机平着放的时候, Y 轴 (通常指的是智能手机屏幕中较长的那个边) 的加速度值接近于 0, 当手机立起来的时候, 则约等于重力加速度的值, 这个值大约是 9.8 米/秒<sup>2</sup>。



## AccelColor.java

```

package com.pearson.lagp.example;

+imports

public class AccelColor extends BaseGameActivity implements
    IAccelerometerListener {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 720;
    private static final int CAMERA_HEIGHT = 480;

    // =====
    // 字段
    // =====

    private Camera mCamera;
    private Scene scene;

    // =====
    // 覆写超类与接口中的方法
    // =====

    @Override
    public Engine onLoadEngine() {
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        return new Engine(new EngineOptions(true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
    }

    @Override
    public void onLoadResources() {
        this.enableAccelerometerSensor(this);
    }

    @Override
    public Scene onLoadScene() {
        scene = new Scene(1);
        scene.setBackground(new ColorBackground(0.0f, 1.0f, 0.0f));
        return scene;
    }

    @Override
    public void onLoadComplete() {

```

```

    }

    @Override
    public void onAccelerometerChanged(
        final AccelerometerData pAccelerometerData) {
        String message = "X= "+pAccelerometerData.getX()+
            "; Y="+pAccelerometerData.getY()+
            "; Z="+pAccelerometerData.getZ()+";";
        //Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
        if (pAccelerometerData.getY() > 5.0f){
            scene.setBackground(new ColorBackground(1.0f, 0.0f,
                0.0f));
        } else {
            scene.setBackground(new ColorBackground(0.0f, 1.0f,
                0.0f));
        }
    }
}

```

## 第9章

1. 修改之后的 .tmx 文件叫做 WAVTilesetEx.tmx, 它可以在本章所附代码的 V309Ex 项目中找到。该文件包含了习题中所要求添加的属性, 同时也包含了习题 2 中所用到的那个新的瓦片集。

2. 含有 “Boo!” 字样的瓦片图像也可以在 V309Ex 项目中找到, 它也是新建的瓦片集图像 WAVTileSetEx.png 的一部分。为了实现题中所要求的功能, 需要对 WAVActivity 进行如下修改:

### 修改之后的 WAVActivity.java 文件

```

package com.pearson.lagp.v3;

+imports

public class WAVActivity extends BaseGameActivity {
    . . .
    // =====
    // 字段
    // =====
    . . .
    private int mTombstoneGID = -1;
    private int mBooGID = -1;
    private int mOpenCoffinGID = 1;
    . . .
    @Override
    public Scene onLoadScene() {
        this.mEngine.registerUpdateHandler(new FPSLogger());
    }
}

```

```

final Scene scene = new Scene(1);
try {
    final TMXLoader tmxLoader = new TMXLoader(this,
        this.mEngine.getTextureManager(),
        TextureOptions.BILINEAR_PREMULTIPLYALPHA,
        new ITMXTilePropertiesListener() {
            @Override
            public void onTMXTileWithPropertiesCreated(
                final TMXTiledMap pTMXTiledMap,
                final TMXLayer pTMXLayer,
                final TMXTile pTMXTile,
                final TMXProperties<TMXTileProperty>
                    pTMXTileProperties) {
                if(pTMXTileProperties.containsTMXProperty(
                    "coffin", "true")) {
                    coffins[coffinPtr++] =
                        pTMXTile.getTileRow() * 15 +
                        pTMXTile.getTileColumn();
                    if (mCoffinGID<0){
                        mCoffinGID = pTMXTile.getGlobalTileID();
                    }
                }

                if(pTMXTileProperties.containsTMXProperty(
                    "tombstone", "true")) {
                    if (mTombstoneGID<0){
                        mTombstoneGID =
                            pTMXTile.getGlobalTileID();
                    }
                }

                if(pTMXTileProperties.containsTMXProperty(
                    "boo", "true")) {
                    if (mBooGID<0){
                        mBooGID =
                            pTMXTile.getGlobalTileID();
                    }
                }
            }
        });
    this.mWAVTMXMap = tmxLoader.loadFromAsset(this,
        "gfx/WAV/WAVmapEx.tmx");
} catch (final TMXLoadException tmxle) {
    Debug.e(tmxle);
}

tmxLayer = this.mWAVTMXMap.getTMXLayers().get(0);
scene.getFirstChild().attachChild(tmxLayer);
scene.setOnSceneTouchListener(new IOnSceneTouchListener() {
    @Override
    public boolean onSceneTouchEvent(final Scene

```

```

pScene, final TouchEvent pSceneTouchEvent) {
    switch(pSceneTouchEvent.getAction()) {
        case TouchEvent.ACTION_DOWN:
            /* 获取被触摸的瓦片对象 */
            tmxTile = tmxLayer.getTMXTileAt(
                pSceneTouchEvent.getX(),
                pSceneTouchEvent.getY());
            if((tmxTile != null) &&
                (tmxTile.getGlobalTileID() ==
                 mOpenCoffinGID)) {
                tmxTile.setGlobalTileID(
                    mWAVTMXMap, mCoffinGID);
            } else {
                if((tmxTile != null) &&
                    (tmxTile.getGlobalTileID() ==
                     mTombstoneGID)) {
                    tmxTile.setGlobalTileID(mWAVTMXMap,
                        mBooGID);
                }
                break;
            }
            case TouchEvent.ACTION_UP:
                break;
        }
        return true;
    }
}

mHandler.postDelayed(openCoffin, gen.nextInt(2000));
...
}

```

这段代码的结构与本章正文中所讲的非常相似。首先使用瓦片所附的属性来找到棺材瓦片与墓碑瓦片的全局 ID，用这些 ID 可以直接找到瓦片所对应的图像。不过不能用这种方式确定“Boo!”瓦片的 ID，因为一开始地图上并没有这种瓦片，它不会被载入，这也意味着 ITMXTilePropertiesListener 监听器中的回调方法不会被调用。不过没关系，我们可以通过 TSX 文件找到第一个瓦片的 ID 号，然后在 Tiled 软件的瓦片集视图中数一下“Boo!”这块瓦片的 ID 号。其实最好的办法还是根据某块瓦片上所附的属性来确定其身份，不过在本题所用的代码结构中，只能这么做了。

## 第 10 章

### 1. 修改之后的代码如下所示。

增加了烟雾粒子效果之后的 Level1Activity.java 文件

```

package com.pearson.lagp.v3;
...
@Override

```

```

public Scene onLoadScene() {
    . . .
    particleEmitter = new CircleParticleEmitter(
        CAMERA_WIDTH * 0.5f, CAMERA_HEIGHT * 0.5f + 20, 40);
    particleSystem = new ParticleSystem(particleEmitter,
        100, 100, 500, this.mParticleTextureRegion);

    particleSystem.addParticleInitializer(
        new ColorInitializer(1, 1, 1));
    particleSystem.addParticleInitializer(
        new AlphaInitializer(0));
    particleSystem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE);
    particleSystem.addParticleInitializer(
        new VelocityInitializer(-2, 2, -2, -2));
    particleSystem.addParticleInitializer(
        new RotationInitializer(0.0f, 360.0f));

    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-
            .ScaleModifier(1.0f, 2.0f, 0, 5));
    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-
            .ColorModifier(0, 0.5f, 0, 0.5f, 0, 0.5f, 0, 2));
    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-
            .ColorModifier(0.5f, 0, 0.5f, 0, 0.5f, 1, 2, 4));
    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-
            .AlphaModifier(0, 1, 0, 1));
    particleSystem.addParticleModifier(
        new org.anddev.andengine.entity.particle.modifier-
            .AlphaModifier(1, 0, 3, 4));
    particleSystem.addParticleModifier(
        new ExpireModifier(2, 4));

    particleSystem.setParticlesSpawnEnabled(false);
    scene.getLastChild().attachChild(particleSystem);

    return scene;
}
. . .
}

```

## 2. 烟雾粒子效果所用的 PX 文件如下所示:

smoke.px

```

<ParticleConfig>
<emitter

```

```

    shape="circle"
    center_x="0.0"
    center_y="0.0"
    radius_x="40.0"
    radius_y="40.0">
</emitter>
<system
    texture="particle_fire.png"
    min_rate="100"
    max_rate="100"
    max_particles="500">
    <init_color
        min_red="1"
        max_red="1"
        min_green="1"
        max_green="1"
        min_blue="1"
        max_blue="1">
    </init_color>
    <init_alpha
        min_alpha="0"
        max_alpha="0">
    </init_alpha>
    <init_velocity
        min_velocity_x="-2"
        max_velocity_x="2"
        min_velocity_y="-2"
        max_velocity_y="2">
    </init_velocity>
    <init_rotation
        min_rotation="0.0"
        max_rotation="360.0">
    </init_rotation>
    <mod_scale
        from_scale_x="1.0"
        to_scale_x="1.0"
        from_scale_y="1.0"
        to_scale_y="1.0"
        to_scale="2.0"
        from_time="0"
        to_time="5">
    </mod_scale>
    <mod_color
        from_red="0"
        to_red="0.5"
        from_green="0"
        to_green="0.5"
        from_blue="0"
        to_blue="0.5"
        from_time="0"

```



```

        to_time="2">
    </mod_color>
    <mod_color
        from_red=".5"
        to_red="0"
        from_green="0.5"
        to_green="0"
        from_blue=".5"
        to_blue="1"
        from_time="2"
        to_time="4">
    </mod_color>
    <mod_alpha
        from_alpha="0"
        to_alpha="1"
        from_time="0"
        to_time="1">
    </mod_alpha>
    <mod_alpha
        from_alpha="1.0"
        to_alpha="0"
        from_time="3"
        to_time="4">
    </mod_alpha>
    <mod_expire
        min_lifetime="2"
        max_lifetime="4">
    </mod_expire>
</system>
</ParticleConfig>

```

---

3. 下雨粒子效果所用的 PX 文件如下所示:

rain.px

---

```

<ParticleConfig>
    <emitter
        shape="rectangle"
        center_x="0.0"
        center_y="0.0"
        width="480"
        height="2.0">
    </emitter>
    <system
        texture="particle_point.png"
        min_rate="100"
        max_rate="100"
        max_particles="500">
        <init_color
            min_red="1"

```

```

        max_red="1"
        min_green="1"
        max_green="1"
        min_blue="1"
        max_blue="1">
    </init_color>
    <init_alpha
        min_alpha="0"
        max_alpha="0">
    </init_alpha>
    <init_velocity
        min_velocity_x="-2"
        max_velocity_x="2"
        min_velocity_y="50"
        max_velocity_y="100">
    </init_velocity>
    <init_rotation
        min_rotation="0.0"
        max_rotation="0.0">
    </init_rotation>
    <init_gravity
        gravity_on="true">
    </init_gravity>
    <mod_scale
        from_scale_x="0.2"
        to_scale_x="0.2"
        from_scale_y="0.3"
        to_scale_y="0.3"
        from_time="0"
        to_time="5">
    </mod_scale>
    <mod_alpha
        from_alpha="0"
        to_alpha="1"
        from_time="0"
        to_time="1">
    </mod_alpha>
    <mod_alpha
        from_alpha="1.0"
        to_alpha="0"
        from_time="3"
        to_time="4">
    </mod_alpha>
    <mod_expire
        min_lifetime="2"
        max_lifetime="4">
    </mod_expire>
</system>
</ParticleConfig>

```

---

## 第 11 章

1. 为了给十字架武器增加音效，需要对 `Level1Activity.java` 进行如下修改：

- ❑ 录制音乐文件。笔者将其以 Ogg/Vorbis 格式存储，命名为 `OCS.ogg`，并导入到游戏项目的 `assets/mfx` 文件夹中。
- ❑ 在 `Level1Activity.java` 中增加一个 `Music` 对象：`Music mOCSMusic`。
- ❑ 在 `Level1Activity.java` 中的 `EngineOption` 对象上增加 `setNeedsMusic()` 方法。
- ❑ 在 `onLoadResources()` 方法中，增加以下几行用于载入音乐的代码：

```
MusicFactory.setAssetBasePath("mfx/");
try {
    this.mOCSMusic =
        MusicFactory.createMusicFromAsset(
            this.mEngine.getMusicManager(),
            this, "OCS.ogg");
    this.mOCSMusic.setLooping(true);
} catch (final IOException e) {
    Debug.e(e);
}
```

- ❑ 找到与 `cross` 对象相关的 `TouchEvent` 事件处理代码，于 `ACTION_UP` 分支中增加如下代码：

```
if (audioOptions.getBoolean("musicOn", false)) {
    mOCSMusic.play();
}
```

- ❑ 在 `onGamePaused()` 方法中，增加一行代码：

```
mOCSMusic.stop();
```

2. 要想使用 MIDI 格式的音乐，需要对 `StartActivity.java` 文件中的 `onLoadResources()` 方法做出修改。在使用 `MusicFactory` 加载音乐文件的 `try/catch` 语句块中，将这行代码修改如下：

```
StartActivity.mMusic = MusicFactory.createMusicFromAsset(
    this.mEngine.getMusicManager(), getApplicationContext(),
    "bachfugue2.mid");
```

3. 要在 `SharedPreferences` 与游戏屏幕上加入分数，首先需要在 `StartActivity.java` 中增加如下几行代码：

#### 根据习题 3 的要求修改之后的 `StartActivity.java` 代码

```
private SharedPreferences scores;
private SharedPreferences.Editor scoresEditor;
...
@Override
public Engine onLoadEngine() {
```

```

    . . .
    scores = getSharedPreferences("scores", MODE_PRIVATE);
    scoresEditor = scores.edit();
    if (!scores.contains("WAV")) {
        scoresEditor.putInt("WAV", 0);
        scoresEditor.putInt("Level1", 0);
        scoresEditor.commit();
    }

```

在 MainMenuActivity.java 的 menuItemClick() 方法中, 将 MENU\_SCORES 分支的代码按照下列程序清单做出修改, 同时要将省略号后面的那个 Runnable 任务加入到系统中。

#### 根据习题 3 的要求修改之后的 MainMenuActivity.java 代码

```

    . . .
    @Override
    public boolean onOptionsItemSelected(final MenuScene pMenuScene,
        final IMenuItem pMenuItem, final float pMenuItemLocalX,
        final float pMenuItemLocalY) {
        switch(pMenuItem.getID()) {
            . . .
            case MENU_SCORES:
                mMainScene.registerEntityModifier(
                    new ScaleAtModifier(0.5f, 1.0f,
                        0.0f, CAMERA_WIDTH/2,
                        CAMERA_HEIGHT/2));
                mStaticMenuScene.registerEntityModifier(
                    new ScaleAtModifier(0.5f, 1.0f,
                        0.0f, CAMERA_WIDTH/2,
                        CAMERA_HEIGHT/2));
                mHandler.postDelayed(mLaunchScoresTask,
                    500);
                return true;
            . . .
            private Runnable mLaunchScoresTask = new Runnable() {
                public void run() {
                    Intent myIntent = new Intent(MainMenuActivity.this,
                        ScoresActivity.class);
                    MainMenuActivity.this.startActivity(myIntent);
                }
            };

```

接下来新建一个名为 ScoresActivity.java 的类, 其代码如下所示。别忘了将它加入到游戏项目的 manifest 文件中, 这样 Android 系统就会知道程序中多了一个可运行的 Activity 了。

#### 根据习题 3 的要求所增加的 ScoresActivity.java 代码

```

import org.anddev.andengine.opengl.texture.Texture;
import org.anddev.andengine.opengl.texture.TextureOptions;

```

```

import org.anddev.andengine.ui.activity.BaseGameActivity;

import android.content.SharedPreferences;
import android.graphics.Color;

public class ScoresActivity extends BaseGameActivity {
    // =====
    // 常量
    // =====

    private static final int CAMERA_WIDTH = 480;
    private static final int CAMERA_HEIGHT = 320;

    // =====
    // 字段
    // =====

    protected Camera mCamera;

    protected Scene mScoresScene;
    private Text mTitle, mWAV, mLevel1;;
    private Texture mFontTexture;
    private Font mFont;
    private SharedPreferences scores;
    private SharedPreferences.Editor scoresEditor;

    // =====
    // 构造器
    // =====

    // =====
    // Getter 与 Setter 方法
    // =====

    // =====
    // 覆写超类与接口中的方法
    // =====

    @Override
    public Engine onLoadEngine() {
        this.mCamera = new Camera(0, 0, CAMERA_WIDTH,
            CAMERA_HEIGHT);
        scores = getSharedPreferences("scores", MODE_PRIVATE);
        scoresEditor = scores.edit();
    }

```

```

        return new Engine(new EngineOptions(true,
            ScreenOrientation.LANDSCAPE,
            new RatioResolutionPolicy(CAMERA_WIDTH,
            CAMERA_HEIGHT), this.mCamera));
    }

    @Override
    public void onLoadResources() {
        /* 载入 Font 与 Textures 对象。 */
        this.mFontTexture = new Texture(256, 256,
            TextureOptions.BILINEAR_PREMULTIPLYALPHA);

        FontFactory.setAssetBasePath("font/");
        this.mFont = FontFactory.createFromAsset(this.mFontTexture,
            this, "Flubber.ttf", 32, true, Color.RED);
        this.mEngine.getTextureManager().loadTexture(
            this.mFontTexture);
        this.mEngine.getFontManager().loadFont(this.mFont);
    }

    @Override
    public Scene onLoadScene() {
        /* 将摄像头置于背景中央。 */
        final int centerX = (CAMERA_WIDTH) / 2;
        final int centerY = (CAMERA_HEIGHT) / 2;

        this.mScoresScene = new Scene(1);
        /* 将背景图像与分数加入场景。 */
        mScoresScene.setBackground(new ColorBackground(0.0f, 0.0f,
            0.0f));
        mTitle = new Text( centerX - 200, centerY - 100, mFont,
            "Scores");
        mWAV = new Text( centerX - 100, centerY - 50, mFont,
            "WAV\t\t" + scores.getInt("WAV", -1));
        mLevel1 = new Text( centerX - 100, centerY, mFont,
            "Level1\t\t" + scores.getInt("Level1", -1));
        mScoresScene.getLastChild().attachChild(mTitle);
        mScoresScene.getLastChild().attachChild(mWAV);
        mScoresScene.getLastChild().attachChild(mLevel1);
        return this.mScoresScene;
    }

    @Override
    public void onLoadComplete() {
    }

```



```

// =====
// 方法
// =====

// =====
// 匿名内部类
// =====

}

```

---

## 第 12 章

本章所有习题的解答都可以在所附的 V312 Exercises 项目代码中找到。

1. 可用于第二关的范例关卡文件如下所示：

iv2.lv

---

```

<level>
  <completeShape>
    <xprop>240.0000</xprop>
    <yprop>1.0000</yprop>
    <height>2.00</height>
    <width>480.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>false</isDynamic>
    <shape>SQUARE</shape>
    <physicsandID>0.5,0.5,0.5,'stone'</physicsandID>
    <verts>'()'</verts>
  </completeShape>
  <completeShape>
    <xprop>240.0000</xprop>
    <yprop>319.0000</yprop>
    <height>2.00</height>
    <width>480.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>false</isDynamic>
    <shape>SQUARE</shape>
    <physicsandID>0.5,0.5,0.5,'stone'</physicsandID>
    <verts>'()'</verts>
  </completeShape>
  <completeShape>
    <xprop>165.0000</xprop>
    <yprop>298.0000</yprop>
    <height>40.00</height>
    <width>40.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>true</isDynamic>

```

```

    <shape>CIRCLE</shape>
    <physicsandID>0.5,0.5,0.5,'glass'</physicsandID>
    <verts>'()'</verts>
</completeShape>
<completeShape>
    <xprop>260.0000</xprop>
    <yprop>300.0000</yprop>
    <height>40.00</height>
    <width>40.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>true</isDynamic>
    <shape>CIRCLE</shape>
    <physicsandID>0.5,0.5,0.5,'wood'</physicsandID>
    <verts>'()'</verts>
</completeShape>
<completeShape>
    <xprop>207.0000</xprop>
    <yprop>276.0000</yprop>
    <height>6.00</height>
    <width>186.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>true</isDynamic>
    <shape>SQUARE</shape>
    <physicsandID>0.5,0.5,0.5,'wood'</physicsandID>
    <verts>'()'</verts>
</completeShape>
<completeShape>
    <xprop>155.0000</xprop>
    <yprop>249.0000</yprop>
    <height>50.00</height>
    <width>50.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>true</isDynamic>
    <shape>SQUARE</shape>
    <physicsandID>0.5,0.5,0.5,'wood'</physicsandID>
    <verts>'()'</verts>
</completeShape>
<completeShape>
    <xprop>480.2494</xprop>
    <yprop>161.5000</yprop>
    <height>-4.50</height>
    <width>319.00</width>
    <rotation>1.5708</rotation>
    <isDynamic>false</isDynamic>
    <shape>SQUARE</shape>
    <physicsandID>0.5,0.5,0.5,'stone'</physicsandID>
    <verts>'()'</verts>
</completeShape>
<completeShape>
    <xprop>234.0000</xprop>
    <yprop>262.0000</yprop>

```

```

    <height>24.00</height>
    <width>50.00</width>
    <rotation>0.0000</rotation>
    <isDynamic>true</isDynamic>
    <shape>CIRCLE</shape>
    <physicsandID>0.5,0.5,0.5,'vamp'</physicsandID>
    <verts>'()'</verts>
</completeShape>
<completeShape>
    <xprop>277.6250</xprop>
    <yprop>268.5000</yprop>
    <height>9.00</height>
    <width>13.25</width>
    <rotation>0.0000</rotation>
    <isDynamic>true</isDynamic>
    <shape>SQUARE</shape>
    <physicsandID>0.5,0.5,0.5,'iq'</physicsandID>
    <verts>'()'</verts>
</completeShape>
</level>

```

## 2. 对 IVActivitiy.java 所做的修改如下:

### 为了使用矢量图而修改的 IVActivity.java 代码

```

. . .
@Override
public void onLoadResources() {
    /* 载入 Textures 对象。 */
    this.mTexture = new Texture(512, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    TextureRegionFactory.setAssetBasePath("gfx/IV/");

    /* 载入 TextureRegions 对象。 */
    ITextureSource mSlingTextureSource =
        new SVGAssetTextureSource(this, "gfx/IV/sling.svg", 1.0f);
    ITextureSource mStakeTextureSource =
        new SVGAssetTextureSource(this, "gfx/IV/stake.svg", 1.0f);
    ITextureSource mGlassTextureSource =
        new SVGAssetTextureSource(this, "gfx/IV/glass.svg", 1.0f);
    ITextureSource mStoneTextureSource =
        new SVGAssetTextureSource(this, "gfx/IV/stone.svg", 1.0f);
    ITextureSource mMatHeadTextureSource =
        new SVGAssetTextureSource(this, "gfx/IV/mathead.svg", 1.0f);
    mSlingTextureRegion =
        TextureRegionFactory.createFromSource(this.mTexture,
            mSlingTextureSource, 0, 0);
    mStakeTextureRegion = TextureRegionFactory.createFromSource(
        this.mTexture, mStakeTextureSource, 0, 40);
    mGlassTextureRegion = TextureRegionFactory.createFromSource(

```

```

        this.mTexture, mGlassTextureSource, 0, 80);
mStoneTextureRegion = TextureRegionFactory.createFromSource(
    this.mTexture, mStoneTextureSource, 0, 120);
mMatHeadTextureRegion = TextureRegionFactory.createFromSource(
    this.mTexture, mMatHeadTextureSource, 0, 160);
mWoodTextureRegion = TextureRegionFactory.createFromAsset(
    this.mTexture, getApplicationContext(), "wood.png",
    0, 210);
this.mEngine.getTextureManager().loadTexture(this.mTexture);

```

## 第 13 章

1. 需要对 mStartVamp() 这个 Runnable 任务做出修改, 修改后的代码如下所示:

### 修改后的 mStartVamp 任务代码

```

private Runnable mStartVamp = new Runnable() {
    ...
    asprVamp[i].animate(frameDurations, 0, 25, true);
    float lagTime = gen.nextFloat()*20.0f;
    float startX = asprVamp[i].getX() -
        (lagTime/60.0f) * (asprVamp[i].getX() - 30.0f);
    pathVamp[i] = aStar[i].getPath(startX, 1, asprVamp[i].getY(),
    10, asprVamp[i].getWidth(), asprVamp[i].getHeight());
    asprVamp[i].registerEntityModifier(
        new SequenceEntityModifier (
            new AlphaModifier(5.0f, 0.0f, 1.0f),
            new MoveXModifier(lagTime, asprVamp[i].getX(), startX),
            new PathModifier(60.0f - lagTime, pathVamp[i])
        )
    );
    scene.getLastChild().attachChild(asprVamp[i]);
    ...
}

```

lagTime 的值就是吸血鬼在使用 A\* 寻路法走到 Miss B 家门之前向左移动的时间, 其后吸血鬼所走的路径在设定 Modifier 之前已经提前算好了。当然, 也可以在构造 MoveModifier 对象时传入一个 IEntityModifierListener 对象, 于该监听器对象内再设定 A\* 路径。

2. 下面列出了对 Level1Activity.java 代码所做的修改, 读者也许会怀疑这种代码实现方式对吸血鬼躲避子弹帮不上什么大忙, 毕竟子弹的速度太快了。还有一种方法, 当玩家从武器库中将武器拖放到屏幕上时, 就对吸血鬼发出预警, 不过这么可能做会产生一些不必要的路径查找运算。也许读者自己还有更好的解决办法。

### 修改之后的 Level1Activity.java 代码, 吸血鬼会试着主动躲避子弹

```

...
@Override

```

```

public Scene onLoadScene() {
    ....
    bullet = new Sprite(20.0f, CAMERA_HEIGHT - 40.0f,
        mBulletTextureRegion){
        @Override
        public boolean onAreaTouched(final TouchEvent
            pAreaTouchEvent, final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            switch(pAreaTouchEvent.getAction()) {
            case TouchEvent.ACTION_DOWN:
                break;
            case TouchEvent.ACTION_UP:
                mWarnVampires(pAreaTouchEvent.getY());
                fireBullet(pAreaTouchEvent.getX(),
                    pAreaTouchEvent.getY());
                break;
            case TouchEvent.ACTION_MOVE:
                this.setPosition( pAreaTouchEvent.getX() -
                    this.getWidth() / 2,
                    pAreaTouchEvent.getY() - this.getHeight() / 2);
                break;
            }
            return true;
        }
    };
    ....
    private void mWarnVampires(float pThreatY){
        // 在纵坐标为 pThreatY 的这条横向路径上存在对吸血鬼的潜在威胁。
        Scene scene = Level1Activity.this.mEngine.getScene();

        for (int i=0; i<nVamp; i++){
            if (Math.abs(asprVamp[i].getY() - pThreatY) < 10.0f) {
                asprVamp[i].clearEntityModifiers();
                pathVamp[i] = aStar[i].getPath(asprVamp[i].getX(), 1,
                    asprVamp[i].getY() - 20.0f, 10,
                    asprVamp[i].getWidth(), asprVamp[i].getHeight());
                asprVamp[i].registerEntityModifier(
                    new SequenceEntityModifier (
                        new MoveYModifier(5.0f, asprVamp[i].getY(),
                            asprVamp[i].getY() - 20.0f),
                        new PathModifier(60.0f, pathVamp[i])
                    ));
                scene.getLastChild().attachChild(asprVamp[i]);
            }
        }
    }
}

```

## 第 14 章

## 1. 修改之后的代码如下所示:

## 修改之后的 IVActivity.java 代码

```

private SharedPreferences audioOptions, scores;

public void onLoadResources() {
    SoundFactory.setAssetBasePath("mfx/");
    try {
        this.mOofSound = SoundFactory.createSoundFromAsset(
            this.mEngine.getSoundManager(), this,
            "oof.ogg");
    } catch (final IOException e) {
        Debug.e(e);
    }
}

@Override
public void onLoadComplete() {
    this.mPhysicsWorld.setContactListener(
        new ContactListener() {
            @Override
            public void beginContact(Contact contact) {
                Body bodyA = contact.getFixtureA().getBody();
                Body bodyB = contact.getFixtureB().getBody();
                String idA = (String)bodyA.getUserData();
                String idB = (String)bodyB.getUserData();
                if ((idA.startsWith("vamp")) &&
                    (idB.equals("floor"))) {
                    playSound(mOofSound);
                    int vampID = Integer.parseInt(
                        idA.substring(4, 5));
                    if (!deadHeads.contains(vampID)) {
                        deadHeads.add(vampID);
                    }
                    mAddScore(VAMPIRE_FLOORED);
                    if (deadHeads.size() == numHeads) {
                        mGameOver(PYER_WINS);
                    }
                }
                if ((idB.startsWith("vamp")) &&
                    (idA.equals("floor"))) {
                    playSound(mOofSound);
                    int vampID = Integer.parseInt(
                        idB.substring(4, 5));
                    if (!deadHeads.contains(vampID)) {
                        deadHeads.add(vampID);
                    }
                }
            }
        }
    );
}

```



```

    }
    mAddScore(VAMPIRE_FLOORED);
    if (deadHeads.size() == numHeads) {
        mGameOver(PPLAYER_WINS);
    }
}

public void endContact(Contact contact) {
}

});
}

```

这段代码将音效文件载入系统，然后对 ContactListener 的代码做出修改。修改后的代码如果发现两个碰撞对象分别是吸血鬼头像与地板的话，就会播放音效（如果游戏音效处于打开状态）。

2. 为了在游戏结束画面中增加“New High Score”功能，需要对 Level1Activity.java、IVActivity.java 及 WAVActivity.java 的代码做出如下修改：

修改之后的 Level1Activity.java、IVActivity.java 及 WAVActivity.java 的代码

```

private TextureRegion mNewHighTextureRegion;

public void onLoadResources() {
    mNewHighTextureRegion =
        TextureRegionFactory.createFromAsset(
            this.mPopUpTexture, getApplicationContext(),
            "newhigh.png", 100, 400);
}

public Scene onLoadScene() {
    newHigh = new Sprite(0.0f, 0.0f, mNewHighTextureRegion);

    private void mGameOver(boolean pWin) {
        // 此方法在游戏结束时会被调用，pWin 的值如果是 true，则表示玩家获胜
        Scene scene = WAVActivity.this.mEngine.getScene();
        boolean newTop = false;
        int[] newHighScores = {0, 0, 0, 0, 0};
        for (int i=4; i>=1; i--) {
            if (thisScore > highScores[i]) {
                newHighScores[i] = thisScore;
                for (int j=i-1; j>=1; j--) {
                    newHighScores[j] = highScores[j+1];
                }
                if (i==4) newTop = true;
                break;
            } else {

```

```

        newHighScores[i] = highScores[i];
    }
    for (int i=0; i<5; i++) highScores[i] = newHighScores[i];
    scoresEditor.putInt("WhAV-4", highScores[4]);
    scoresEditor.putInt("WhAV-3", highScores[3]);
    scoresEditor.putInt("WhAV-2", highScores[2]);
    scoresEditor.putInt("WhAV-1", highScores[1]);
    scoresEditor.putInt("WhAV-0", highScores[0]);
    scoresEditor.commit();

    if (pWin){
        scene.setChildScene(mCreateEndScene(newTop, true,
        "Congratulations!!"), false, true, true);
    } else {
        scene.setChildScene(mCreateEndScene(false, false,
        "You Suck!\n      blood"));
    }
}

private Scene mCreateEndScene(boolean pNewHigh, boolean pWin,
String pTitle){
    Scene endScene = new Scene(2);
    endScene.getLastChild().attachChild(endBack);
    Text mTitle = new Text( 50.0f, 50.0f, mFont32, pTitle);
    endScene.getLastChild().attachChild(mTitle);
    if (pNewHigh){
        newHigh.setPosition(300.0f, 50.0f);
        endScene.getLastChild().attachChild(newHigh);
    }
    Text mYourScore = new Text( 50.0f, 150.0f, mFont32,
    "Your Score: " + thisScore);
}

```

这段代码用一张含有“New High Score”字样的图片创建了相关的 TextureRegion 对象。在游戏结束计算分数时，检查玩家当前所获的分数是否比数组中的所有分数都高。如果当前分数是最高分的话，就以 true 作为 pNewHigh 参数的值来调用 mCreateEndScene() 方法，该方法的代码会将表示新高分的精灵对象加入场景中。

## 第 15 章

■ 这道习题又一次说明了将游戏资源在动态壁纸等场合中重用是多么方便。

### 带有爆炸效果的 V3LiveWallpaper.java 代码

```

package com.pearson.lagp.vinb;
+ imports
+
+
+ // =====

```

```

// 字段
// =====

private ParticleSystem particleSystem;
private BaseParticleEmitter particleEmitter;

@Override
public Scene onLoadScene() {

    scene.registerUpdateHandler(new IUpdateHandler() {
        @Override
        public void reset() { }

        @Override
        public void onUpdate(final float pSecondsElapsed) {
            for (int i=0; i<nVamp; i++){
                if (asprVamp[i].getX() < 30.0f) {
                    // 显示爆炸效果
                    particleEmitter.setCenter(
                        asprVamp[i].getX(),
                        asprVamp[i].getY());
                    particleSystem.setParticles-
                        SpawnEnabled(true);
                    mHandler.postDelayed(
                        mEndPESpawn, 2000);
                    // 让吸血鬼重新出现在屏幕右方
                    float startY = gen.nextFloat() *
                        (CAMERA_HEIGHT - 50.0f);
                    asprVamp[i].clearEntityModifiers();
                    asprVamp[i].registerEntityModifier(
                        new MoveModifier(40.0f,
                            CAMERA_WIDTH - 30.0f, 0.0f,
                                startY, 340.0f)
                    );
                }
            }
        }
    });
    try {
        final PXLoader pxLoader = new PXLoader(this,
            this.mEngine.getTextureManager(),
            TextureOptions.BILINEAR_PREMULTIPLYALPHA );
        particleSystem = pxLoader.createFromAsset(this,
            "gfx/particles/explo.px");
    } catch (final PXLoadException pxle) {
        Debug.e(pxle);
    }
    particleSystem.setBlendFunction(GL10.GL_SRC_ALPHA,
        GL10.GL_ONE);
    particleSystem.setParticlesSpawnEnabled(false);
    particleEmitter =
        (BaseParticleEmitter) particleSystem.getParticleEmitter();
}

```

```

        scene.getLastChild().attachChild(particleSystem);
    }

    private Runnable mEndPESpawn = new Runnable() {
        public void run() {
            particleSystem.setParticlesSpawnEnabled(false);
        }
    };

```

这个类使用了一部分 Level1Activity.java 中的爆炸粒子效果代码，并将它增加到动态壁纸服务的逻辑中。只要有吸血鬼到达屏幕左侧，也就是 Miss B 的屋子那里，粒子效果就会激活。除了修改这个类的代码，还需要向项目中加入如下文件：

❑ 用于加载粒子效果的文件必须放入项目的 src 文件夹下：

- PXLoader.java
- PXParser.java
- PXConstants.java
- PXLoaderException.java
- PXParserException.java

❑ 定义粒子效果的 XML 文件 explo.px 必须放入 assets/gfx/particles 文件夹下。

❑ 粒子效果所使用的 PNG 文件 particle\_fire.png 必须放入 assets/gfx/Wallpaper 文件夹下。

2. 本书并没有对 MOD 文件进行再次发布的权利，不过读者可以在 <http://modarchive.org> 这样的网站中免费下载并使用 MOD 文件。程序清单 15.3 中给出了将 assets 文件夹下的 mod 文件移动到 SD 卡中的方法。再多说一句，请修改 startPlayingMod() 方法的代码，让它根据 SharedPreferences 中存储的相关 Boolean 值来决定是否要播放音乐。

3. 修改后的代码如下所示。请注意这几处重要修改：

❑ 令该类实现 IOrientationListener 这个接口。

❑ 增加一个用于表示手机方向的 ChangeableText 对象。

❑ 覆写 onOrientationChanged() 回调方法。

❑ 覆写 onPause() 与 onResume() 方法，以便根据程序运行状态来决定是否打开方向侦测功能。

❑ 使用 NumberFormat 对象来限制方向读数的显示。

#### 增加了手机方向显示功能的增强现实游戏代码

```

package com.pearson.lagp.vinb;

+imports

public class VampiresInBackyard extends BaseAugmentedRealityGameActivity
    implements IOrientationListener {

```

```

. . .
// =====
// 字段
// =====
. . .
private Texture mFontTexture;
private Font mFont32;
private ChangeableText mCurrHeading;

. . .
@Override
public Engine onLoadEngine() {
. . .
    nf = NumberFormat.getInstance();
    nf.setMaximumFractionDigits(2);
    nf.setMinimumFractionDigits(2);
. . .
}

@Override
public void onLoadResources() {
. . .
    this.mFontTexture = new Texture(256, 512,
        TextureOptions.BILINEAR_PREMULTIPLYALPHA);
    FontFactory.setAssetBasePath("font/");
    mFont32 = FontFactory.createFromAsset(
        this.mFontTexture, this, "Flubber.ttf",
        32, true, Color.RED);
    mEngine.getTextureManager().loadTexture(
        this.mFontTexture);
    mEngine.getFontManager().loadFont(this.mFont32);
}

@Override
public void onLoadScene()
. . .
    mCurrHeading = new ChangeableText(0.5f * CAMERA_WIDTH, 10.0f,
        mFont32, "Heading: 0", "Heading: XXXXXX".length());
    scene.getLastChild().attachChild(mCurrHeading);
. . .
@Override
protected void onResume() {
    super.onResume();
    this.enableOrientationSensor(VampiresInBackyard.this);
}

@Override
protected void onPause() {
    super.onPause();
    this.mEngine.disableOrientationSensor(this);
}

```



```

    }
    public void onOrientationChanged(OrientationData pOD) {
        // 更新方向显示
        float head = pOD.getYaw();
        mCurrHeading.setText("Heading: " + nf.format(head));
    }
}

```

## 第 16 章

### 1. 修改之后的 Level1Activity.java 代码如下所示。

修改之后的 Level1Activity.java, 它会在程序暂停时停止相关的 Runnable 任务

```

...
@Override
public void onGamePaused() {
    super.onGamePaused();
    mGunshotSound.stop();
    mExploSound.stop();
    mOCSMusic.stop();
    mSaveMeSound.stop();
    mActivityVisible = false;
    mHandler.removeCallbacks(mStartSarah);
    mHandler.removeCallbacks(mStartVamp);
}

@Override
public void onGameResumed() {
    super.onGameResumed();
    mActivityVisible = true;
    mHandler.removeCallbacks(mStartSarah);
    mHandler.removeCallbacks(mStartVamp);
    mHandler.postDelayed(mStartVamp, mVampRate);
    if (mDistract) mHandler.postDelayed(mStartSarah, 5000);
}
...

```

在 onGamePaused() 方法中调用 removeCallbacks() 方法是应该的, 然而为什么在 onGameResumed() 方法中还要调用它呢? 因为作为 Activity 启动过程的一部分, onGameResumed() 方法在游戏启动的时候也会执行一次。我们并不想让 Sarah 的头像与吸血鬼精灵在 onLoadScene() 方法执行完毕之前就提前出现, 所以要在该方法内先调用一次 removeCallbacks() 方法。

2. 对 OptionsActivity.java 所做的修改很简单: 就是将代表 WAV 与 IV 小游戏的 menuItems 对象删掉即可。在本章所附的 V316Exercises 项目代码中, 这两个菜单项的代码都被注释掉了。对 StartActivity.java 代码所做的修改如下所示。



修改之后的 StartActivity.java, 它加入了两个不可见的按钮

```

. . .
// =====
// 字段
// =====
. . .
private TextureRegion mBlackButtonTextureRegion;
. . .
@Override
public void onLoadResources() {
. . .
    this.mBlackButtonTextureRegion =
        TextureRegionFactory.createFromAsset(this.mTexture,
            this, "blackbutton.png", 0, 330);
. . .
@Override
public Scene onLoadScene() {
. . .
    /* 创建用于启动 WAV 与 IV 小游戏的按钮对象。 */
    final Sprite WAVButton = new Sprite(CAMERA_WIDTH - 32,
        CAMERA_HEIGHT - 64, mBlackButtonTextureRegion) {
        @Override
        public boolean onAreaTouched(
            final TouchEvent pAreaTouchEvent,
            final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            mHandler.removeCallbacks(mLaunchMenuTask);
            mHandler.post(mLaunchWAVTask);
            return true;
        }
    };
    mScene.registerTouchArea(WAVButton);
    mScene.setTouchAreaBindingEnabled(true);
    mScene.getLastChild().attachChild(WAVButton);

    final Sprite IVButton = new Sprite(CAMERA_WIDTH - 32,
        CAMERA_HEIGHT - 32, mBlackButtonTextureRegion) {
        @Override
        public boolean onAreaTouched(
            final TouchEvent pAreaTouchEvent,
            final float pTouchAreaLocalX,
            final float pTouchAreaLocalY) {
            mHandler.removeCallbacks(mLaunchMenuTask);
            mHandler.post(mLaunchIVTask);
            return true;
        }
    };
    mScene.registerTouchArea(IVButton);
    mScene.getLastChild().attachChild(IVButton);
. . .

```

```

@Override
public void onDestroy() {
    super.onDestroy();
    mHandler.removeCallbacks(mLaunchMenuTask);
    mHandler.removeCallbacks(mLaunchWAVTask);
    mHandler.removeCallbacks(mLaunchIVTask);
}

...

private Runnable mLaunchWAVTask = new Runnable() {
    public void run() {
        Intent myIntent = new Intent(StartActivity.this,
            WAVActivity.class);
        StartActivity.this.startActivity(myIntent);
    }
};

private Runnable mLaunchIVTask = new Runnable() {
    public void run() {
        Intent myIntent = new Intent(StartActivity.this,
            IVActivity.class);
        StartActivity.this.startActivity(myIntent);
    }
};

```

3. 可以使用 AlertDialog 对话框来显示游戏帮助, 帮助对话框所用的文本已经加入到项目 res 文件夹下的 strings.xml 文件中了。针对 MainMenuActivity.java 所做的修改如下所示。

#### 增加了显示游戏帮助功能之后的 MainMenuActivity.java 代码

```

@Override
public boolean onOptionsItemSelected(final MenuScene pMenuScene,
    final IMenuItem pMenuItem, final float pMenuItemLocalX,
    final float pMenuItemLocalY) {
    ...

    case MENU_HELP:
        mShowHelp();
        return true;
    ...

    private void mShowHelp() {
        String title = this.getString(R.string.app_name);
        String message = this.getString(R.string.help);

        AlertDialog.Builder builder = new AlertDialog.Builder(this)
            .setTitle(title)
            .setCancelable(false)
            .setMessage(message)
    }
}

```

```
        .setPositiveButton(R.string.help_dialog_play,
new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int whichButton) {
        mMainScene.registerEntityModifier(
new ScaleAtModifier(0.5f, 1.0f, 0.0f, CAMERA_WIDTH/2,
CAMERA_HEIGHT/2));
        mStaticMenuScene.registerEntityModifier(
new ScaleAtModifier(0.5f, 1.0f, 0.0f, CAMERA_WIDTH/2,
CAMERA_HEIGHT/2));
        mHandler.postDelayed(mLaunchLevel1Task, 500);
    }
})
        .setNegativeButton(R.string.help_dialog_cancel,
new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int whichButton) {
        dialog.dismiss();
    }
});
builder.create().show();
}
```

---