移动开发系列丛书

索尼爱立信移动通信（中国）有限公司
Android研发工程师 魏鹏飞
乐视网Android高级开发工程师 于庭龙
新华网项目经理 段洁男
用友软件华北区技术总监 李迪锋
eoeAndroid
51CTO 博客园
ZD.NET
鼎力推荐

Google Android 游戏开发大全

吴亚峰 苏亚光 编著 百纳科技 审校

涵盖游戏开发的全部技术，如前台渲染、界面开发步骤进开发步骤进行介绍，并逐人工智能算法、流行的物理引擎及OpenGL ES 3D

包括经典的8类游戏，如体育类、益智类、滚个小时的配个小时的配书视频讲解 苦防类、物理引擎类及策略类。

揭秘“愤怒的小鸟”的开发技术—“盛怒的老鼠”，结合Box2D物理引擎与2D渲染技术详细介绍“愤怒的小鸟”这类物理引擎游戏的开发技术与技巧

每一个游戏案例都是按照实际的开发步骤进行介绍，并逐步地将游戏开发的整个过程展现在读者面前。

配书光盘赠送全部源程序，10多个小时的配书视频讲解

人民邮电出版社
POSTS & TELECOM PRESS

视频与源程序

Google
Android
游戏开发大全

棋牌类游戏案例——中国象棋人机对弈，结合真正的AI（人工智能）向读者揭示了对弈类游戏AI模块的核心开发技术。

体育类游戏案例——2D迷你桌球，给出了一个桌球类游戏的完整解决方案，包括前台渲染呈现、后台物理引擎等核心技术，具有很高的实际参考价值。

益智类案例——3D推箱子，给出了一个完整3D游戏案例，详细介绍了3D模型、纹理贴图、摄像机变换等多种3D游戏开发中必知必会的知识。

物理引擎类案例——“盛怒的老鼠”，结合Box2D物理引擎与2D渲染技术，揭秘了“愤怒的小鸟”这类物理引擎游戏的开发思路与核心实现技术。

本书支持社区

分类建议：计算机/程序设计/移动开发
人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-26493-0
定价：88.00 元（附光盘）

移动开发系列丛书

Google Android 游戏开发大全

吴亚峰 苏亚光 编著 百纳科技 审校

人民邮电出版社
北京

图书在版编目（CIP）数据

Android游戏开发大全 / 吴亚峰，苏亚光编著. --
北京 : 人民邮电出版社，2012.1
ISBN 978-7-115-26493-0

Ⅰ. ①A… Ⅱ. ①吴… ②苏… Ⅲ. ①移动电话机－游戏程序－程序设计 Ⅳ. ①TN929.53②TP317

中国版本图书馆CIP数据核字(2011)第199409号

内 容 提 要

本书以 Android 手机游戏的开发为主题，结合真实的游戏案例向读者详细介绍了 Android 平台下游戏开发的整个流程，同时在游戏开发的介绍过程中还分享了作者多年积累的开发技巧与经验。

全书分为两篇，共 19 章，第 1 篇（第 1～11 章）以简单易懂的实例为依托，详细介绍了 Android 的前台渲染技术、常用界面布局、网络开发、SQLite 数据库的存取、传感器应用的开发、游戏中经常使用的人工智能算法、流行的物理引擎 Box2D 以及 OpenGL ES 3D 开发的相关知识。第 2 篇（第 12～19 章）详细介绍了 8 个真实的游戏案例，涵盖了手机游戏中的 8 个不同种类，包括体育类、益智类、滚屏动作类、棋牌类、物理传感器类、塔防类、物理引擎类以及策略类游戏。每一个游戏都是按照实际的开发步骤进行介绍，并逐步地将 Android 手机游戏的整个开发过程展现在读者面前。同时在源代码中还包含了详细的注释，以尽量帮助读者掌握代码中的每一个细节。另外，第 2 篇中所有的案例项目都配有讲解视频，以帮助读者快速理解并得到提升。

本书的讲述由浅入深，从游戏开发的基础知识到开发大型游戏，结构清晰、语言简洁，非常适合 Android 游戏初学者和 Android 游戏进阶开发者阅读参考。

Android 游戏开发大全

◆ 编　　著　吴亚峰　苏亚光
　审　　校　百纳科技
　责任编辑　张　涛
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
　邮编　100061　　电子邮件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京鑫正大印刷有限公司印刷
◆ 开本：800×1000　1/16
　印张：43.25　　彩插：2
　字数：1054 千字　　2012 年 1 月第 1 版
　印数：1 – 4 000 册　　2012 年 1 月北京第 1 次印刷

ISBN 978-7-115-26493-0

定价：88.00 元（附光盘）

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

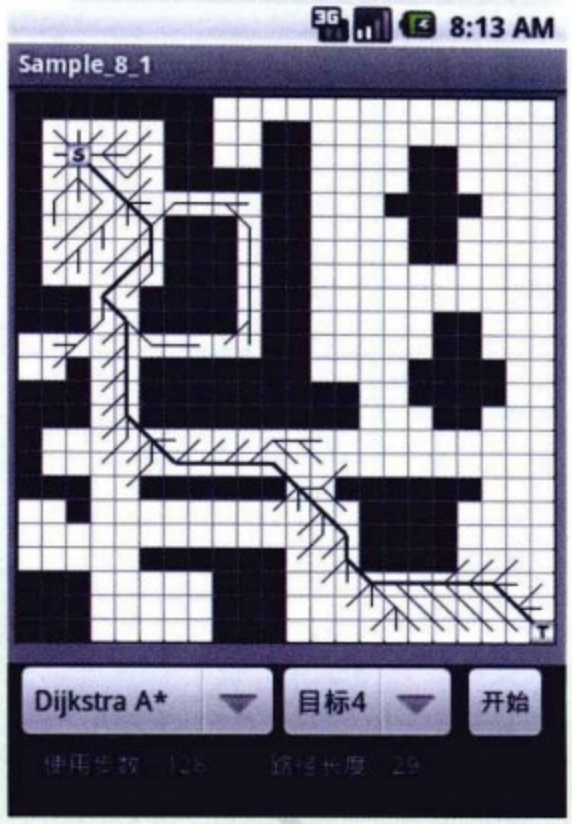
本书案例

第5章 Android游戏开发之网络编程

开发亮点：Socket、HTTP、Google Map的应用

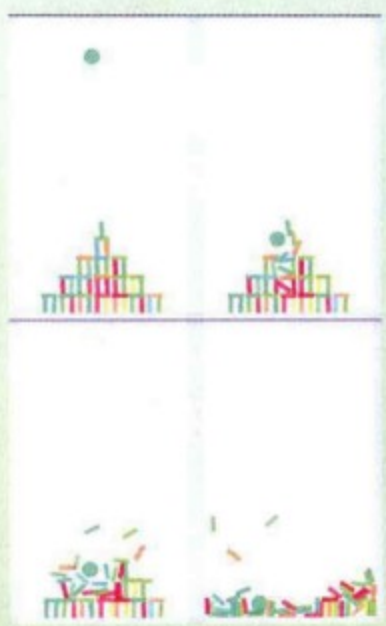
第7章 游戏背后的数学与物理

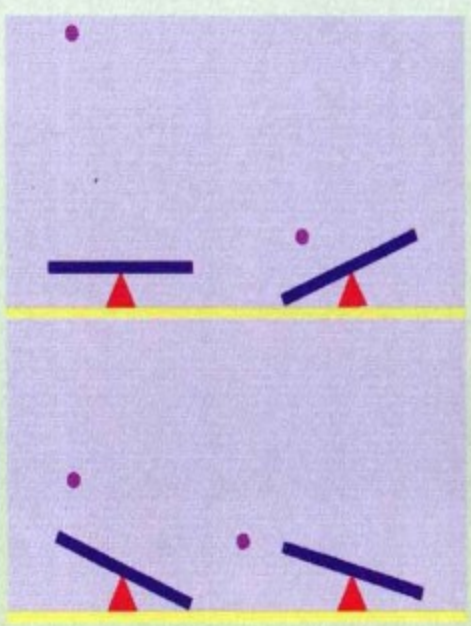
开发亮点：游戏开发过程中所涉及的数学与物理知识，常用碰撞检测技术的实现



第8章 游戏中的人工智能

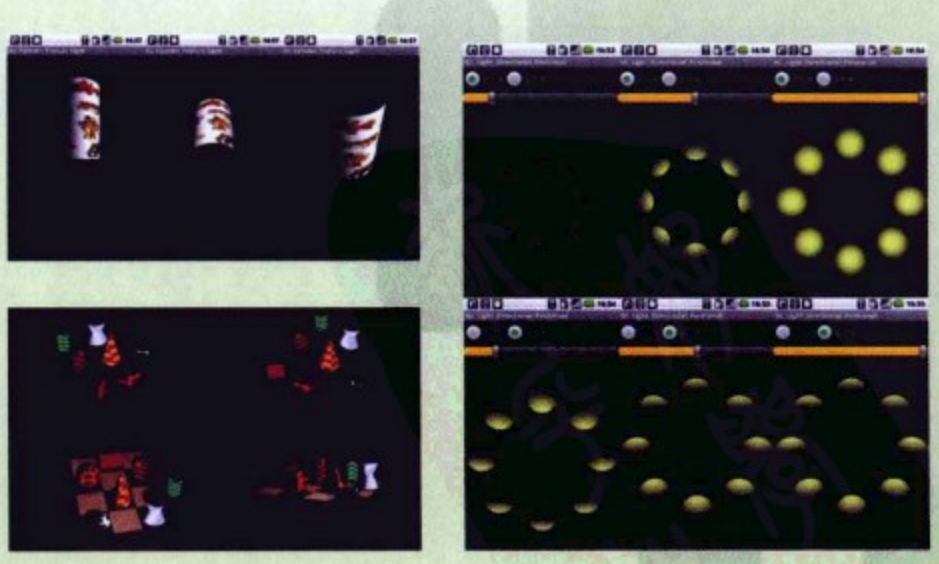
开发亮点：3种路径搜索算法及A*优化，有限状态机的实现





第10章 游戏的心脏——物理引擎

开发亮点：揭秘风靡全球的“愤怒的小鸟”物理引擎Box2D原理及使用



第11章 OpenGL ES应用开发基础

开发亮点：OpenGL ES 3D基础知识，酷炫的3D应用

本书案例

第12章 滚屏动作游戏——太空保卫战

开发亮点：动画的搭建、SoundPool声音池的使用及图片的切割技术

第13章 棋牌游戏——中国象棋人机对弈

开发亮点：揭秘棋牌类游戏的开发思路、人机对弈AI算法的开发技巧

第14章 物理传感器游戏——小球快跑

开发亮点：传感器体感游戏的开发，Android平台特有传感器模块的使用

第15章 塔防游戏——精灵塔防

开发亮点：塔防类游戏的开发流程及技巧，界面设计，动画渲染

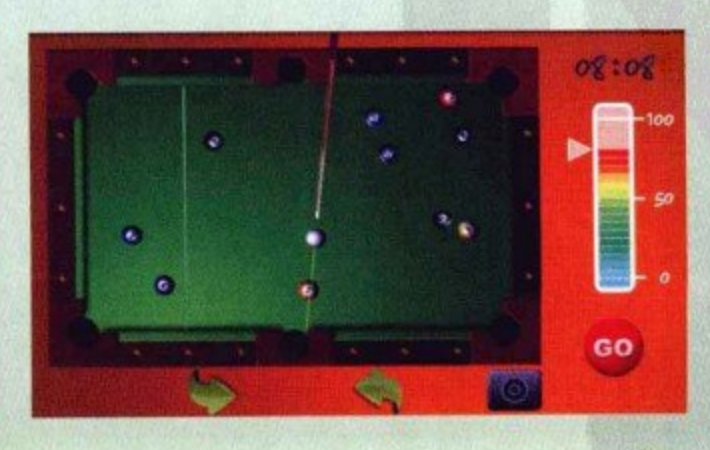
本书案例

第16章 策略游戏——回到战国

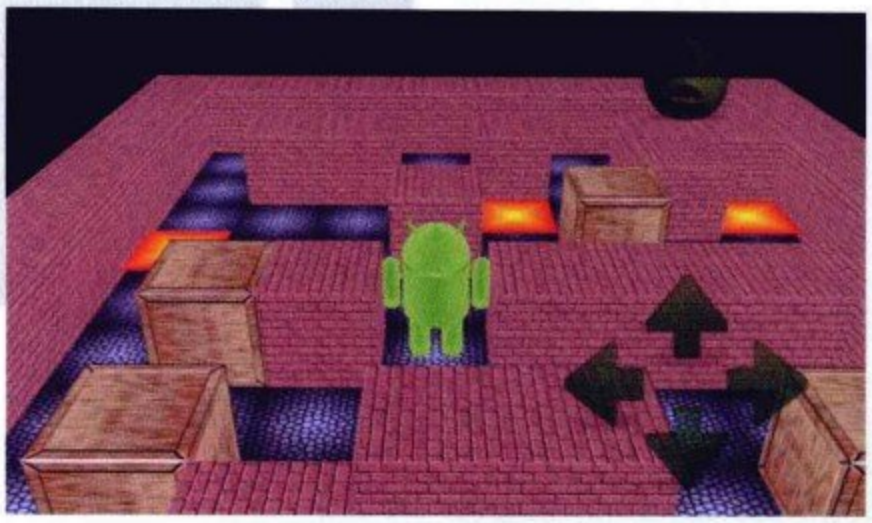
开发亮点：策略类游戏的开发，大型RPG游戏的设计模式，大型地图的特有开发技术

第17章 体育游戏——2D迷你桌球

开发亮点：桌球游戏开发、自定义物理引擎、2D技术实现3D的视觉效果



本书案例

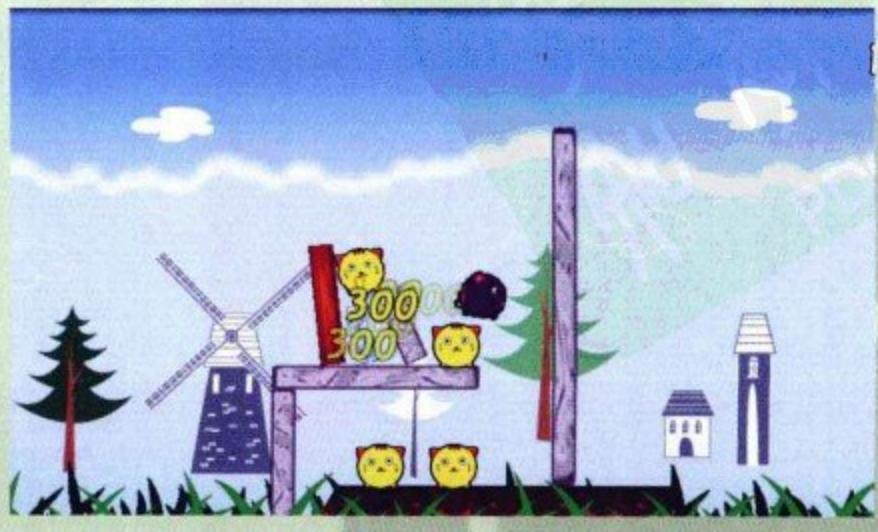


第18章 益智游戏——3D推箱子

开发亮点：益智游戏的开发、OpenGL ES 3D技术、炫酷3D游戏的开发技巧

第19章 物理引擎游戏——盛怒的老鼠

开发亮点：权威揭秘Box2D物理引擎与2D渲染技术开发“愤怒的小鸟”这类物理引擎游戏的核心技术与技巧



前　言

为什么要写一本这样的书

Android 正以前所未有的速度聚集着来自世界各地的开发者，越来越多的创意被应用到 Android 应用程序的开发中，大有席卷整个手机产业的趋势。

面对如此火爆的 Android 大潮，一些有关 Android 的技术书籍也开始登上各地书架。但是纵观这些本来就为数不多的 Android 书籍，却没有一本是关于 Android 游戏开发专题的，而手机游戏历来是移动开发中最重要的部分。

本书正是在这种情况下应运而生的，作为国内第一本讲解 Android 游戏案例开发的专业书籍，作者为这本书倾注了很多的心血。书中既有对 Android 应用程序框架的介绍，也有对游戏开发相关知识的讲解，同时还有多个 Android 平台下的实际游戏案例。希望可以快速帮助读者提高在 Android 平台下进行游戏开发的能力。

内容导读

本书内容分为两篇，第 1 篇主要介绍 Android 平台下应用程序的框架和基础开发知识，同时还介绍了游戏开发相关的知识，主要内容安排如下。

章　名	主 要 内 容
第 1 章　Android 平台简介	介绍 Android 的来龙去脉，并介绍 Android 应用程序的框架，然后对 Android 的开发环境进行搭建和调试
第 2 章　Android 游戏开发之前台渲染	对 Android 的用户界面进行详细介绍，同时讲解图形、动画、音频、视频的实现，并对图像采集技术进行讲述
第 3 章　Android 游戏开发之交互式通信	简要介绍应用程序的基本组件，详细介绍应用程序内部或组件之间的通信方式
第 4 章　Android 游戏开发之数据存储和传感器	通过实例介绍 Android 平台下 SQLite 数据库与传感器的原理及使用方法
第 5 章　Android 游戏开发之网络编程	首先对 Socket 套接字以及 HTTP 进行介绍，然后通过一个实例讲解 Google Map 应用的实现
第 6 章　不一样的游戏，一样的精彩	介绍不同类型游戏的特色及开发特点
第 7 章　游戏背后的数学与物理	介绍游戏开发过程中所涉及的数学与物理的相关知识，并详细讲解了碰撞检测技术的实现方式
第 8 章　游戏中的人工智能	首先对 3 种基本路径搜索算法进行介绍，并对其进行 A*优化；然后通过简单的实例介绍有限状态机的实现

续表

章　名	主要内容
第 9 章　游戏开发小秘技	通过地图编辑器的开发与设计，进一步加深对游戏开发的理解，同时对游戏的模糊逻辑进行简单介绍
第 10 章　游戏的心脏——物理引擎	通过对风靡全球的物理引擎游戏“愤怒的小鸟”所采用的物理引擎 Box2D 的详细介绍，带领读者进入游戏物理引擎的世界
第 11 章　OpenGL ES 应用开发基础	通过对在 Android 平台下进行 3D 开发的 OpenGL ES 必知必会知识的讲解，带领读者进入酷炫的 3D 手机世界的开发

第 2 篇主要内容为 8 个游戏案例，其中包括体育游戏、益智游戏、动作游戏、RPG 游戏等不同的游戏类型，每种游戏类型的开发都有其独特的地方，具体安排如下。

章　名	主要内容
第 12 章　滚屏动作游戏——太空保卫战	详细介绍了简单动画的搭建方法、SoundPoll 声音的使用方法以及图片的切割技术
第 13 章　棋牌游戏——中国象棋人机对弈	通过人机对弈象棋软件的开发，讲解棋牌类游戏的开发思路以及对弈算法的基本开发流程
第 14 章　物理传感器游戏——小球快跑	详细讲解传感器游戏“小球快跑”的开发，介绍 Android 平台特有传感器模块的使用
第 15 章　塔防游戏——精灵塔防	通过一个具体的案例，详细介绍了非常流行的塔防类游戏的开发流程及技术要点
第 16 章　策略游戏——回到战国	通过介绍策略类游戏回到战国的开发，详细讲解大型 RPG 游戏的设计模式，并综合应用了之前介绍过的知识
第 17 章　体育游戏——2D 迷你桌球	详细介绍了桌球游戏开发中需要用到的碰撞引擎及如何用 2D 的技术实现 3D 的视觉效果
第 18 章　益智游戏——3D 版推箱子	通过益智游戏推箱子的开发，详细深入地介绍了如何在 Android 平台上使用 OpenGL ES 技术开发出炫酷的 3D 游戏
第 19 章　物理引擎游戏——盛怒的老鼠	通过结合 Box2D 物理引擎与 Android 平台下的 2D 渲染技术，向读者揭秘了类似于“愤怒的小鸟”这类物理引擎游戏的开发思路与技术

本书特点

1. 内容饱满，由浅入深

本书内容既包括 Android 平台下开发的基础知识，也有游戏编程的实用技巧，同时还有多个游戏实际案例供读者学习。在知识的层次上由浅入深，真正地将 Android 和游戏开发结合起来。

2. 实例丰富，讲解详细

本书在介绍 Android 基础内容时，每个知识点都配有相应的实例，通过这些实例，读者可以更

好地理解书中所介绍的知识。同时在实例的讲解上也尽量做到条理清楚，读者可以按照书中列出的步骤非常容易地实现实例中的功能。

3. 案例经典，含金量高

本书中的游戏案例均是作者精心挑选的，不同类型的游戏有着其独特的开发方式。本书中的案例囊括了不同的游戏类型，以及不同的游戏开发技巧，以期让读者全面地掌握手机游戏的开发，具有很高的含金量，非常适合各类读者学习。

4. 配合光盘，视频教学

为了帮助读者更好地利用本书提高自己的开发技能，本书附赠了 DVD 光盘，其中包含了本书中所有实例和游戏案例的源代码、电子教案和教学视频。读者可以参考 DVD 光盘中的内容提高学习效率。

本书面向的读者

- **Android 初学者**

对于 Android 的初学者，可以通过本书第 1 篇的内容巩固 Android 的知识，并了解与游戏开发相关的诸如人工智能和物理引擎等知识。然后以此为踏板学习本书第 2 篇的游戏案例，这样可以全面地掌握 Android 平台下游戏开发的技巧。

- **有 Java 基础的读者**

Android 平台下的开发基于 Java 语言，所以对于有 Java 基础的读者来说，阅读本书将不会感觉到困难。读者可以通过第 1 篇的基础内容迅速熟悉 Android 平台下应用程序的框架和开发流程，然后通过第 2 篇的游戏案例提高自己在游戏开发方面的能力。

- **在职开发人员**

本书中的游戏案例都是作者精心挑选的，其中涉及的与游戏开发相关的知识均是作者多年来积累的经验与心得体会。具有一定开发经验的在职开发人员可以通过本书进一步提高开发水平，并迅速转职成为 Android 的游戏开发人员。

关于作者

吴亚峰，毕业于北京邮电大学，后留学澳大利亚卧龙岗大学取得硕士学位。1998 年开始从事 Java 的开发，有十多年的 Java 开发与培训经验。主要的研究方向为手机游戏及 3D、Java EE、搜索引擎，现在为手机游戏、Java EE 独立软件开发工程师，同时兼任 Sun 授权 Java 培训中心认证教师。

为数十家著名企业培训了上千名软件开发人员。2008 年开始关注 Android 平台下的游戏开发，并开发了一系列的游戏娱乐应用。

苏亚光，Java EE 独立软件工程师，有 8 年软件领域的从业经验，在软件开发和计算机教学方面有着丰富的经验。早期主要从事底层软件的设计与开发工作，现主要从事 ARM 平台下嵌入式程序的设计、开发等方面的工作。同时，在 SOA 领域也做了很多有益的工作。

致谢

本书在编写过程中得到了唐山百纳科技有限公司 Java 培训中心的大力支持，同时于庭龙、魏鹏飞、李腾飞、王旅波、李胜杰、郭超、王思维、仇磊、夏学良、冯儒涛、王海峰以及作者的家人为本书的编写提供了很多帮助，在此表示衷心感谢！

由于作者水平有限，书中疏漏之处再所难免，欢迎广大读者批评指正。

编　者

目　录

第一篇　Android 游戏开发核心技术

第 1 章　Android 平台简介 ……2
1.1　Android 的来龙去脉 ……2
1.2　掀起 Android 的盖头来 ……2
1.2.1　选择 Android 的理由 ……2
1.2.2　Android 的应用程序框架 ……3
1.3　Android 开发环境的搭建 ……6
1.3.1　SDK 的下载及安装 ……6
1.3.2　Eclipse 集成开发环境的搭建 ……6
1.3.3　虚拟设备的创建与模拟器的运行 ……9
1.3.4　第一个 Android 程序 ……11
1.3.5　Android 程序的监控与调试 ……14
1.4　小结 ……15
第 2 章　Android 游戏开发之前台渲染 ……16
2.1　创建 Android 用户界面 ……16
2.1.1　布局管理 ……16
2.1.2　常用控件及其事件处理 ……22
2.2　图形与动画在 Android 中的实现 ……24
2.2.1　简单图形的绘制 ……24
2.2.2　贴图的艺术 ……26
2.2.3　自定义动画的播放 ……27
2.3　Android 平台下的多媒体开发 ……30
2.3.1　音频的播放 ……30
2.3.2　视频的播放 ……33
2.3.3　Camera 图像采集 ……36
2.4　小结 ……38
第 3 章　Android 游戏开发之交互式通信 ……39
3.1　Android 应用程序的基本组件 ……39
3.1.1　Activity 组件 ……39
3.1.2　Service 组件 ……41
3.1.3　Broadcast Receiver 组件 ……42
3.1.4　Content Provider 组件 ……43
3.1.5　AndroidManifest.xml 文件简介 ……43
3.2　应用程序的内部通信 ……47
3.2.1　消息的处理者——Handler 类简介 ……47
3.2.2　使用 Handler 进行内部通信 ……48
3.3　应用程序组件之间的通信 ……50
3.3.1　Intent 类简介 ……50
3.3.2　应用程序组件——IntentFilter 类简介 ……52
3.3.3　示例 1：与 Android 系统组件通信 ……53
3.3.4　示例 2：应用程序组件间通信示例 Activity 部分的开发 ……54
3.3.5　示例 3：应用程序组件间通信示例 Service 部分的开发 ……56
3.4　小结 ……59
第 4 章　Android 游戏开发之数据存储和传感器 ……60
4.1　在 Android 平台上实现数据存储 ……60
4.1.1　私有文件夹文件的写入与读取 ……60
4.1.2　读取 Resources 和 Assets 中的文件 ……63
4.1.3　轻量级数据库 SQLite 简介 ……65
4.1.4　SQLite 的使用示例 ……69
4.1.5　数据共享者——Content Provider 的使用 ……72
4.1.6　简单的数据存储——Preferences 的使用 ……76

4.2 Android 平台下传感器应用的开发……78
4.2.1 传感器应用开发流程……78
4.2.2 常用传感器简介……79
4.2.3 传感器应用开发示例……81
4.2.4 使用 SensorSimulator 模拟传感器变化……83
4.2.5 使用新版本的 API 开发传感器应用……86
4.3 小结……88
第 5 章 Android 游戏开发之网络编程……89
5.1 基于 Socket 套接字的网络编程……89
5.2 基于 HTTP 协议的网络编程……93
5.2.1 通过 URL 获取网络资源……93
5.2.2 在 Android 中解析 XML……95
5.3 Android 平台下的 Google Map……95
5.3.1 定位类 GenPoint 与显示地图类 MapView 的搭配使用……95
5.3.2 位置监听器——LocationListener 的使用……99
5.4 其他网络通信方式……101
5.4.1 使用 WiFi 进行开发……101
5.4.2 借助于蓝牙（Bluetooth）技术进行开发……102
5.5 小结……102
第 6 章 不一样的游戏，一样的精彩……103
6.1 射击类游戏……103
6.1.1 游戏玩法……103
6.1.2 视觉效果……104
6.1.3 游戏内容设计……104
6.2 竞速类游戏……104
6.2.1 游戏玩法……105
6.2.2 视觉效果……105
6.2.3 游戏内容设计……106
6.3 益智类游戏……106
6.3.1 游戏玩法……106
6.3.2 视觉效果……107
6.3.3 游戏内容设计……107
6.4 角色扮演游戏……107
6.4.1 游戏玩法……108
6.4.2 视觉效果……108
6.4.3 游戏内容设计……109
6.5 闯关动作类游戏……109
6.5.1 游戏玩法……109
6.5.2 视觉效果……110
6.5.3 游戏内容设计……110
6.6 冒险游戏……110
6.6.1 游戏玩法……110
6.6.2 视觉效果……111
6.6.3 游戏内容设计……112
6.7 策略游戏……112
6.7.1 游戏玩法……112
6.7.2 视觉效果……113
6.7.3 游戏内容设计……113
6.8 养成类游戏……113
6.8.1 游戏玩法……114
6.8.2 视觉效果……114
6.8.3 游戏内容设计……114
6.9 经营类游戏……115
6.9.1 游戏玩法……115
6.9.2 视觉效果……115
6.9.3 游戏内容设计……116
6.10 体育类游戏……116
6.10.1 游戏玩法……116
6.10.2 视觉效果……117
6.10.3 游戏内容设计……117
6.11 小结……117
第 7 章 游戏背后的数学与物理……118
7.1 编程中经常用到的数理知识……118
7.1.1 数学方面……118
7.1.2 物理方面……120
7.2 物理小球在游戏中的应用……121
7.2.1 开发运动体 Movable 类的代码……121
7.2.2 开发物理引擎 BallThread 类的代码……123

7.2.3 视图类——开发 BallView 类的代码……126
7.2.4 绘制线程——开发 DrawThread 类的代码……129
7.2.5 开发 Activity 部分的代码……131
7.3 粒子系统……132
7.3.1 粒子对象类——开发粒子对象 Particle 类和粒子集合 ParticleSet 类……132
7.3.2 开发焰火粒子系统的物理引擎 ParticleThread 类的代码……134
7.3.3 视图类——开发视图类 ParticleView 及其相关类……135
7.3.4 开发程序 Activity 部分的代码……137
7.3.5 将焰火粒子系统改为瀑布粒子系统……138
7.4 碰撞检测技术……139
7.4.1 碰撞检测技术基础……139
7.4.2 游戏中实体对象之间的碰撞检测……140
7.4.3 游戏实体对象与环境之间的碰撞检测……143
7.5 小结……144
第 8 章 游戏中的人工智能……145
8.1 让怪物聪明起来——Android 中的路径搜索……145
8.1.1 路径搜索示例基本框架的搭建……145
8.1.2 路径搜索示例的控制面板实现……147
8.1.3 路径搜索示例 GameView 的实现……151
8.1.4 深度优先路径搜索 DFS……154
8.1.5 广度优先路径搜索 BFS……156
8.1.6 路径搜索算法——Dijkstra……159
8.1.7 用 A*算法优化搜索……162
8.2 有限状态机……164
8.2.1 何为有限状态机……164
8.2.2 有限状态机的简单实现……165
8.2.3 有限状态机的 OO 实现……170
8.3 小结……172
第 9 章 游戏开发小秘技……173
9.1 地图编辑器与关卡设计……173
9.1.1 关卡地图的重要性……173
9.1.2 图片分割界面的实现……175
9.1.3 地图设计界面的实现……179
9.2 游戏中的模糊逻辑……185
9.2.1 模糊的才是真实的……185
9.2.2 如何在 Android 中将游戏模糊化……186
9.3 游戏的基本优化技巧……188
9.3.1 代码上的小艺术……188
9.3.2 Android 中的查找表技术……190
9.3.3 游戏的感觉和性能问题……192
9.4 小结……193
第 10 章 游戏的心脏——物理引擎……194
10.1 物理引擎很重要……194
10.1.1 什么是物理引擎……194
10.1.2 常见的物理引擎……194
10.2 2D 的王者 JBox2D……197
10.2.1 基本的物理学概念……197
10.2.2 JBox2D 中常用类的介绍……199
10.3 球体撞击木块金字塔案例……203
10.3.1 案例运行效果……204
10.3.2 案例的基本框架结构……204
10.3.3 常量类——Constant……205
10.3.4 抽象类——MyBody……206
10.3.5 圆形刚体类——MyCircleColor……206
10.3.6 生成刚体形状的工具类——Box2DUtil……207
10.3.7 颜色工具类——ColorUtil……208

10.3.8 主控制类——MyBox2dActivity ……209
10.3.9 显示界面类——GameView ……211
10.3.10 绘制线程类——DrawThread ……212
10.4 简易打砖块案例 ……213
10.4.1 一般碰撞版 ……213
10.4.2 碰撞后消失版 ……215
10.5 旋转关节跷跷板案例 ……220
10.5.1 旋转关节介绍 ……220
10.5.2 多边形刚体类MyPolygonColor的开发 ……220
10.5.3 生成刚体形状的工具类——Box2DUtil ……221
10.5.4 将场景中的刚体摆放到位 ……222
10.5.5 增加旋转关节 ……223
10.6 旋转关节链条摆案例 ……225
10.6.1 案例运行效果 ……225
10.6.2 案例的机械结构 ……225
10.6.3 主控制类——MyBox2dActivity ……226
10.7 组合机械结构案例 ……227
10.7.1 案例的运行效果 ……227
10.7.2 整体场景的机械结构 ……228
10.7.3 案例的基本框架结构 ……229
10.7.4 圆形刚体类——MyCircleColor ……229
10.7.5 生成刚体形状的工具类——Box2DUtil ……230
10.7.6 主控制类——MyBox2dActivity ……230
10.7.7 游戏界面类——GameView ……232
10.7.8 绘制线程类——DrawThread ……233
10.8 小结 ……233

第 11 章 OpenGL ES 应用开发基础 ……234
11.1 OpenGL ES 概述及 3D 基本图形绘制 ……234
11.1.1 OpenGL 及 OpenGL ES 简介 ……234
11.1.2 3D 基本知识 ……235
11.1.3 使用索引的不同绘制方式 ……236
11.1.4 用索引法绘制三角形的案例 ……238
11.1.5 不使用索引数据绘制 ……244
11.2 正交投影和透视投影 ……245
11.2.1 正交投影 ……246
11.2.2 透视投影 ……246
11.2.3 两种投影方式的原理及视口 ……247
11.2.4 两种投影方式的案例 ……248
11.3 光照与材质 ……250
11.3.1 光照的 3 种组成元素 ……250
11.3.2 定向光与定位光 ……252
11.3.3 法向量 ……253
11.3.4 材质 ……254
11.3.5 两种光源的案例 ……255
11.4 纹理及纹理映射 ……257
11.4.1 纹理映射基本原理 ……257
11.4.2 使用纹理映射的案例 ……258
11.4.3 几种纹理拉伸方式 ……261
11.4.4 几种纹理过滤方式 ……262
11.5 摄像机和雾特效 ……263
11.5.1 摄像机的设置 ……264
11.5.2 设置合理的视角 ……264
11.5.3 雾特效的开发 ……266
11.6 典型几何体的开发 ……267
11.6.1 圆柱 ……268
11.6.2 圆锥 ……273
11.6.3 球 ……276
11.6.4 椭圆体 ……278
11.6.5 胶囊体 ……281

11.6.6 几何体大集合……284
11.7 小结……286

第二篇 Android 游戏开发实战综合案例

第 12 章 滚屏动作游戏——太空保卫战……288
12.1 游戏的背景及功能概述……288
12.1.1 背景概述……288
12.1.2 功能简介……288
12.2 游戏的策划及准备工作……290
12.2.1 游戏的策划……290
12.2.2 Android 平台下游戏的准备工作……291
12.3 游戏的架构……292
12.3.1 各个类的简要介绍……292
12.3.2 游戏的框架简介……293
12.4 辅助界面相关类的实现……294
12.4.1 主控制类——PlaneActivity 的实现……294
12.4.2 欢迎界面 WelcomeView 类……296
12.4.3 其他辅助界面的介绍……300
12.5 游戏界面的框架设计……303
12.6 游戏实体相关类的实现……306
12.6.1 飞机 Plane 类的实现……306
12.6.2 敌方飞机 EnemyPlane 类的实现……309
12.6.3 子弹 Bullet 类的实现……311
12.6.4 其他相关类的实现……312
12.7 游戏界面的完善……314
12.7.1 地图类 Maps 的实现……314
12.7.2 背景滚动类 GameViewBackGroundThread 的实现……316
12.7.3 物体移动线程 MoveThread 的实现……317
12.7.4 键盘监听线程 KeyThread 的实现……319
12.7.5 图片初始化方法 initBitmap 的实现……320
12.7.6 绘制方法 onDraw 的实现……322
12.8 游戏的优化与改进……324
第 13 章 棋牌游戏——中国象棋人机对弈……325
13.1 游戏的背景及功能概述……325
13.1.1 背景概述……325
13.1.2 功能介绍……325
13.2 游戏的策划及准备工作……327
13.2.1 游戏的策划……327
13.2.2 Android 平台下游戏的准备工作……327
13.3 游戏的架构……329
13.3.1 各个类简要介绍……329
13.3.2 游戏框架简介……330
13.4 主控制类——Chess_DJB_Activity……331
13.5 辅助界面相关类……334
13.6 游戏界面相关类……336
13.6.1 游戏界面绘制类 GameView……336
13.6.2 游戏界面常量类 ViewConstant……353
13.7 走法引擎相关类……354
13.7.1 常量类 Constant……354
13.7.2 工具类 Chess_LoadUtil……359
13.7.3 走法引擎主类 LoadUtil……360
13.7.4 走法的排序规则类 MyComparator……371
13.7.5 记录走棋步骤类 StackplayChess……371
13.8 游戏的优化与改进……371
第 14 章 物理传感器游戏——小球快跑……372
14.1 游戏背景及功能概述……372
14.1.1 背景概述……372

14.1.2 功能简介……372
14.2 游戏的策划及准备工作……374
14.2.1 游戏的策划……374
14.2.2 小球快跑游戏开发的准备工作……375
14.3 游戏的架构……375
14.3.1 游戏的总体架构……376
14.3.2 游戏的类结构……376
14.4 主控制类——DriftBall 类的开发……378
14.4.1 DriftBall 类的代码框架…378
14.4.2 DriftBall 类的主要成员方法的实现……379
14.5 游戏主菜单的开发……381
14.5.1 欢迎界面类——Welcome View 类的代码框架……381
14.5.2 WelcomeView 类主要成员方法……382
14.5.3 WelcomeThread 类的开发……383
14.5.4 菜单界面的用户交互事件处理……384
14.6 游戏界面模块的开发……385
14.6.1 游戏界面类——GameView 的成员变量……385
14.6.2 GameView 的成员方法简介……387
14.6.3 游戏界面绘制线程——GameThread 类的代码框架……387
14.6.4 GameMenuThread 类的开发……388
14.6.5 用户交互事件处理……389
14.7 游戏中各个图层的开发……391
14.7.1 地图图层的开发……391
14.7.2 其他图层的开发与实现…392
14.8 游戏后台逻辑的开发……395
14.8.1 小球的运动控制……395
14.8.2 小球的碰撞检测……396
14.8.3 大炮相关类 Cannon 和 Missile 的开发……400
14.9 传感器计算模块的开发……403
14.9.1 手机姿态变化监听器——BallListener 类的开发……403
14.9.2 工具类——RotateUtil 类的代码框架……404
14.9.3 工具类——RotateUtil 类的开发……406
14.10 游戏的优化与改进……408
第 15 章 塔防游戏——精灵塔防……410
15.1 游戏的背景及功能概述……410
15.1.1 背景概述……410
15.1.2 功能介绍……410
15.2 游戏的策划及准备工作……413
15.2.1 游戏的策划……413
15.2.2 Android 平台下游戏开发的准备工作……414
15.3 游戏的架构……415
15.3.1 各个类的简要介绍……416
15.3.2 游戏框架简介……417
15.4 主控制类 TafangGameActivity 和数据库类 DBUtil……418
15.4.1 主控制类——TafangGame Activity 主要框架……418
15.4.2 主控制类——TafangGame Activity 中部分数据库的实现……421
15.4.3 主控制类——TafangGame Activity 中对话框的开发……423
15.4.4 数据库类 DBUtil 的开发……427
11.5 界面相关类……428
15.5.1 欢迎界面 WelcomeView 类的介绍……428

15.5.2 主界面 MainMenuSurfaceView 类的介绍……430
15.5.3 音效设置界面 MusicSurfaceView 类的介绍……434
15.5.4 游戏结束界面 GameOverView 类和帮助界面 HelpView 类的介绍……435
15.5.5 积分榜界面 HighJifenSurfaceView 类的介绍……435
15.6 游戏界面 GameView 及相关类……438
15.6.1 精灵怪物 Target 类……439
15.6.2 箭塔 SingleJianta 类……443
15.6.3 箭 Shell 类……444
15.6.4 精灵怪物的出击现场——TargetNumThread……447
15.6.5 箭塔控制发射线程——ShellNumThread……449
15.6.6 游戏地图矩阵模拟化……451
15.6.7 摆放箭塔守护城池……452
15.6.8 游戏中的 2.5D 效果……455
15.6.9 弹指间怪物灰飞烟灭——游戏水晶……455
15.7 游戏的优化和改进……459
第 16 章 策略游戏——回到战国……460
16.1 游戏的背景及功能概述……460
16.1.1 背景概述……460
16.1.2 功能简介……460
16.2 游戏的策划及准备工作……465
16.2.1 游戏的策划……465
16.2.2 Android 平台下游戏的准备工作……465
16.3 游戏的架构……467
16.3.1 游戏的模块架构……467
16.3.2 游戏各个类的简要介绍……468
16.4 地图设计器的开发……470
16.4.1 底层地图设计器的开发……470
16.4.2 上层地图设计器的开发……474
16.5 Activity 和游戏工具类的开发……475
16.5.1 主控制类——HDZGActivity 的介绍……475
16.5.2 公式封装类——GameFormula 的介绍……478
16.5.3 常量工具类 ConstantUtil 的介绍……479
16.6 数据存取模块的开发……481
16.6.1 城池信息以及地图层信息的封装类……481
16.6.2 数据存取相关类的介绍……485
16.7 英雄角色模块的开发……488
16.7.1 Hero 类的代码框架……488
16.7.2 英雄运动线程——HeroGoThread 类的开发……491
16.7.3 辅助线程——HeroBackDataThread 类的开发……493
16.8 表示层界面模块的开发……493
16.8.1 滚屏类——ScreenRollView 类的开发……494
16.8.2 滚屏线程——ScreenRollThread 的开发……495
16.8.3 游戏界面 GameView 的框架介绍……496
16.8.4 游戏界面绘制方法 onDraw 的介绍……498
16.8.5 游戏界面屏幕监听方法 onTouch 的介绍……500
16.8.6 游戏界面后台线程 GameViewThread 的介绍……502
16.9 管理面板模块的开发……503
16.9.1 人物属性面板类 ManPanelView 的开发……503
16.9.2 城池管理面板类 CityManageView 的开发……508

16.10 地图中可遇实体模块的开发……511
16.10.1 绘制类——MyDrawable的开发……511
16.10.2 抽象类——MyMeetableDrawable的开发……513
16.10.3 森林类——ForestDrawable的开发……514
16.10.4 可遇实体对象的调用流程……516
16.11 英雄技能模块的开发……518
16.11.1 技能抽象类——Skill的开发……518
16.11.2 伐木技能类——Lumber Skill的开发……519
16.11.3 随心步技能类——SuiXinBuSkill的开发……520
16.12 游戏提示模块的开发……520
16.12.1 提示模块抽象类——GameAlert的开发……520
16.12.2 点击确定按钮显示的信息类——PlainAlert的开发……521
16.12.3 显示粮草危机信息类——FoodAlert的开发……522
16.12.4 辅助线程HeroBackData Thread中对FoodAlert的调用……524
16.13 游戏的优化与改进……525
第17章 体育游戏——2D迷你桌球……527
17.1 2D桌球的背景及功能概述……527
17.1.1 背景概述……527
17.1.2 功能简介……527
17.2 游戏的策划及准备工作……530
17.2.1 游戏的策划……530
17.2.2 Android平台下游戏的准备工作……530
17.3 游戏的架构……531
17.3.1 游戏的框架简介……531
17.3.2 各个类的简要介绍……532
17.4 公共类的实现……534
17.4.1 主控制类——GameActivity的代码框架……534
17.4.2 GameActivity类主要成员变量及方法的实现……536
17.4.3 常量类——Constant的实现……540
17.5 辅助界面相关类的实现……542
17.5.1 欢迎动画界面WelcomeView类的实现……542
17.5.2 主菜单界面MainMenuView的代码框架……544
17.5.3 主界面类——MainMenu View部分成员方法的实现……545
17.5.4 主界面动画线程类View DrawThread的实现……548
17.5.5 排行榜界面HighScore View的代码框架……549
17.5.6 HighScoreView类的部分方法的实现……550
17.6 游戏界面相关类的实现……552
17.6.1 游戏界面GameView类的代码框架……552
17.6.2 GameView类部分成员方法的实现……553
17.6.3 键盘监听线程KeyThread类的实现……557
17.6.4 球运动的线程BallGoThread类的实现……558
17.7 情景相关类的实现……559
17.7.1 球台Table类的实现……559
17.7.2 桌球Ball类的代码框架……561
17.7.3 Ball类部分成员方法的实现……563
17.7.4 球杆Cue类的实现……566

17.8 自定义控件及工具类的实现……568
17.8.1 球与球碰撞检测的工具类 CollisionUtil 的实现……568
17.8.2 定时器 Timer 类的实现…571
17.8.3 主菜单按钮 MainMenu Button 类的实现……572
17.8.4 获取日期的工具 DateUtil 类的实现……573
17.9 游戏的优化与改进……574
第 18 章 益智游戏——3D 版推箱子……575
18.1 3D 版推箱子的背景及功能概述…575
18.1.1 背景概述……575
18.1.2 功能简介……575
18.2 游戏的策划及准备工作……579
18.2.1 游戏的策划……579
18.2.2 Android 平台下游戏开发的准备工作……579
18.3 游戏的架构……580
18.3.1 游戏的框架简介……580
18.3.2 各个类的简要介绍……581
18.4 公共类的实现……582
18.4.1 主控制类——MyActivity 的代码框架……582
18.4.2 MyActivity 类成员方法的实现……584
18.4.3 常量类 Constant 的实现…585
18.5 辅助界面相关类的实现……587
18.5.1 欢迎动画界面 WelcomeView 类的实现……587
18.5.2 主菜单界面 MainView 的实现……589
18.6 游戏界面相关类……590
18.6.1 游戏界面 MySurfaceView 类的设计与实现……590
18.6.2 键盘监听线程 KeyThread……599
18.7 自定义控件及工具类……600
18.7.1 缩放图片的方法 PicLoadUtil……600
18.7.2 主菜单上的按钮类 MainMenuButton……601
18.8 情景相关类的实现……602
18.8.1 机器人类的代码框架……602
18.8.2 机器人类成员方法的实现……603
18.8.3 机器人组类 RobotGroup 类的实现……605
18.8.4 机器人组类成员方法的实现……606
18.8.5 圆面 Circle 类的实现……607
18.8.6 矩形纹理类 TextureRect 的实现……609
18.8.7 纹理矩形组类 TextureRect Group 的实现……610
18.8.8 墙类 Wall 的实现……611
18.8.9 墙 Wall 类成员方法的实现……612
18.8.10 箱子 Cube 类的实现……616
18.8.11 箱子组 CubeGroup 类的实现……618
18.8.12 箱子移动 CubeGo 类的代码框架……619
18.8.13 箱子移动 CubeGo 类的成员方法的实现……620
18.8.14 地板类 Floor 的实现……621
18.9 推箱子地图设计器的开发……623
18.10 游戏的优化与改进……624
第 19 章 物理引擎游戏——盛怒的老鼠……625
19.1 游戏背景及功能概述……625
19.1.1 游戏开发背景……625
19.1.2 游戏功能概述……625
19.2 游戏策划及准备工作……628
19.2.1 游戏策划……628
19.2.2 游戏开发的准备工作……628

19.3 游戏的框架……630
19.3.1 游戏主要用到的技术……631
19.3.2 游戏各个类的介绍……631
19.3.3 游戏的基本框架……633
19.4 公共类……634
19.4.1 主控制类——MyBox2dActivity……634
19.4.2 常量类 Constant……635
19.5 主界面的设计与实现……644
19.5.1 鼠头和猫头类 Taj……644
19.5.2 鼠头和猫头控制线程——TJThread……646
19.5.3 主界面 MainMenuView……646
19.5.4 刷帧线程 MainMenuDrawThread……653
19.6 工具类……654
19.6.1 加载及缩放图片的工具类 PicLoadUtil……654
19.6.2 生成刚体工具类 Box2DUtil……655
19.6.3 声音工具类 SoundUtil……656
19.7 刚体相关类……657
19.7.1 多边形类 MyPolygonImg……657
19.7.2 猫头类 BodyCat……659
19.7.3 冰块类 BodyIce……660
19.7.4 木条类 BodyWood……661
19.7.5 刚体类型枚举类 BodyType……661
19.7.6 刚体查询工具类 BodySearchUtil……662
19.8 游戏界面相关类……663
19.8.1 皮筋类 Pijin……663
19.8.2 记录分数类 Score……664
19.8.3 刷帧线程 DrawThread……665
19.8.4 游戏界面类 GameView……668
19.9 游戏优化与改进……673

第一篇

Android 游戏开发核心技术

第 1 章　Android 平台简介

第 2 章　Android 游戏开发之前台渲染

第 3 章　Android 游戏开发之交互式通信

第 4 章　Android 游戏开发之数据存储和传感器

第 5 章　Android 游戏开发之网络编程

第 6 章　不一样的游戏，不一样的体验

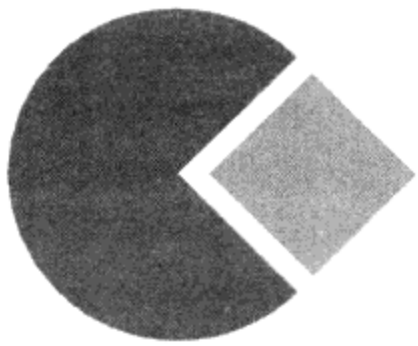
第 7 章　游戏背后的数学与物理

第 8 章　游戏中的人工智能

第 9 章　游戏开发小秘技

第 10 章　游戏的心脏——物理引擎

第 11 章　OpenGL ES 应用开发基础



第 1 章　Android 平台简介

Android 是 Google 公司于 2007 年 11 月 5 日发布的基于 Linux 内核的移动平台，该平台由操作系统、中间件、用户界面和应用软件组成，是一个真正开放的移动开发平台。

本章将介绍 Android 系统的起源、特点、应用程序框架以及开发环境的搭建，让读者对 Android 平台有个初步的了解，之后将开发第一个 Android 程序 Hello Android，并通过对该程序的简单分析，带领读者步入 Android 开发的大门。

1.1 Android 的来龙去脉

Android 的创始人 Andy Rubin 是硅谷著名的“极客”，他离开 Danger 移动计算公司后不久便创立了 Android 公司，并开发了 Android 平台，他一直希望将 Android 平台打造成完全开放的移动终端平台。之后 Android 公司被 Google 公司看中并将其收购。不久，号称全球最大的搜索服务商 Google 大举进军移动通信市场，并推出自主品牌的移动终端产品。

2007 年 11 月初，Google 正式宣布与其他 33 家手机厂商、软硬件供应商、手机芯片供应商、移动运营商联合组成开放手机联盟（Open Handset Alliance），并发布名为 Android 的开放手机软件平台，希望建立标准化、开放式的移动电话软件平台，在移动行业内形成一个开放式的生态系统。

1.2 掀起 Android 的盖头来

自从 Android 发布以来，越来越多的人关注 Android 的发展，越来越多的开发人员在 Android 系统平台上开发应用，那是什么使 Android 备受青睐，什么使 Android 在众多移动平台中脱颖而出呢？

1.2.1　选择 Android 的理由

与其他手机的操作系统相比，Android 具有如下优点：

- 开放性；
- 平等性；
- 无界性；
- 方便性；
- 硬件的丰富性。

下面将对以上各个优点进行简单介绍。

1. 开放性

提到 Android 的优势，首先想到的一定是其真正的开放，其开放性包含底层的操作系统以及上层的应用程序等。Google 与开放手机联盟合作开发 Android 的目的就是建立标准化、开放式的移动软件平台，在移动产业内形成一个开放式的生态系统。

Android 的开放性也同样会使大量的程序开发人员投入到 Android 程序的开发中，这将为 Android 平台带来大量新的应用。

2. 平等性

在 Android 的系统上，所有的应用程序完全平等，系统默认自带的程序与自己开发的程序没有任何区别，程序开发人员可以开发个人喜爱的应用程序来替代系统的程序，构建个性化的 Android 手机系统，这些功能在其他的手机平台是没有的。

在开发之初，Android 平台就被设计成由一系列应用程序组成的平台，所有的应用程序都运行在一个虚拟机上面。该虚拟机提供了系列应用程序之间和硬件资源通信的 API。

3. 无界性

Android 平台的无界性表现在应用程序之间的无界，开发人员可以很轻松地将自己开发的程序与其他应用程序进行交互，比如应用程序需要播放声音的模块，而正好你的手机中已经有一个成熟的音乐播放器，此时就不需要再重复开发音乐播放功能，只需简单地加上几行代码即可将成熟的音乐播放功能添加到自己的程序中。

4. 方便性

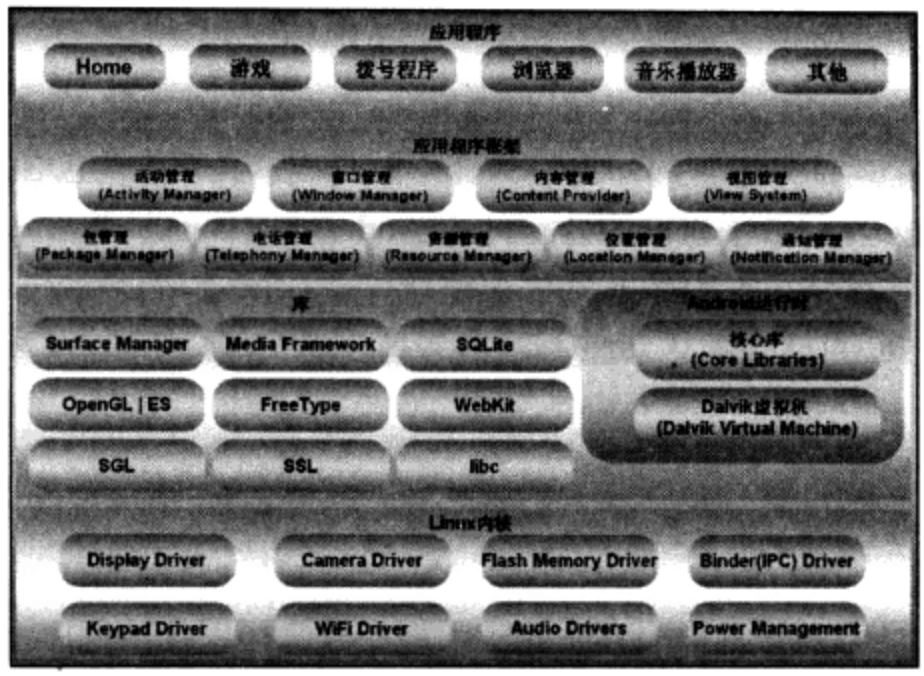
在 Android 平台中开发应用程序是非常方便的，如果对 Android 平台比较熟悉，想开发一个功能全面的应用程序并不是什么难事。Android 平台为开发人员提供了大量的实用库及方便的工具，同时也将 Google Map 等强大的功能集成了进来，只需简单的几行调用代码即可将强大的地图功能添加到自己的程序中。

5. 硬件的丰富性

由于平台的开放，众多的硬件制造商推出了各种各样的产品，但这些产品功能上的差异并不影响数据的同步与软件的兼容，例如，原来在诺基亚手机上的应用程序，可以很轻松地被移植到摩托罗拉手机上使用，且联系人、短信息等资料更是可以方便地转移。

1.2.2　Android 的应用程序框架

从软件分层的角度来说，Android 平台由应用程序、应用程序框架、Android 运行时库层以及 Linux 内核共 4 部分构成，本节将分别介绍各层的功能，分层结构如图 1-1 所示。



▲图 1-1　Android 平台架构图

1. 应用程序层

本层的所有应用程序都是用 Java 编写的，一般情况下，很多应用程序都是在同一系列的核心应用程序包中一起发布的，主要有拨号程序、浏览器、音乐播放器、通信录等。该层的程序是完全平等的，开发人员可以任意将 Android 自带的程序替换成自己的应用程序。

2. 应用程序框架层

对于开发人员来说，接触最多的就是应用程序框架层。该应用程序的框架设计简化了组件的重用，其中任何一个应用程序都可以发布自身的功能供其他应用程序调用，这也使用户可以很方便地替换程序的组件而不影响其他模块的使用。当然，这种替换需要遵循框架的安全性限制。

该层主要包含以下 9 部分，如图 1-2 所示。

▲图 1-2　应用程序框架

- 活动管理（Activity Manager）。用来管理程序的生命周期，以及提供最常用的导航回退功能。
- 窗口管理（Window Manager）。用来管理所有的应用程序窗口。
- 内容管理（Content Providers）。通过内容管理，可以使一个应用程序访问另一个应用程序的数据，或者共享数据。
- 视图管理（View System）。用来构建应用程序的基本组件，包括列表、网格、按钮、文本框，甚至是可嵌入的 Web 浏览器。
- 包管理（Package Manager）：用来管理 Android 系统内的程序。

- 电话管理（Telephony Manager）。所有的移动设备的功能统一归电话管理器管理。
- 资源管理（Resource Manager）。资源管理器可以为应用程序提供所需要的资源，包括图片、文本、声音、本地字符串，甚至是布局文件。
- 位置管理（Location Manager）。该管理器是用来提供位置服务的，如 GPRS 定位等。
- 通知管理（Notification Manager）。主要对手机顶部状态栏的管理，开发人员在开发 Android 程序时会经常使用，如来短信提示、电量低提示，还有后台运行程序的提示等。

3. Android 运行时库

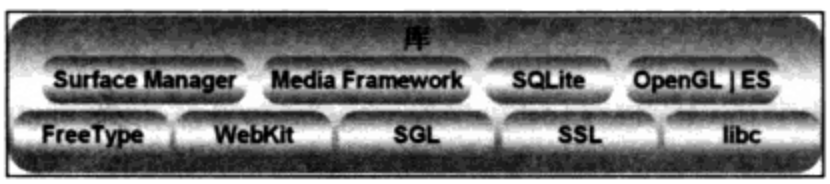
该层包含两部分，程序库及 Android 运行时库。

程序库为一些 C/C++库，这些库能够被 Android 系统中不同的应用程序调用，并通过应用程序框架为开发者提供服务。而 Android 运行时库包含了 Java 编程语言核心库的大部分功能，提供了程序运行时所需调用的功能函数。

程序库主要包含的功能库如图 1-3 所示。

- libc。是一个从 BSD 继承来的标准 C 系统函数库，专门针对移动设备优化过的。
- Media Framework。基于 PacketVideo 公司的 OpenCORE。支持多种常用音频、视频格式回放和录制，并支持多种图像文件，如 MPEG-4、H.264、MP3、AAC、AMR、JPG、PNG 等。
- Surface Manager。Surface Manager 主要管理多个应用程序同时执行时，各个程序之间的显示与存取，并且为多个应用程序提供了 2D 和 3D 图层无缝的融合。
- SQLite。所有应用程序都可以使用的轻量级关系型数据库引擎。
- WebKit。是一套最新的网页浏览器引擎。同时支持 Android 浏览器和一个可嵌入的 Web 视图。
- OpenGLIES。是基于 OpenGL ES 1.0 API 标准来实现的 3D 绘制函数库。该函数库支持软件和硬件两种加速方式执行。
- FreeType。提供位图（bitmap）和矢量图（vector）两种字体显示。
- SGL。提供了 2D 图形绘制的引擎。

Android 运行时库包括核心库及 Dalivik 虚拟机，如图 1-4 所示。



▲图 1-3　程序库框架

▲图 1-4　Android 运行时库

- 核心库（Core Libraries）。该核心库包括 Java 语言所需要的基本函数以及 Android 的核心库。与标准 Java 不一样的是，系统为每个 Android 的应用程序提供了单独的 Dalvik 虚拟机来执行，即每个应用程序拥有自己单独的线程。
- Dalvik 虚拟机（Dalvik Virtual Machine）。大多数的虚拟机（包括 JVM）都是基于栈的，而 Dalvik 虚拟机则是基于寄存器的，它可以支持已转换为.dex 格式的 Java 应用程序的运行。.dex 格式是专门为 Dalvik 虚拟机设计的，更适合内存和处理器速度有限的系统。

4. Linux 内核

Android 平台中操作系统采用的是 Linux 2.6 内核，其安全性、内存管理、进程管理、网络协议栈和驱动模型等基本依赖于 Linux。对于程序开发人员，该层为软件与硬件之间增加了一层抽象层，使开发过程中不必时时考虑底层硬件的细节。而对于手机开发商而言，对此层进行相应的修改即可将 Android 平台运行到自己的硬件平台之上。

1.3 Android 开发环境的搭建

本节主要讲述如何构建基于 Eclipse 的 Android 开发环境，之后对开发环境进行测试并创建第一个 Android 应用程序 Hello Android。

1.3.1 SDK 的下载及安装

前面已经对 Android 平台进行了简单的介绍，从本节开始，将带领读者逐步搭建自己的开发环境，具体的步骤如下。

（1）登录到 SUN 官方网站 http://java.sun.com/javase/downloads 下载最新的 JDK 安装程序。

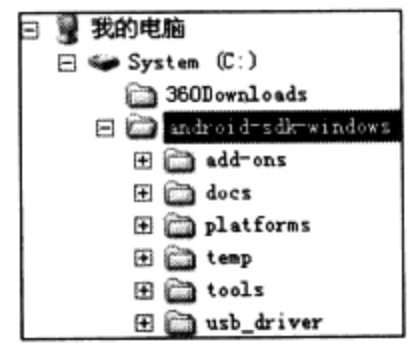
（2）双击刚刚下载的 JDK 安装程序 jdk-6u18-windows-i586.exe，根据提示将 JDK 安装到默认目录。

（3）右键单击我的电脑，依次选择属性/高级/环境变量，在系统变量中新建一个名为 JAVA_HOME，值为“C:\Program Files\Java\jdk1.6.0_18”的环境变量。

（4）再打开 Path 环境变量，在最后加上“C:\Program Files\Java\jdk1.6.0_18\bin;”，单击“确定”按钮即可。

（5）到 http://developer.android.com 页面下载最新的 SDK，本书使用的版本是 2.1，其他版本的安装与配置方法基本相同。

（6）将下载好的 SDK 压缩包解压到 C 盘根目录下，如图 1-5 所示。



▲图 1-5 Android SDK 安装环境

（7）右键单击我的电脑，依次选择属性/高级/环境变量，打开 Path 系统环境变量，在其最后加上 SDK 的解压目录中的 tools 目录“C:\android-sdk-windows\tools”，单击“确定”按钮完成配置，如图 1-6 所示。

1.3.2 Eclipse 集成开发环境的搭建

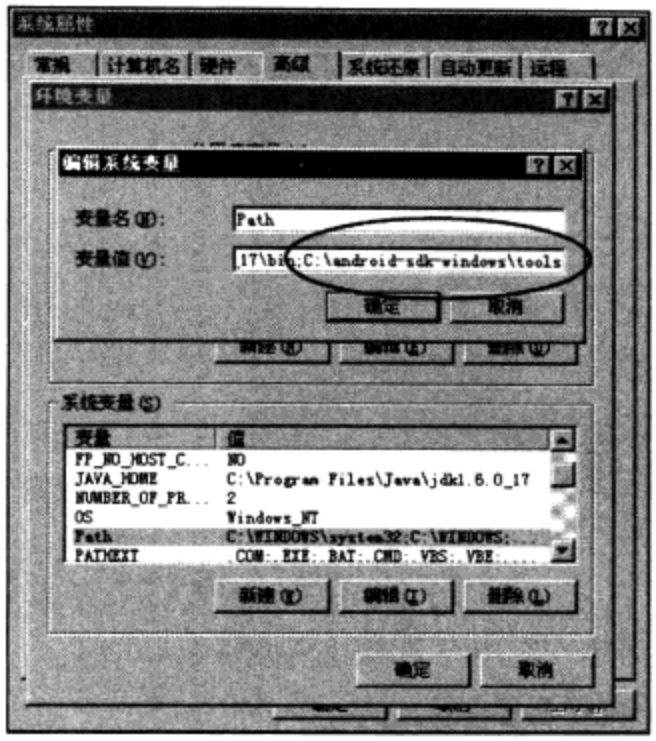
上一小节已经讲解了 SDK 的安装与配置，本小节将带领读者构建基于 Eclipse 的集成开发环境，具体步骤如下。

（1）读者只需登录到 http://www.eclipse.org/downloads/即可下载最新版本的 Eclipse 安装程序，本书采用的版本是 3.5.1。

（2）将下载好的 Eclipse 安装程序解压到 C 盘根目录下，如图 1-7 所示。

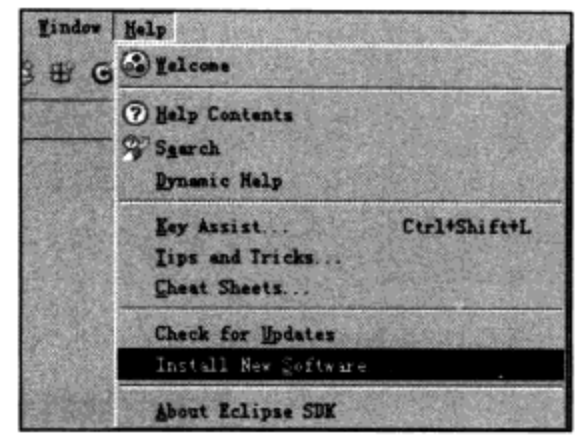
（3）双击 C:\eclipse 目录下的 eclipse.exe 文件，运行 Eclipse，再依次单击菜单 Help/Install New

Software 项，如图 1-8 所示。



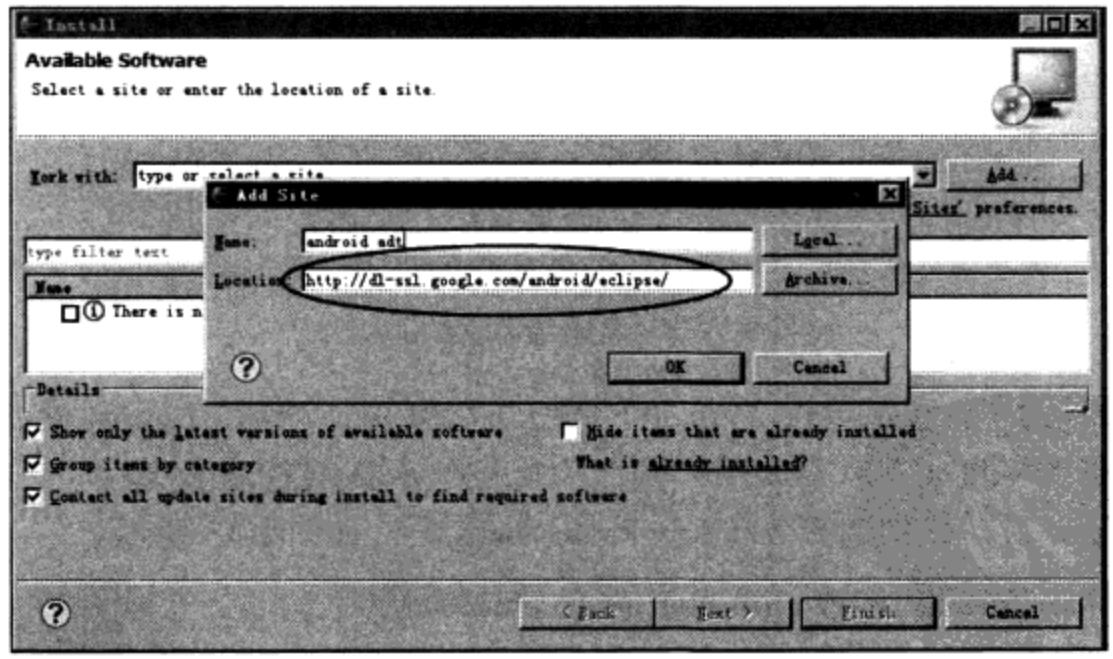
▲图 1-6　SDK 环境变量配置

▲图 1-7　Eclipse 安装目录



▲图 1-8　Eclipse 的软件安装

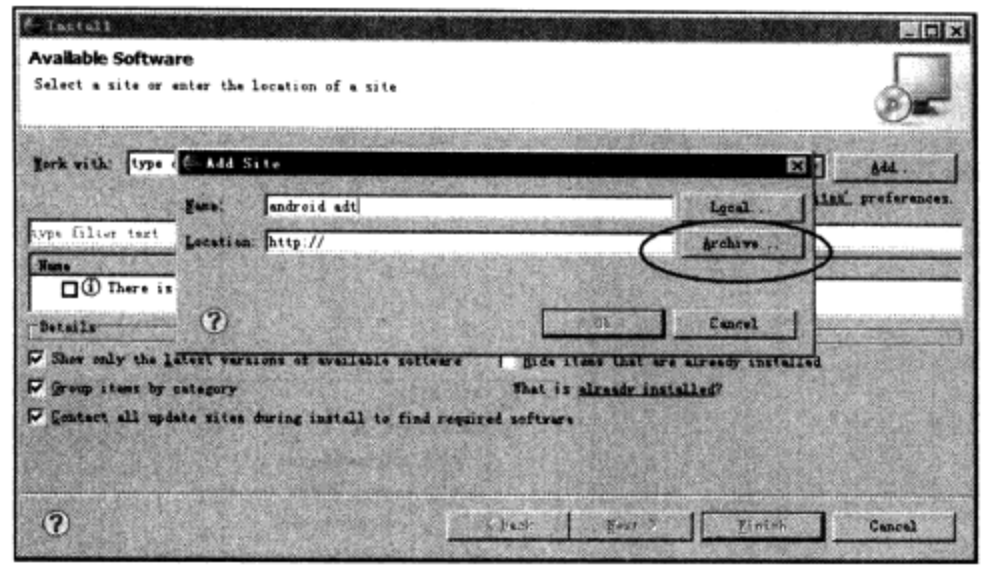
（4）单击弹出的 Install 界面中的“Add”按钮，在 Name 栏中输入任意的名字，并在 Location 栏中输入 ADT 的下载地址 https://dl-ssl.google.com/android/eclipse/，如图 1-9 所示。



▲图 1-9　填写 ADT 地址

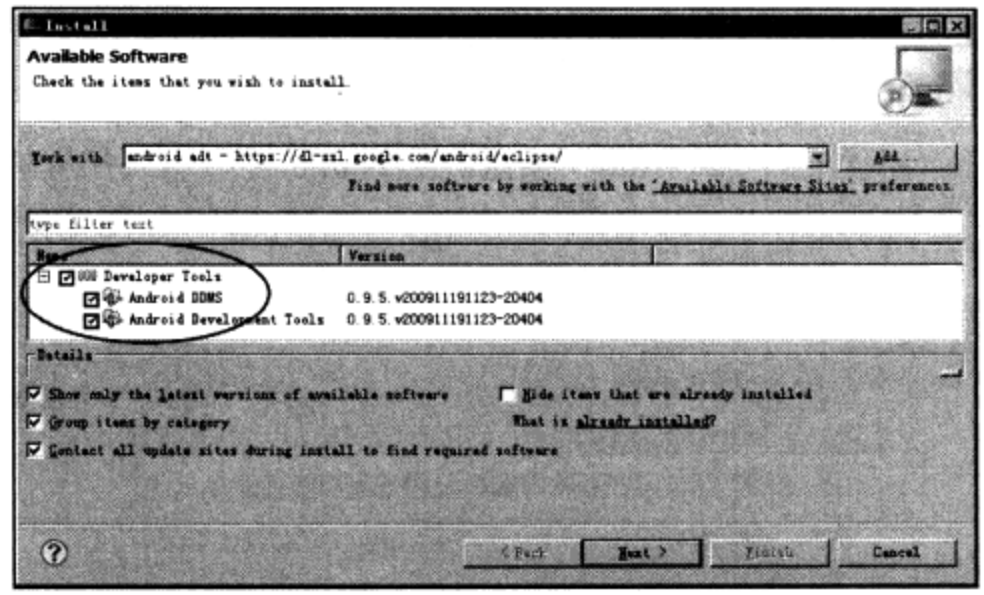
提示

如果输入的地址不能正常下载 ADT，读者可以尝试将 https://dl-ssl.google.com/android/eclipse/改为 http://dl-ssl.google.com/android/eclipse/。如果出现下载的 ADT 与已经安装的 SDK 版本不兼容的情况，可以手动下载 ADT 压缩包，然后单击“Archive”按钮选择手动下载 ADT 压缩文件即可安装，如图 1-10 所示。



▲图 1-10　选择 ADT 压缩文件

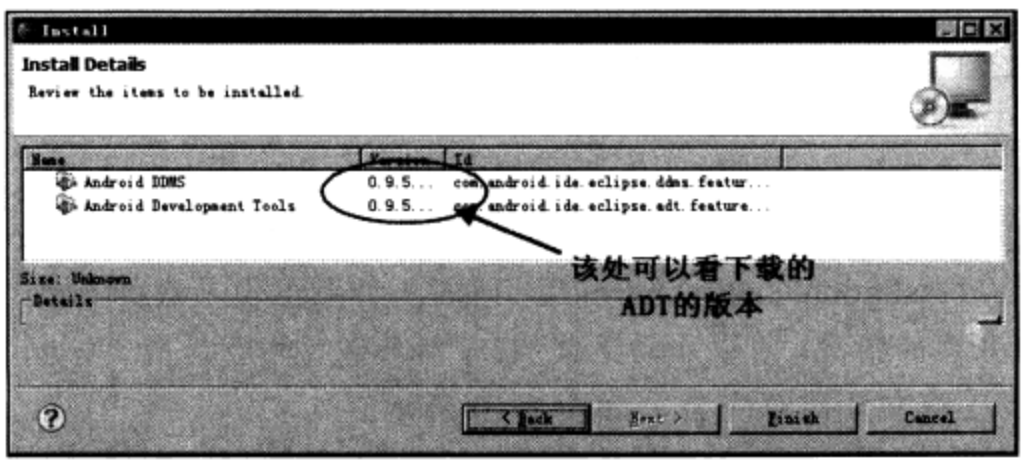
（5）在 Install 界面上选中“Developer Tools”并单击“Next”按钮，如图 1-11 所示。



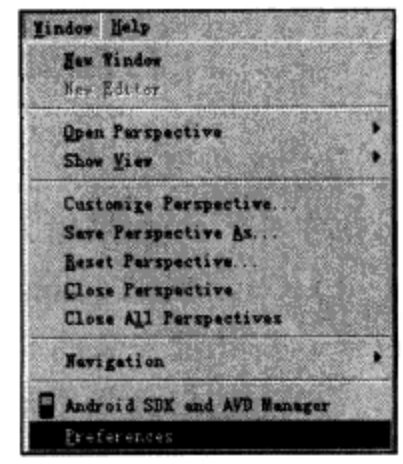
▲图 1-11　选择 Develpoer Tools

（6）在弹出的界面中，直接单击“Finish”按钮，如图 1-12 所示。Eclipse 将弹出提示窗口，单击“Yes”按钮，系统将自动重启 Eclipse，到此为止已经完成 ADT 的安装。

（7）重启 Eclipse 后，依次选择 Window/Preferences 项，如图 1-13 所示。



▲图 1-12　ADT 安装完成界面



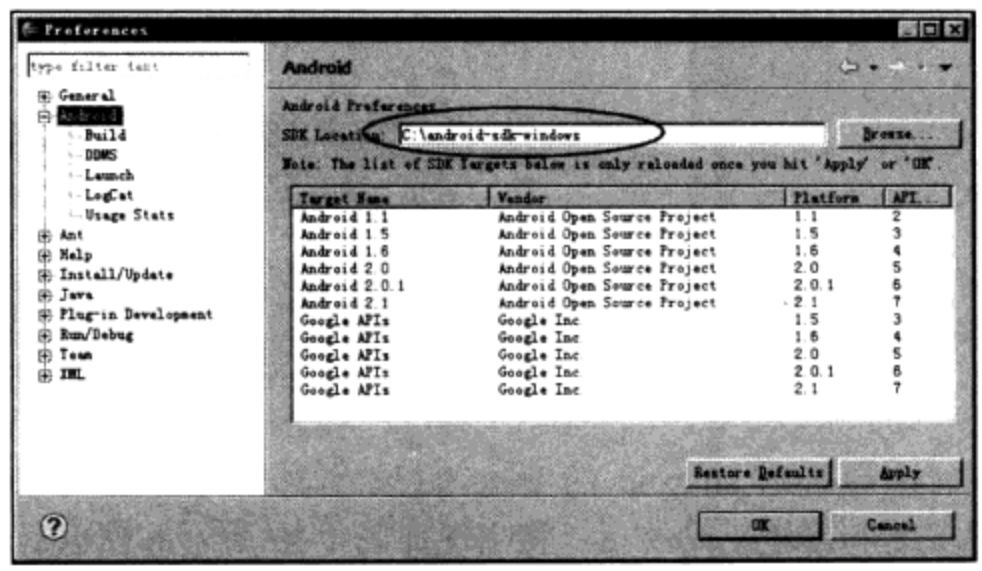
▲图 1-13　选择“Preferences”菜单

（8）在弹出的窗口中选择左侧的“Android”，然后通过单击右侧的“Browse”按钮来选择 SDK 的安装目录，单击“OK”按钮完成配置，如图 1-14 所示。

1.3.3　虚拟设备的创建与模拟器的运行

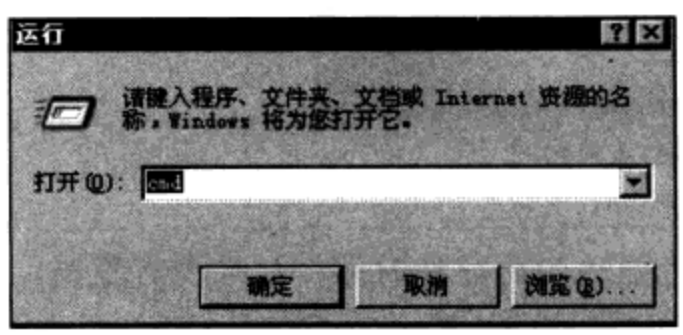
前几小节已经完成了 Eclipse 集成环境的搭建工作，但在运行模拟器前还需要创建 AVD（Android Virtual Device，Android 虚拟设备）。AVD 实质上是用来描述模拟器属性的，有了 AVD 开发人员可以为自己的模拟器进行必要的设置，如屏幕尺寸、内存等。本节将运用之前搭建好的 Eclipse 环境创建虚拟设备并运行模拟器，具体步骤如下。

（1）在系统的开始菜单中单击“运行”按钮，在弹出的窗口中输入“cmd”进入命令行，如图 1-15 所示。

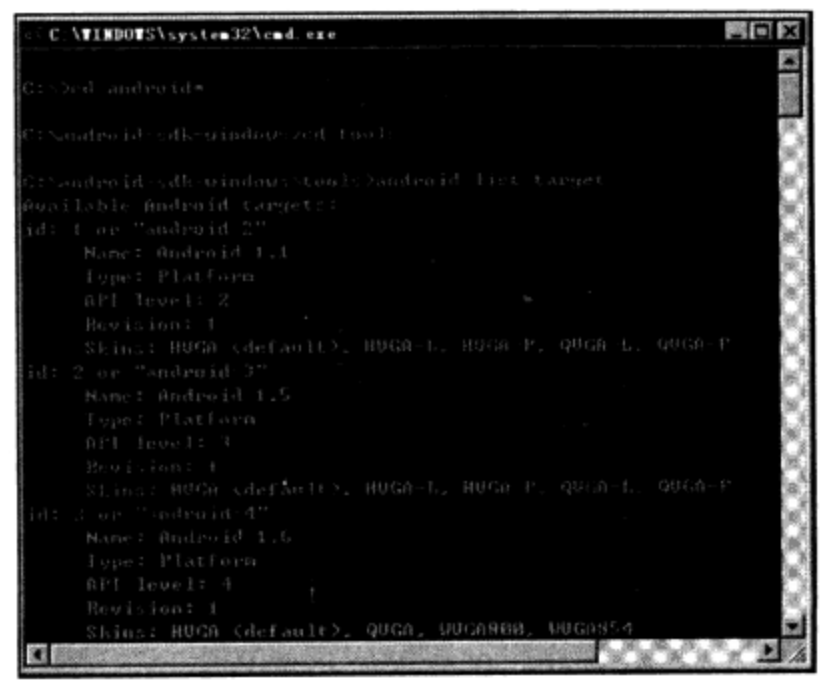


▲图 1-14　Preferences 配置

（2）通过 cd 命令导航到 android 的安装目录下的 tools 下，并通过 android list target 命令来显示当前可用的目标机，如图 1-16 所示。



▲图 1-15　启动 cmd 命令行

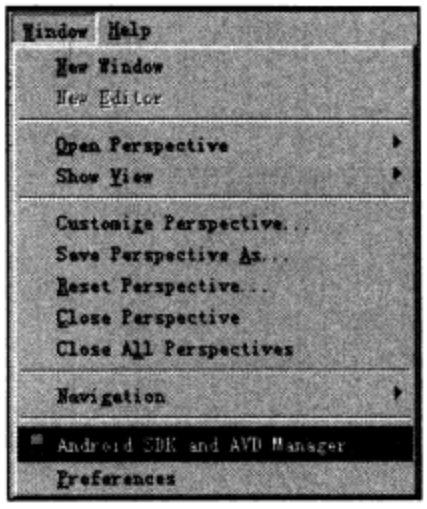


▲图 1-16　当前可用的目标机

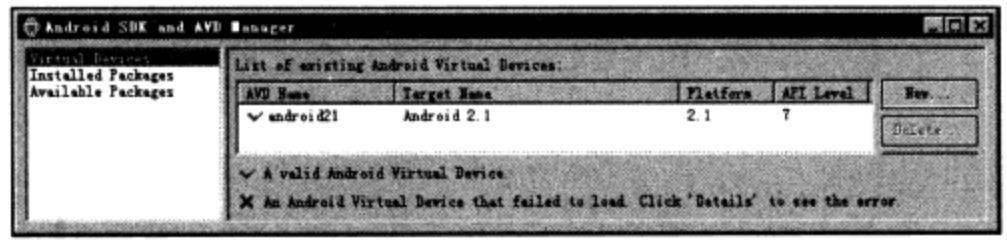
（3）然后通过 android create avd--name android21--target 6 命令创建一个名为 android21、目标机 id 为 6 的 AVD，创建的 AVD 默认存放在我的文档下，而 Android 的 AVD 并不支持中文目录，可以通过配置 ANDROID_SDK_HOME 环境变量来指定所创建的 AVD 的存放位置。

提示

除了使用命令行方式来创建 AVD 外，还可以使用 Eclipse 自带的 AVD Manager 来创建。在 Eclipse 中依次选择“Window/Android SDK and AVD Manager”打开 AVD 管理界面，如图 1-17 和图 1-18 所示；在 AVD 管理界面中不仅可以创建 AVD，还可以对已经存在的 AVD 进行管理。

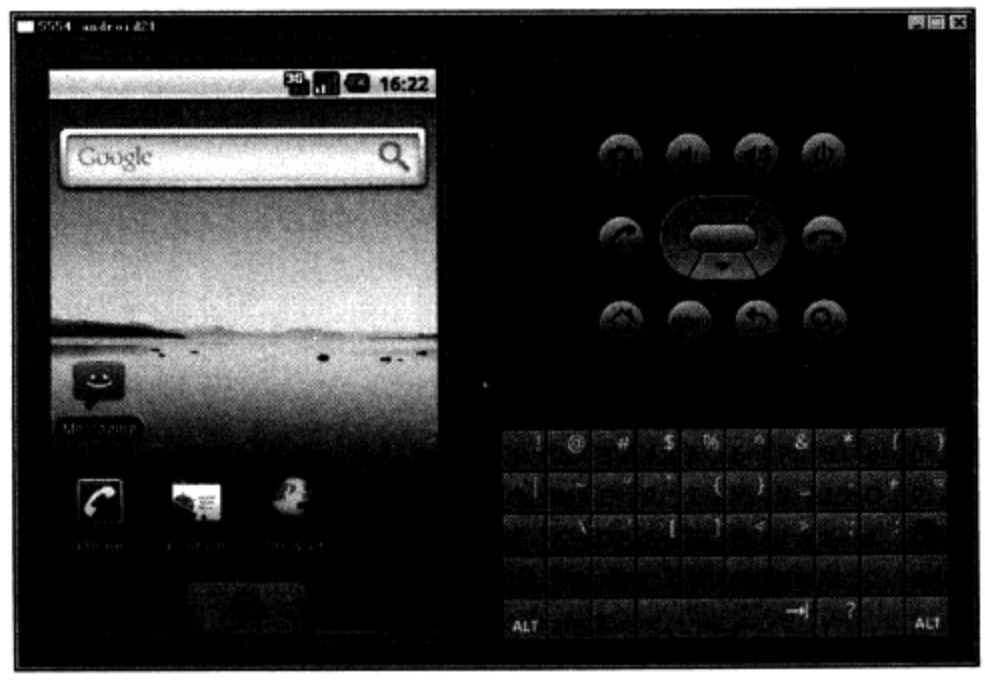


▲图 1-17　打开 AVD 管理器



▲图 1-18　AVD 管理界面

（4）在 cmd 命令行中导航到 android 的 tools 目录后输入 emulator-avd android21 命令启动 Android 模拟器，如果安装和配置正确的话，将出现如图 1-19 所示的模拟器界面。



▲图 1-19　Android2.1 模拟器效果图

提示

运行模拟器时可能会很慢，有时甚至可能达到几分钟时间，如果没有报错，请读者耐心等待。

1.3.4　第一个 Android 程序

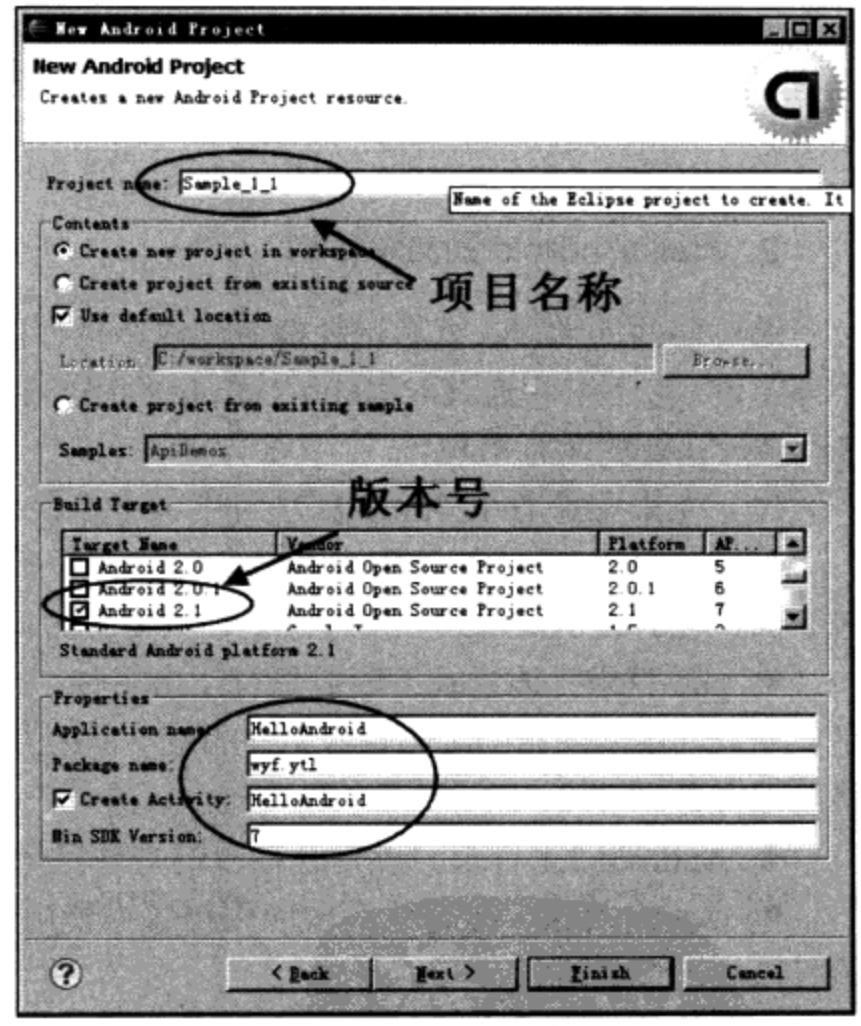
前面已经对 Android 的开发环境和模拟器进行配置，本小节将带领读者构建第一个 Android 程序并对该程序进行简单的讲解。

1. 创建第一个 Android 应用程序

（1）启动 Eclipse，依次选择“File/New/ Other...Eclipse”将弹出创建项目的窗口，如图 1-20 所示。

（2）选择“Android Project”，单击“Next”按钮，将弹出填写项目信息的界面，如图 1-21 所示。

▲图 1-20　创建项目界面

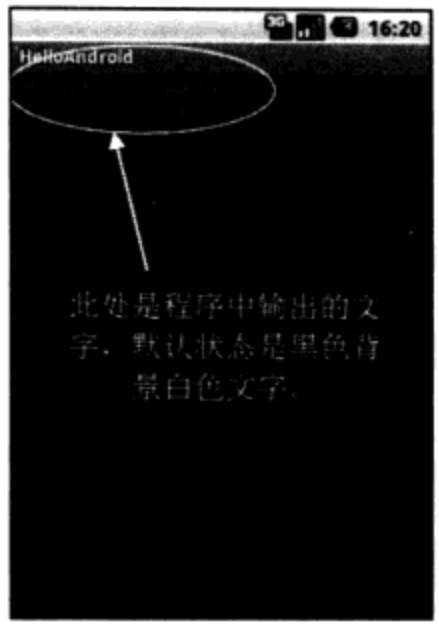


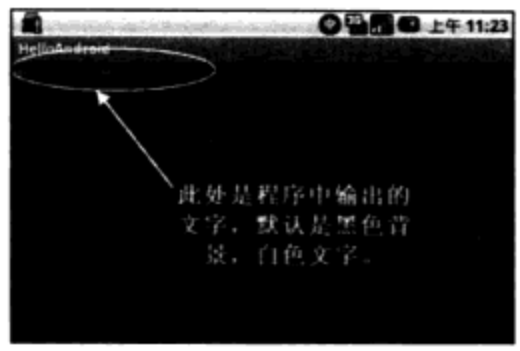
▲图 1-21　填写项目信息

（3）填写如图 1-21 所示的信息后单击“Finish”按钮即可完成项目的创建。创建后可在 Package Explorer 窗口中观察到项目的目录结构，如图 1-22 所示。

（4）在项目名上单击鼠标右键，然后依次选择“Run As/Android Application”即可运行刚才创建的 HelloAndroid 项目，运行效果如图 1-23 所示。

提示　因为有很多程序或者游戏是横屏模式的，所以在程序调试过程中，可能需要将模拟器切换成横屏模式，读者可以通过使用快捷键 Ctrl+F12 来切换模拟器的横竖屏模式，HelloAndroid 程序在横屏模拟器中运行效果如图 1-24 所示。





▲图 1-22　HelloAndroid 项目目录结构▲图 1-23　HelloAndroid 项目竖屏效果图 ▲图 1-24　HelloAndroid 项目横屏效果图

2. HelloAndroid 的简单讲解

通过前面的学习，读者已经能够创建并运行简单的 Android 程序了，但可能对 Android 项目还不够了解，接下来将通过对 HelloAndroid 程序的详细介绍使读者了解 Android 项目的目录结构以及 HelloAndroid 的运行机理。

（1）先来看看 HelloAndroid 项目中各个目录和文件的作用。

- src 目录，src 目录用来存放应用程序中所有的源代码，代码的源文件一般是存放在相应的包下面，在开发 Android 应用程序时，大部分时间都是在编写 src 中的源代码。
- gen 目录，该目录下一般只有一个文件，即 R 文件。该文件是由 ADT 自动产生的，存放的是应用程序中所使用的全部资源文件的 ID，在应用程序开发过程中只是使用 R 文件，一般不需人工修改该文件。
- Android 2.1 目录。该目录存放的是项目所需要的支持.jar 包。
- assets 目录。该目录存放应用程序中使用的外部资源文件，程序中可以通过输入/输出流对该目录中的文件进行读写。
- res 资源目录：该目录下有多个目录，分别用来存放程序中用到的图片、界面布局文件及 XML 格式的描述文件。
- AndroidManifest.xml。该文件是整个程序的系统控制文件，是每个应用程序都不可缺少的。其描述了应用程序有哪些组件、哪些资源、哪些权限等。

（2）AndroidManifest.xml 文件是该项目的系统控制文件，该文件的代码如下。

代码位置：见随书光盘中源代码/第 1 章/Sample_1_1 目录下的 AndroidManifest.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>      <!--XML 的版本以及编码方式-->
2    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3         package="wyf.ytl"
4         android:versionCode="1"
5         android:versionName="1.0">      <!--该标记定义了该项目的使用架构,所在的包以及版本号-->
6       <application android:icon="@drawable/icon" android:label="@string/app_name">
```

```
7                                          <!-- 定义了该项目在手机中的图标以及名称 -->
8           <activity android:name=".HelloAndroid"
9                   android:label="@string/app_name">     <!-- 声明 Activity 组件 -->
10              <intent-filter>
11                  <action android:name="android.intent.action.MAIN" />
12                  <category android:name="android.intent.category.LAUNCHER" />
13              </intent-filter>                    <!-- 声明 Activity 可以接受的 Intent -->
14          </activity>
15      </application>
16      <uses-sdk android:minSdkVersion="7" />   <!-- 声明 SDK 的版本 -->
17  </manifest>
```

- 第 1～5 行定义了程序的版本、编码方式、用到的架构以及该程序所在的包与版本号。
- 第 6 行定义了程序在手机上的显示图标及显示名称。
- 第 8～14 行定义了一个名为 HelloAndroid 的 Activity 以及该 Activity 能够接受的 Intent。

（3）main.xml 是该项目的布局文件，其代码如下。

代码位置：见随书光盘中源代码/第 1 章/Sample_1_1/res/Layout 目录下的 main.xml。

```
1   <?xml version="1.0" encoding="utf-8"?>      <!-- XML 的版本以及编码方式 -->
2   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       android:orientation="vertical"
4       android:layout_width="fill_parent"
5       android:layout_height="fill_parent"
6       >                                       <!--定义了一个线性布局,布局方式是垂直的-->
7   <TextView
8       android:layout_width="fill_parent"
9       android:layout_height="wrap_content"
10      android:text="@string/hello"
11      />                                      <!--向线性布局中添加一个 TextView 控件-->
12  </LinearLayout>
```

- 第 2 行定义了布局方式为 LinearLayout，且左右和上下的填充方式为 fill_parent。
- 第 7 行中向该布局中添加了一个 TextView 控件，其宽度和高度模式分别为 fill_parent、wrap_content，在 TextView 控件显示的内容为 string.xml 中的 hello 的内容。

（4）项目的主类 HelloAndroid.java 的代码如下。

代码位置：见随书光盘中源代码/第 1 章/Sample_1_1/src/wyf/ytl 目录下的 HelloAndroid.java。

```
1   package wyf.ytl;
2   import android.app.Activity;                          //引入相关类
3   import android.os.Bundle;
4   public class HelloAndroid extends Activity {          //定义一个 Activity
5       @Override
6       public void onCreate(Bundle savedInstanceState) {  //重写的 onCreate 回调方法
7           super.onCreate(savedInstanceState);           //调用基类的 onCreate 方法
8           setContentView(R.layout.main);                //指定当前显示的布局
9       }
10  }
```

- 第 4 行是对继承自 Activity 子类的声明。
- 第 6～9 行重写了 Activity 的 onCreate 回调方法，在 onCreate 方法中先调用基类的 onCreate 方法，然后指定用户界面为 R.layout.main，对应的文件为 res/layout/main.xml。

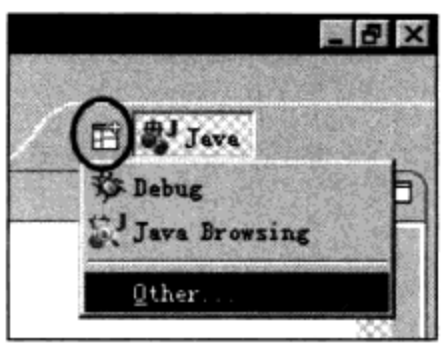
1.3.5 Android 程序的监控与调试

前面几个小节已经对 Android 应用程序的创建进行了详细讲解，本小节将介绍如何通过 DDMS 来监控 Android 应用程序的运行以及如何调试 Android 程序。在调试过程中使用了 android.util.log 类，该类简单易用。监控与调试详细步骤如下。

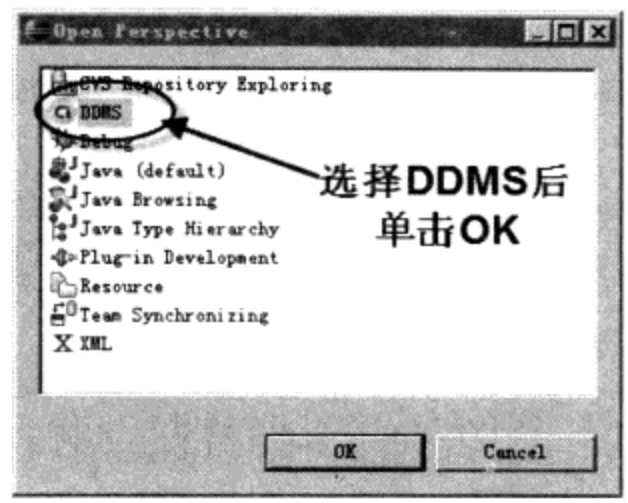
（1）打开刚刚建立的项目，找到 HelloAndroid.java 文件，在第 8 行 super.onCreate (savedInstanceState); 之后添加“Log.d("TAG", "This is message!");”语句。

（2）在项目名上单击鼠标右键，然后依次选择“Run As/Android Application”运行该项目。

（3）依次单击 Eclipse 右上角的 Open Perspective/Other，如图 1-25 所示。Eclipse 将弹出“Open Perspective”对话框，如图 1-26 所示。

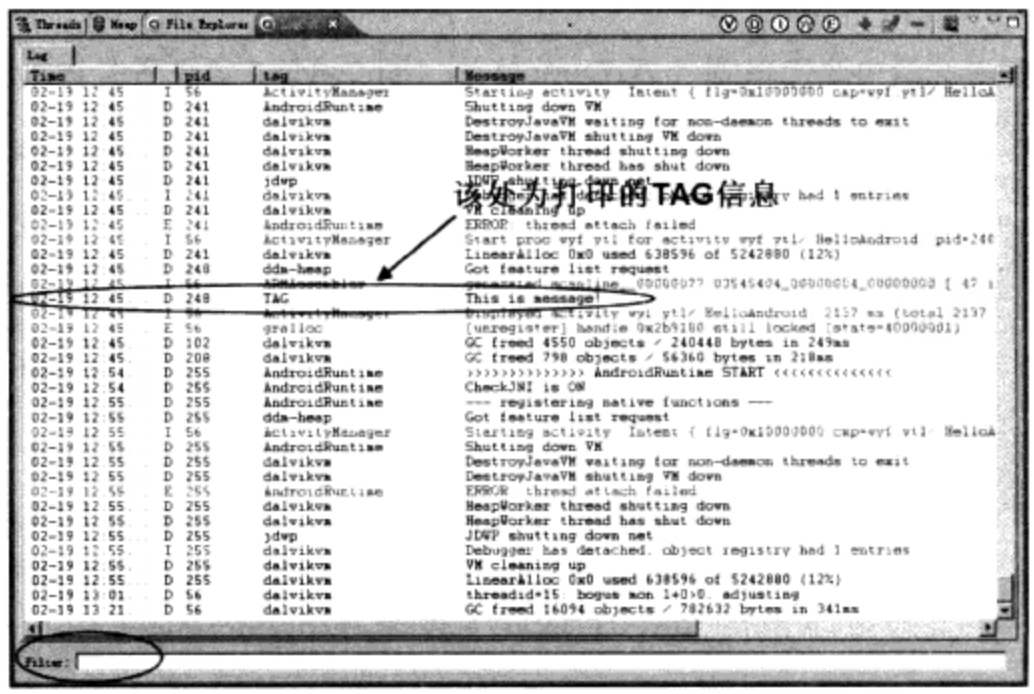


▲图 1-25　打开 Open Perspective 窗口

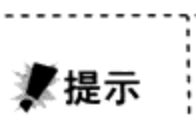


▲图 1-26　切换到 DDMS 视图

（4）单击“OK”按钮后将 Eclipse 切换到 DDMS 视角，如图 1-27 所示。LogCat 显示在屏幕的下方，系统中所有的日志都将出现在 LogCat 中，通过对 LogCat 的观察可以详细了解 Android 程序运行的过程。



▲图 1-27　LogCat 界面



在程序的开发和调试过程中，少不了对文本的输出，而在 Android 程序的开发中，建议使用 Log 类来打印需要打印的文本。

（5）在图 1-27 中可以看到在程序中添加的日志输出，这样在程序的开发过程中可以随时使用 Log 类来打印需要打印的信息，而当 LogCat 中日志过多时，可以使用过滤器 Filter 通过对 tag 进行过滤来筛选 log。

1.4 小结

本章介绍了 Android 平台的来源及优点，并详细介绍了在 Eclipse 中如何构建 Android 的开发环境，最后创建了第一个 Android 的应用程序。通过本章的学习，读者应该已经对 Android 平台下应用程序的开发步骤有了初步的了解。

第 2 章　Android 游戏开发之前台渲染

在正式动手开发游戏之前，有必要先对 Android 的前台渲染技术进行学习，掌握了这些知识，才能更好地学习游戏的开发。本章将对 Android 游戏开发的前台渲染技术进行详细介绍，为以后各章节的游戏开发做好技术储备。

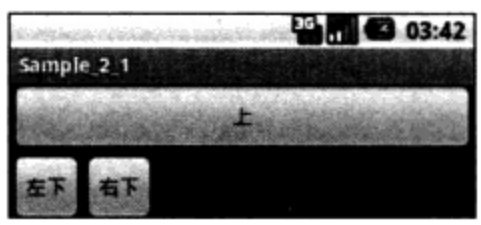
2.1 创建 Android 用户界面

任何程序都少不了用户界面，游戏也不例外，本节将对 Android 应用中常用的用户界面、布局管理以及简单的事件处理进行介绍。

2.1.1　布局管理

Android 的控件有很多种，需要使用 Layout 对这些控件进行管理，以使这些控件显示在屏幕的正确位置。Android SDK 中已经内置了 5 种布局模型，开发人员通过对这几种布局模型进行组合便可构建出各种复杂的用户界面。下面将对这 5 种布局模型进行详细的讲解。

1. 线性布局



▲ 图 2-1　线性布局

线性布局（LinearLayout）是 Android 应用程序中最简单的布局方式，有水平和竖直两种排列方式，通过对参数的设置可以控制各个控件在布局中的相对大小。接下来将通过一个简单的例子来介绍线性布局的使用方法，该例子最终的效果如图 2-1 所示，其开发步骤如下。

（1）创建一个新项目 Sample_2_1，然后在 res/layout 下创建 XML 布局文件 my_layout.xml，并在其中输入如下代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_1/res/layout 目录下的 my_layout.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="wrap_content"
6        >                                           <!--定义了一个线性布局，方式是垂直的-->
7          <Button
```

```
8            android:layout_width="fill_parent"
9            android:layout_height="fill_parent"
10           android:text="上"
11       />                                        <!-- 向线性布局中添加一个普通按钮控件-->
12       <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
13           android:orientation="horizontal"
14           android:layout_width="wrap_content"
15           android:layout_height="wrap_content"
16       >                                         <!-- 向线性布局中添加一个水平的线性布局-->
17           <Button
18               android:layout_width="wrap_content"
19               android:layout_height="wrap_content"
20               android:text="左下"
21           />                                    <!-- 向水平的线性布局中添加一个普通按钮控件-->
22           <Button
23               android:layout_width="wrap_content"
24               android:layout_height="wrap_content"
25               android:text="右下"
26           />                                    <!-- 向水平的线性布局中添加一个普通按钮控件-->
27   </LinearLayout>
28 </LinearLayout>
```

- 第 2～6 行是一个垂直的线性布局，宽度填满整个屏幕，高度自动适应子控件的大小。
- 第 7～11 行向线性布局中添加一个按钮控件，宽度和高度全部填满父控件。
- 第 12～16 行又向外层的线性布局中添加一个水平的线性布局，高度和宽度全部适应子控件。
- 第 17～26 行往水平的线性控件中依次添加两个按钮控件。

（2）打开 src/wyf/ytl 目录下的 Sample_2_1.java，将其代码修改为如下代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_1/src/wyf/ytl 目录下的 Sample_2_1.java。

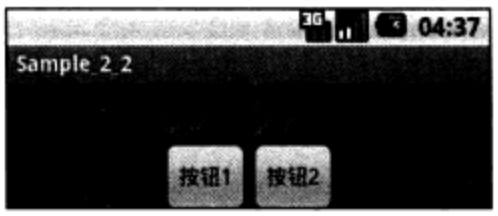
```
1   package wyf.ytl;
2   import android.app.Activity;                          //引入相关类
3   import android.os.Bundle;
4   public class Sample_2_1 extends Activity {            //定义一个 Activity
5       public void onCreate(Bundle savedInstanceState) {  //重写 onCreate 回调方法
6           super.onCreate(savedInstanceState);           //调用父类的 onCreate 方法
7           setContentView(R.layout.my_layout);           //设置当前用户界面
8       }
9   }
```

- 第 5～8 行重写 Activity 的 onCreate 方法，该方法在 Activity 创建时被系统调用。
- 第 9 行指定当前显示的主界面为 my_layout.xml。

提示　为了不干扰读者的思路，本章例子中按钮上的文字全部写在程序中，读者可以将这些文字提取到 strings.xml 文件中，为以后程序的维护与修改提供便利。

2. 表格布局

表格布局（TableLayout）是以行列的形式来管理子控件的，在表格布局中的每一行可以是一个



▲图 2-2 表格布局

View 控件或者是一个 TableRow 控件。而 TableRow 控件中还可以添加子控件。下面通过一个简单的例子来说明表格布局的使用方法，运行效果如图 2-2 所示，其开发步骤如下。

（1）创建一个新项目 Sample_2_2，然后在 res/layout 下创建 XML 布局文件 my_layout.xml，并在其中输入如下代码。

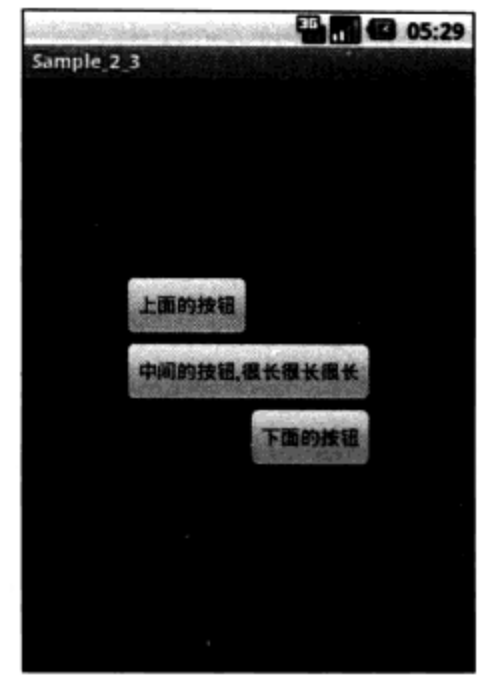
代码位置：见随书光盘中源代码/第 2 章/Sample_2_2/res/layout 目录下的 my_layout.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>        <!-- XML 的版本以及编码方式 -->
2    <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
3          android:layout_width="fill_parent"
4          android:layout_height="fill_parent"
5          >                                       <!-定义一个表格布局，宽高全部填满父控件-->
6          <TextView
7             android:layout_width="fill_parent"
8             android:layout_height="wrap_content"
9             android:gravity="center"
10            android:text="表头"
11         />                                      <!-在表格的第一行填充一个文本控件-->
12         <TableRow
13              android:gravity="center"
14         >                                       <!-再向表格中添加一行-->
15              <TextView
16                   android:layout_width="wrap_content"
17                   android:layout_height="wrap_content"
18                   android:text="第 0 列"
19              >                                  <!-在该行的第一列添加一个文本控件-->
20              </TextView>
21              <TextView
22                   android:layout_width="wrap_content"
23                   android:layout_height="wrap_content"
24                   android:text="第 1 列"
25              >                                  <!-在该行的第二列添加一个文本控件-->
26              </TextView>
27         </TableRow>
28         <TableRow
29              android:gravity="center"
30         >                                       <!-- 向表格中添加一行，对齐方式为居住-->
31              <Button
32                   android:layout_width="wrap_content"
33                   android:layout_height="wrap_content"
34                   android:text="按钮 1"
35              />                                 <!-往该行第一列添加一个按钮控件-->
36              <Button
37                   android:layout_width="wrap_content"
38                   android:layout_height="wrap_content"
39                   android:text="按钮 2"
40              />                                      <!-- 向第二列添加一个按钮控件-->
41         </TableRow>
42   </TableLayout>
```

- 第 2～5 行定义了一个宽高全部填满屏幕的 TableLayout。
- 第 6～11 行向 TableLayout 中的第一行添加一个 TextView 控件，并且设置对齐方式为居中。
- 第 22～27 行添加 TableRow 控件并继续添加两个自适应宽高的 TextView 控件。
- 第 28～41 行再向 TableLayout 中添加一行，并直接向该行中添加两列，每列一个按钮。

提示　表格布局中，每行可以是其他控件，而每列也可以是其他控件，在程序的开发过程中常常使用表格布局来排版。

（2）和前面的例子一样，编写完布局文件后，应该将该布局文件设为当前显示的用户界面，设置方法与线性布局的例子一样，因本书篇幅所限，就不再赘述，读者可以参考线性布局的例子自行更改。



▲图 2-3　相对布局

3. 相对布局

相对布局（RelativeLayout）也比较简单，其子控件是根据所设置的参照控件来进行布局的，设置的参照控件可以是父控件，也可以是其他的子控件。下面通过一个简单的例子来介绍相对布局的使用方法，该例子运行的效果如图 2-3 所示，其开发步骤如下。

（1）创建项目 Sample_2_3，在 res/layout 下添加布局文件 my_layout.xml，代码如下。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_3/res/layout 目录下的 my_layout.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>        <!-- XML 的版本以及编码方式 -->
2    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:layout_width="fill_parent"
4        android:layout_height="fill_parent"
5        >                                          <!-- 定义一个相对布局,宽高填满整个窗口 -->
6        <Button
7                android:id="@+id/button1"
8                android:layout_width="wrap_content"
9                android:layout_height="wrap_content"
10               android:text="中间的按钮,很长很长很长"
11               android:layout_centerInParent="true"
12       >
13       </Button>                          <!-- 添加一个 id 为 button1 的按钮,位于屏幕的正中间-->
14       <Button
15               android:id="@+id/button2"
16               android:layout_width="wrap_content"
17               android:layout_height="wrap_content"
18               android:text="上面的按钮"
19               android:layout_above="@id/button1"
20               android:layout_alignLeft="@id/button1"
21       >                                  <!-- 添加一个按钮，位于 button1 的左上方-->
22       </Button>
23       <Button
24               android:id="@+id/button3"
25               android:layout_width="wrap_content"
26               android:layout_height="wrap_content"
27               android:text="下面的按钮"
28               android:layout_below="@id/button1"
29               android:layout_alignRight="@id/button1"
30       >
```

```
31        </Button>                                <!-- 添加一个按钮，位于 button1 的右下方-->
32    </RelativeLayout>
```

- 在第 2～5 行定义了一个相对布局，宽高全部填满父控件，即填满整个屏幕。
- 第 6～12 行定义了一个参照按钮控件，宽高自适应，位置为父控件的正中间。
- 第 14～22 行添加一个按钮控件，宽高自适应，位于参照控件的左上方。
- 第 23～31 行继续添加一个按钮控件，宽高仍然是自适应，位于参照控件的右下方。

（2）接下来是更改 Sample_2_3.java 源代码，将刚刚编写的布局设为当前用户界面，设置方法同线性布局的例子一样，在此不再赘述，读者可查看随书光盘中源代码/第 2 章/Sample_2_3/src/wyf/ytl 下的 Sample_2_3.java 文件。

▲图 2-4　单帧布局

4. 单帧布局

单帧布局的使用方法更为简单，不需要任何特殊的配置，布局中的所有控件都被放置在布局的左上角。下面的例子是将 3 个 ImageView 控件添加到一个单帧布局中，效果如图 2-4 所示，其开发步骤如下。

（1）创建一个名为 Sample_2_4 的新项目，在该项目的 res/layout 下添加名为 my_layout.xml 的布局文件，其代码如下。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_4/res/layout 目录下的 my_layout.xml。

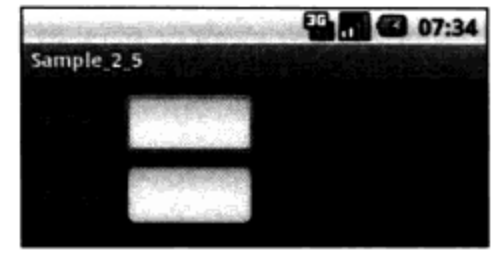
```
1    <?xml version="1.0" encoding="utf-8"?>       <!-- XML 的版本以及编码方式 -->
2    <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:layout_width="fill_parent"
4        android:layout_height="fill_parent"
5        >                                        <!-- 添加一个占满全屏幕的单帧布局 -->
6        <ImageView
7            android:layout_width="wrap_content"
8            android:layout_height="wrap_content"
9            android:src="@drawable/big"
10       >                                        <!-- 往单帧布局中添加一个 ImageView 控件-->
11       </ImageView>
12       <ImageView
13           android:layout_width="wrap_content"
14           android:layout_height="wrap_content"
15           android:src="@drawable/center"
16       >                                        <!-- 往单帧布局中添加一个 ImageView 控件-->
17       </ImageView>
18       <ImageView
19           android:layout_width="wrap_content"
20           android:layout_height="wrap_content"
21           android:src="@drawable/small"
22       >                                        <!-- 往单帧布局中添加一个 ImageView 控件--
23       </ImageView>
24   </FrameLayout>
```

- 在第 2～5 行定义了一个单帧布局，宽高全部填满父控件，即填满整个屏幕。
- 第 6～23 行向单帧布局中顺序添加 3 个 ImageView 控件。这些控件将全部对齐到布局的左上角，ImageView 控件的 android:src="@drawable/big"属性设置为 res/drawable-mdpi 下的图片文件。

提示　Android 平台中，图片等资源文件不支持大写字母，读者如希望使自己的资源图片名称更简洁、清晰，可以使用下划线来分割资源名中的单词。

（2）最后，同样是将刚编写的单帧布局设为当前显示的用户界面，设置方法同前面的例子完全相同，读者可以自行开发。

5. 坐标布局



▲图 2-5　坐标布局

下面介绍最后一个布局——坐标布局。该布局在程序开发中应用得较多，其中所有控件的位置坐标是直接指定的，这样屏幕中的控件位置完全由自己控制，增强了灵活性，下面的例子介绍了该布局的使用方法，运行效果如图 2-5 所示。

（1）创建新项目 Sample_2_5，在 res/layout 下创建 my_layout.xml，其代码如下。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_5/res/layout 目录下的 my_layout.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>        <!-- XML 的版本以及编码方式 -->
2    <AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:layout_width="fill_parent"
4        android:layout_height="fill_parent"
5        >                                          <!-定义一个填满整个屏幕的坐标布局-->
6        <TextView
7                android:layout_width="wrap_content"
8                android:layout_height="wrap_content"
9                android:text="用户名："
10               android:layout_x="10px"
11               android:layout_y="20px"
12       >                                          <!-- 在指定的位置上添加一个 TextView 控件-->
13       </TextView>
14       <EditText
15               android:layout_width="90px"
16               android:layout_height="wrap_content"
17               android:layout_x="70px"
18               android:layout_y="10px"
19       >                                          <!-- 在指定的位置上添加一个 EditText 控件-->
20       </EditText>
21       <TextView
22               android:layout_width="wrap_content"
23               android:layout_height="wrap_content"
24               android:text="密　码："
25               android:layout_x="10px"
26               android:layout_y="75px"
27       >                                          <!-- 在指定的位置上添加一个 TextView 控件-->
28       </TextView>
29       <EditText
30               android:layout_width="90px"
31               android:layout_height="wrap_content"
32               android:layout_x="70px"
33               android:layout_y="60px"
34       >                                          <!-- 在指定的位置上添加一个 EditText 控件-->
35       </EditText>
36   </AbsoluteLayout>
```

- 第 2～5 行定义了一个坐标布局，并设置其填满整个屏幕。

- 第 6～35 行向坐标布局中添加 4 个控件，通过 layout_x 和 layout_y 两个属性确定控件的位置。

（2）设置当前显示的用户界面为刚编写的布局界面，设置的方法与前面的例子相同，读者可查看随书光盘中源代码/第 2 章/ Sample_2_5/src/wyf/ytl 下的 Sample_2_5.java 文件。

2.1.2 常用控件及其事件处理

Android 中提供了大量丰富多彩的常用控件，Android 平台已经完整地实现了这些控件功能，开发人员只需简单的几句调用或参数设置的语句即可用其构建完整的用户界面。

Android 中控件的使用方法一般有两种，一种是在 XML 中配置，而另一种是在 Java 程序中直接调用。在游戏开发过程中，很少使用在 XML 中配置的方法，一般都是在 Java 源代码中直接使用。

常用的控件有 TextView、ImageView、CheckBox、RadioButton、Button、ImageButton、EditText、ToggleButton、AnalogClock、DigitalClock 等。这些控件的使用方法基本相同，接下来将通过一个简单的例子来讲解控件的使用方法以及 Android 平台的事件处理，具体开发步骤如下。

（1）创建一个名为 Sample_2_6 的新项目。

（2）打开 main.xml 布局文件，将其中的代码替换成下列代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_6/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>        <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                          <!-- 定义一个垂直的线性布局并填满这个屏幕 -->
7      <TextView
8          android:id="@+id/textView"
9          android:layout_width="fill_parent"
10         android:layout_height="wrap_content"
11         android:text="您没有单击任何按钮"
12     />                                             <!-- 先添加一个文本控件-->
13     <Button
14         android:id="@+id/button"
15         android:layout_width="wrap_content"
16         android:layout_height="wrap_content"
17         android:text="普通按钮"
18     >                                              <!-- 添加一个普通按钮控件 -->
19     </Button>
20     <ImageButton
21         android:id="@+id/imageButton"
22         android:layout_width="wrap_content"
23         android:layout_height="wrap_content"
24         android:src="@drawable/img"
25     >                                              <!-- 添加一个 ImageButton 按钮控件 -->
26     </ImageButton>
27     <ToggleButton
28         android:id="@+id/toggleButton"
29         android:layout_width="wrap_content"
30         android:layout_height="wrap_content"
31     >                                              <!-- 添加一个 ToggleButton 按钮控件 -->
32     </ToggleButton>
33   </LinearLayout>
```

- 第 2～5 行定义了一个垂直的线性布局，宽高全部填充满屏幕。

- 第 7～12 行向线性布局中添加一个文本控件，宽度填满父控件，高度自适应子控件，用于显示被单击的按钮。
- 之后依次向布局中添加 3 种按钮，每个控件的宽度和高度全部设置为自适应子控件，并指定其控件的 ID，使 Java 程序中可以找到该控件。

提示　该程序中用到了图片资源，在使用图片之前应该先将用到的图片资源放到 Sample_2_6/res/drawable-mdpi 文件夹下。

（3）打开系统自动创建的 Sample_2_6.java 文件，将其内容改为如下代码。

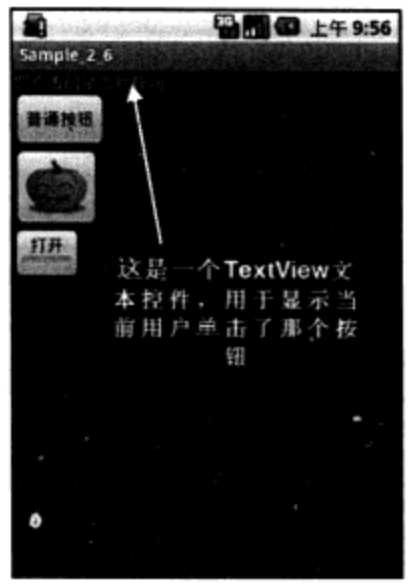
代码位置：见随书光盘中源代码/第 2 章/Sample_2_6/src/wyf/ytl 目录下的 Sample_2_6.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.os.Bundle;                                 //引入相关类
4    import android.view.View;                                 //引入相关类
5    import android.view.View.OnClickListener;                 //引入相关类
6    import android.widget.Button;                             //引入相关类
7    import android.widget.ImageButton;                        //引入相关类
8    import android.widget.TextView;                           //引入相关类
9    import android.widget.ToggleButton;                       //引入相关类
10   public class Sample_2_6 extends Activity implements OnClickListener{
11     Button button;                                          //普通按钮
12     ImageButton imageButton;                                //图片按钮
13     ToggleButton toggleButton;                              //开关按钮
14     TextView textView;                                      //文本控件
15       /** Called when the activity is first created. */
16       @Override
17       public void onCreate(Bundle savedInstanceState) {     //回调方法
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.main);                     //设置显示的 View
20          textView = (TextView) this.findViewById(R.id.textView);//得到文本控件的引用
21          button = (Button) this.findViewById(R.id.button);
22          toggleButton = (ToggleButton) this.findViewById(R.id.toggleButton);
23          imageButton = (ImageButton) this.findViewById(R.id.imageButton);
24          imageButton.setOnClickListener(this);              //为 imageButton 添加监听器
25          button.setOnClickListener(this);                   //为 button 添加监听器
26          toggleButton.setOnClickListener(this);             //为toggleButton添加监听器
27       }
28       public void onClick(View v) {                         //重写的事件处理回调方法
29         if(v == button){                                    //单击的是普通按钮
30            textView.setText("您单击的是普通按钮");
31         }
32         else if(v == imageButton){                          //单击的是图片按钮
33            textView.setText("您单击的是图片按钮");
34         }
35         else if(v == toggleButton){                         //单击的是开关按钮
36            textView.setText("您单击的是开关按钮");
37         }
38       }
39   }
```

- 第 11～14 行声明了 3 种按钮及文本控件的引用。
- 第 19 行是将 main 布局文件设置为当前显示的 View。
- 第 20～23 行通过布局文件中控件的 ID 得到各个控件的引用。

- 第 24～26 行为各个控件添加监听。
- 第 28～38 行重写了事件监听器的回调方法。当单击不同控件时分别做不同的处理。

（4）运行本例子，将看到如图 2-6 所示的效果，当用户单击按钮时，文本的内容会随之改变。



▲图 2-6　控件及事件监听

提示　在此通过 3 种按钮控件来讲解 Android 中控件的使用方法以及控件事件的处理方法，Android 中各个控件的使用方法基本相同，读者可以模仿本例学习其他控件的使用。

2.2 图形与动画在 Android 中的实现

程序开发过程中，图形、动画是必不可少的，本节将详细讲解图形、动画在 Android 中的实现，通过对本节的学习，使读者能够构建出简单的图形用户界面。

2.2.1 简单图形的绘制

在 Android 游戏的开发中，时时都需要绘制图形、图片，接下来将通过一个绘制图形的例子，让读者理解 Android 中图形绘制的机理，其开发步骤如下。

（1）创建一个新项目 Sample_2_7，然后在 src/wyf/ytl 目录下创建文件 MyView.java，并在其中输入下面的代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_7/src/wyf/ytl 目录下的 MyView.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    import android.content.Context;                                     //引入相关类
3    import android.graphics.Canvas;                                     //引入相关类
4    import android.graphics.Color;                                      //引入相关类
5    import android.graphics.Paint;                                      //引入相关类
6    import android.graphics.RectF;                                      //引入相关类
7    import android.util.AttributeSet;                                   //引入相关类
8    import android.view.View;                                           //引入相关类
9    public class MyView extends View{
10     public MyView(Context context, AttributeSet attrs) {              //构造器
11       super(context, attrs);
12     }
```

```
13      protected void onDraw(Canvas canvas) {                    //重写的绘制方法
14        super.onDraw(canvas);
15        canvas.drawColor(Color.BLACK);                          //绘制黑色背景
16        Paint paint = new Paint();                              //创建画笔
17        paint.setColor(Color.RED);                              //设置画笔颜色为红色
18        canvas.drawRect(10, 10, 110, 110, paint);               //绘制矩形
19        canvas.drawText("这是字符串", 10, 130, paint);          //字符串，以字符串下面
                                                                  //为基准
20        RectF rf1 = new RectF(10, 130, 110, 230);               //定义一个矩形
21        canvas.drawArc(rf1, 0, 45, true, paint);                //画弧，顺时针
22        canvas.drawLine(150, 10, 250, 110, paint);              //画线
23        RectF rf2 = new RectF(150, 130, 250, 230);              //定义一个矩形
24        canvas.drawOval(rf2, paint);                            //画圆
25      }
26    }
```

- 第 10～12 行为该类的构造器，View 下构造器有 3 种重载方式，如需在 XML 中配置应用该 View，必须实现该构造器。
- 第 13 行重写负责绘制的 onDraw 方法，第 15 行绘制 View 的背景色为黑色，而第 16～17 行创建并设置画笔。
- 第 18 行向 View 中绘制矩形，drawRect 的前 4 个参数分别为左上角的 x、y 和右下角的 x、y。
- 第 19 行向 View 中绘制字符串，需要注意的是，之后的两个参数为字符串所占矩形的左下角 x、y。
- 第 20～21 行先创建一个矩形对象，然后根据矩形对象的大小和位置绘制弧形，给出起始的角度和终止的角度。
- 第 22 行绘制直线，前 4 个参数为起始点的 x、y 以及终点的 x、y。第 23～24 行同样创建一个矩形对象，并根据矩形绘制圆。

（2）将 res/layout 下的 main.xml 文件的代码改为如下代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_7/res/layout 目录下的 main.xml。

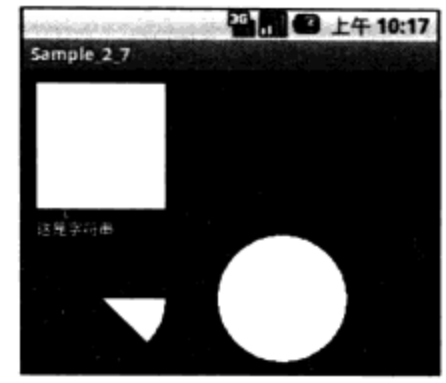
```
1    <?xml version="1.0" encoding="utf-8"?>        <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                          <!--添加一个垂直的线性布局，并填满屏幕-->
7        <wyf.ytl.MyView
8            android:id="@+id/myView"
9                android:layout_width="fill_parent"
10               android:layout_height="fill_parent"
11       />                                         <!-- 添加自定义的 View 到线性布局中-->
12   </LinearLayout>
```

说明　代码的第 7～10 行在垂直的线性布局中添加自定义的 View，并为它指定 ID，宽度和高度全部填充父控件。

（3）在模拟器中运行该项目，观察的效果如图 2-7 所示。

提示　在 XML 中配置自定义的 View 需要使用全称类名，在该例子中，也可以使用除线性布局之外其他布局，效果是一样的。

2.2.2 贴图的艺术



▲图 2-7 简单的图形绘制

前面的小节介绍了在 Android 平台下如何绘制基本的图形，但是只有这些图形往往是不能满足应用开发需求的，接下来将通过一个贴图的小例子，使读者学会如何在 Android 平台下绘制图片，详细步骤如下。

（1）创建一个新项目，取名为 Sample_2_8。

（2）将程序中用到的图片 img.png 放到 Sample_2_8/res/drawable-mdpi 文件夹下。

（3）在该项目的 src 下创建文件 MyView.java，并输入如下代码。

代码位置：见随书光盘中源代码/第二章/Sample_2_8/src/wyf/ytl 目录下的 MyView.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.content.Context;                           //引入 Context 类
3    import android.graphics.Bitmap;                           //引入 Bitmap 类
4    import android.graphics.BitmapFactory;                    //引入 BitmapFactory 类
5    import android.graphics.Canvas;                           //引入 Canvas 类
6    import android.graphics.Color;                            //引入 Color 类
7    import android.graphics.Paint;                            //引入 Paint 类
8    import android.util.AttributeSet;                         //引入 AttributeSet 类
9    import android.view.View;                                 //引入 View 类
10   public class MyView extends View{                         //继承自 View
11     Bitmap myBitmap;                                        //图片的引用
12     Paint paint;                                            //画笔的引用
13     public MyView(Context context, AttributeSet attrs) {    //构造器
14       super(context, attrs);
15       this.initBitmap();                                    //调用初始化图片的方法
16     }
17     public void initBitmap(){
18       paint = new Paint();                                  //创建一个画笔
19       myBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.img);
20     }
21     protected void onDraw(Canvas canvas) {                  //重写的绘制方法
22       super.onDraw(canvas);
23       paint.setAntiAlias(true);                             //打开抗锯齿
24       paint.setColor(Color.WHITE);                          //设置画笔的颜色
25       paint.setTextSize(15);
26       canvas.drawBitmap(myBitmap, 10, 10, paint);           //绘制图片
27       canvas.save();                                        //保存画布状态
28       Matrix m1=new Matrix();                               //创建 Matrix 对象
29       m1.setTranslate(500, 10);                             //平移矩阵
30       Matrix m2=new Matrix();                               //创建 Matrix 对象
31       m2.setRotate(15);                                     //以一定的角度旋转矩阵
32       Matrix m3=new Matrix();                               //创建 Matrix 对象
33       m3.setConcat(m1, m2);
34       m1.setScale(0.8f, 0.8f);                              //缩放矩阵
35       m2.setConcat(m3, m1);
36       canvas.drawBitmap(myBitmap, m2, paint);               //绘制图片
37       canvas.restore();                                     //恢复画布状态
38       canvas.save();                                        //保存画布状态
39       paint.setAlpha(180);                                  //设置透明度
40       m1.setTranslate(200,100);                             //平移矩阵
41       m2.setScale(1.3f, 1.3f);                              //缩放矩阵
42       m3.setConcat(m1, m2);
```

```
43          canvas.drawBitmap(myBitmap, m3,  paint);          //绘制图片
44          paint.reset();                                    //恢复画笔设置
45          canvas.restore();                                 //恢复画布
46          paint.setTextSize(40);                            //设置字体大小
47          paint.setColor(0xffFFFFFF);                       //设置画笔颜色
48          canvas.drawText("图片的宽度:"+myBitmap.getWidth(),20,220,paint);
49          canvas.drawText("图片的高度:"+myBitmap.getHeight(),150,220,paint);
50          paint.reset();                                    //恢复画笔设置
51        }
52    }
```

- 第 10 行自定义一个继承自 View 的子类。第 13～17 行为 MyView 的构造器，在构造器中调用初始化图片的方法来初始化需要用到的图片。
- 第 18～19 行创建一个画笔，获取图片资源。
- 第 21 行重写了 onDraw 方法，用于绘制 View 中需要显示的内容。
- 第 23～24 行打开抗锯齿功能，并设置画笔的颜色为白色，第 25 行设置字体的大小。
- 第 27～37 行首先保存画布状态，然后设置相应的平移、旋转以及缩放，调用 setConcat 方法，将对应的两个 Matrix 对象连接起来并绘制，最后恢复画布状态。
- 第 38～45 行首先保存画布状态，然后调用 setTranslate 方法平移矩阵，调用 setScale 方法缩放矩阵，调用 setConcat 方法链接两个 Matrix 对象并绘制图片，最后恢复画布状态。
- 第 46～50 行首先设置字体的大小和画笔的颜色，然后绘制两个字符串“图片的宽度”以及“图片的高度”，最后恢复画笔设置。

（4）打开/res/layout 下的 main.xml 文件，将其代码替换为如下代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_8/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                           <!-- 定义一个线性布局 -->
7    <wyf.ytl.MyView
8      android:id="@+id/myView"
9        android:layout_width="fill_parent"
10       android:layout_height="fill_parent"
11       />                                          <!--将自定义的 View 添加到布局中 -->
12   </LinearLayout>
```

- 第 2～6 行定义一个垂直的线性布局，宽度和高度填满整个屏幕。
- 第 7～11 行向线性布局中添加一个自定义的 View 并填满父控件。

（5）运行该项目其效果如图 2-8 所示。

▲图 2-8　贴图实例

2.2.3　自定义动画的播放

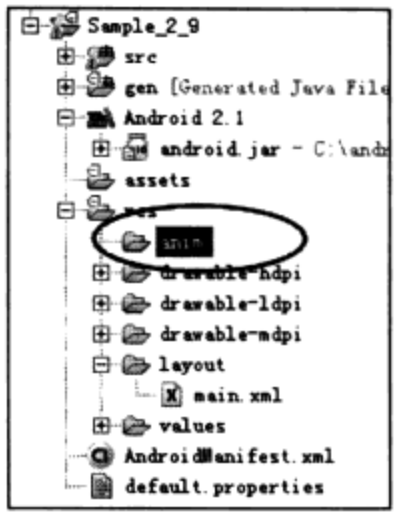
Android 中主要有两种动画模式，一种是 tweened animation（渐变动画），即通过对场景里的对象不断做图像变换产生动画效果；一种是 frame by frame（帧动画），即按顺序播放事先配置好的动画帧。

渐变动画有 4 种动画类型，alpha（透明度）、scale（尺寸伸缩）、translate（位置变换）和 rotate（图形旋转）。接下来通过一个动画播放的例子来介绍，如何在 Android 中构建并播放自己的动画，详细步骤如下。

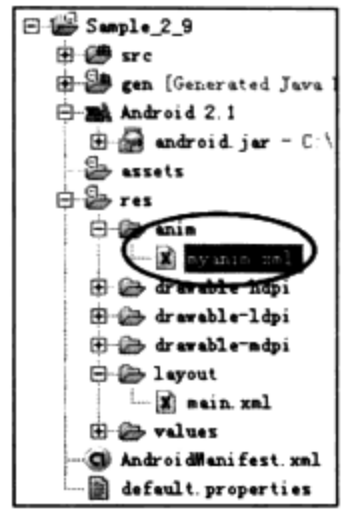
（1）创建新项目 Sample_2_9，然后将项目中用到的图片 img.png 放到 res/drawable-mdpi 下。

（2）在 res 目录中新建 anim 文件夹，如图 2-9 所示。

（3）在新建的 anim 文件夹中新建文件 myanim.xml，如图 2-10 所示。



▲图 2-9　anim 文件夹



▲图 2-10　myanim.xml 文件

（4）打开 myanim.xml 文件，输入如下代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_9/res/anim 目录下的 myanim.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>                <!-- XML 的版本以及编码方式 -->
2    <set xmlns:android="http://schemas.android.com/apk/res/android">
3      <alpha
4       android:fromAlpha="0.1"
5       android:toAlpha="1.0"
6       android:duration="2000"
7      />                                                   <!-- 透明度的变换 -->
8      <scale
9       android:interpolator= "@android:anim/accelerate_decelerate_interpolator"
10      android:fromXScale="0.0"
11      android:toXScale="1.4"
12      android:fromYScale="0.0"
13      android:toYScale="1.4"
14      android:pivotX="50%"
15      android:pivotY="50%"
16      android:fillAfter="false"
17      android:duration="3000"
18     />                                                   <!-- 尺寸的变换 -->
19     <translate
20     android:fromXDelta="30"
21     android:toXDelta="0"
22     android:fromYDelta="30"
23     android:toYDelta="50"
24     android:duration="3000"
25     />                                                   <!-- 位置的变换 -->
26     <rotate
27     android:interpolator="@android:anim/accelerate_decelerate_interpolator"
28     android:fromDegrees="0"
29     android:toDegrees="+350"
```

```
30      android:pivotX="50%"
31      android:pivotY="50%"
32      android:duration="3000"
33      />                                                    <!-- 旋转变换 -->
34    </set>
```

- 第 1 行定义了 XML 的版本以及编码方式。第 3～7 行定义透明度控制动画效果，fromAlpha 属性为动画起始时透明度，toAlpha 属性为动画结束时的透明度，duration 为动画持续的时间。
- 第 8～18 行定义了尺寸变换动画，interpolator 指定一个动画的插入器，fromXScale 属性为动画起始时 x 坐标上的伸缩尺寸，toXScale 属性为动画结束时 x 坐标上的伸缩尺寸，fromYScale 属性为动画起始时 y 坐标上的伸缩尺寸，toYScale 属性为动画结束时 y 坐标上的伸缩尺寸，pivotX 和 pivotY 设置动画相对于自身的位置，fillAfter 表示动画的转换在动画结束后是否被应用。
- 第 19～25 行定义了位置的变换动画，fromXDelta 属性为动画起始时 x 坐标上的位置，toXDelta 属性为动画结束时 x 坐标上的位置，fromYDelta 属性为动画起始时 y 坐标上的位置，toYDelta 属性为动画结束时 y 坐标上的位置。
- 第 26～33 行定义了旋转动画，interpolator 同样为一个动画的插入器，fromDegrees 属性为动画起始时物件的角度，toDegrees 属性为动画结束时物件旋转的角度。

（5）打开 res/layout 下的 main.xml 文件，将其中的代码改为如下代码。

代码位置：见随书光盘中源代码/第二章/Sample_2_9/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>    <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6    >                                    <!-- 定义一个垂直的线性布局-->
7    <ImageView
8      android:id="@+id/myImageView"
9      android:layout_width="fill_parent"
10     android:layout_height="fill_parent"
11     android:src="@drawable/img"
12   />                                   <!--添加一个 id 为 myImageView 的 ImageView 控件-->
13   </LinearLayout>
```

- 第 2～6 行定义一个垂直的线性布局，且填满整个屏幕。
- 第 7～12 行添加一个 id 为 myImageView 的 ImageView 控件到线性布局中，并设置其填充方式为填满整个父控件。

（6）将 src/wyf/ytl 下的 Sample_2_9.java 文件代码改为如下代码。

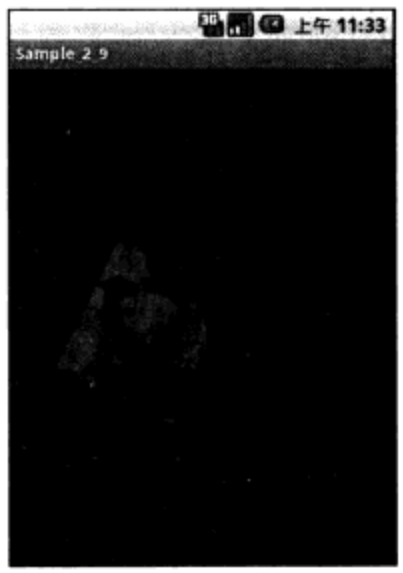
代码位置：见随书光盘中源代码/第二章/Sample_2_9/src/wyf/ytl 目录下的 Sample_2_9.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.app.Activity;                              //引入 Activity 类
3    import android.os.Bundle;                                 //引入 Bundle 类
4    import android.view.animation.Animation;                  //引入 Animation 类
5    import android.view.animation.AnimationUtils;             //引入 AnimationUtils 类
6    import android.widget.ImageView;                          //引入 ImageView 类
7    public class Sample_2_9 extends Activity {
8      Animation myAnimation;                                  //动画的引用
9      ImageView myImageView;                                  //ImageView 的引用
```

```
10      /** Called when the activity is first created. */
11      @Override
12      public void onCreate(Bundle savedInstanceState) {      //重写的 onCreate 回调方法
13          super.onCreate(savedInstanceState);
14          setContentView(R.layout.main);                     //设置当前显示的 View
15          myAnimation= AnimationUtils.loadAnimation(this,R.anim.myanim);
                                                               //加载动画
16          myImageView =(ImageView)this.findViewById(R.id.myImageView);
                                                               //得到 ImageView 的引用
17          myImageView.startAnimation(myAnimation);           //启动动画
18      }
19  }
```

- 第 8 行声明一个动画的引用，第 12 行重写了 onCreate 回调方法，该方法会在 activity 第一次被创建时被调用。
- 第 15 行加载之前编写的动画。
- 第 16～17 行得到 main 中 ImageView 的引用并对其使用动画。

（7）运行该项目，运行效果如图 2-11、图 2-12 和图 2-13 所示。



▲图 2-11　刚开始启动动画

▲图 2-12　动画运行过程中

▲图 2-13　动画结束后

2.3 Android 平台下的多媒体开发

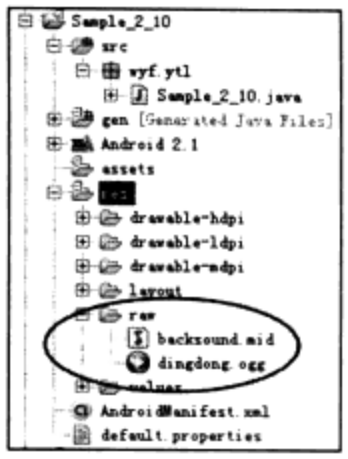
前面的小节对 Android 中图形与动画进行了介绍，可是只有图形界面并不能满足游戏开发的需求，音频、视频在游戏开发过程中往往也会用得上，本节将对 Android 平台下的多媒体开发进行介绍。

2.3.1　音频的播放

Android 平台中关于音频的播放有两种方式，一种是 SoundPool，另一种是 MediaPlayer。SoundPool 适合短促但对反应速度要求较高的情况（如游戏中的爆炸声），而 MediaPlayer 则适合较长但对时间要求不高的情况。读者可以根据具体需求自行选择，下面将通过一个音频播放的例子详细介绍这两种声音播放的方式，步骤如下。

（1）创建一个新的 Android 项目，起名为 Sample_2_10。

（2）在 res 目录下创建名为 raw 的文件夹，然后将程序中所用到的声音资源全部放进该文件夹，如图 2-14 所示。



▲图 2-14　raw 文件夹

（3）打开 res/layout 下的 main.xml，用下面的代码替代之前的代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_10/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>        <!-- XML 的版本以及编码方式 -->
2    <LinearLayout   xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6    >                                             <!--定义一个垂直的线性布局 -->
7        <TextView
8            android:id="@+id/textView"
9                android:layout_width="fill_parent"
10               android:layout_height="wrap_content"
11               android:text="没有播放任何声音"
12       />                                        <!-- 向线性布局中添加一个TextView控件 -->
13           <Button
14               android:id="@+id/button1"
15               android:layout_width="wrap_content"
16               android:layout_height="wrap_content"
17               android:text="使用 MediaPlayer 播放声音"
18           />                                    <!-- 向线性布局中添加一个 Button 控件 -->
19           <Button
20               android:id="@+id/button2"
21               android:layout_width="wrap_content"
22               android:layout_height="wrap_content"
23               android:text="暂停 MediaPlayer 声音"
24           />                                    <!-- 向线性布局中添加一个 Button 控件 -->
25           <Button
26               android:id="@+id/button3"
27               android:layout_width="wrap_content"
28               android:layout_height="wrap_content"
29               android:text="使用 SoundPool 播放声音"
30           />                                    <!-- 向线性布局中添加一个 Button 控件 -->
31           <Button
32               android:id="@+id/button4"
33               android:layout_width="wrap_content"
34               android:layout_height="wrap_content"
35               android:text="暂停 SoundPool 声音"
36       />                                        <!-- 向线性布局中添加一个 Button 控件 -->
37   </LinearLayout>
```

- 第 2～6 行定义一个垂直的线性布局，并将宽和高设置成自动填满父控件。
- 第 7～12 行向线性布局中添加一个 TextView 控件，并为其添加 ID。
- 第 13～36 行向线性布局中添加 4 个按钮控件，并分别为其添加 ID、设置按钮上显示的文字。

（4）打开 src/wyf/ytl 下的 Sample_2_10.java 文件，将其代码替换成下面的代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_10/src/wyf/ytl 目录下的 Sample_2_10.java。

```
1    package wyf.ytl;                                  //声明包语句
2    import java.util.HashMap;                         //引入 HashMap 类
3    import android.app.Activity;                      //引入 Activity 类
4    import android.content.Context;                   //引入 Context 类
```

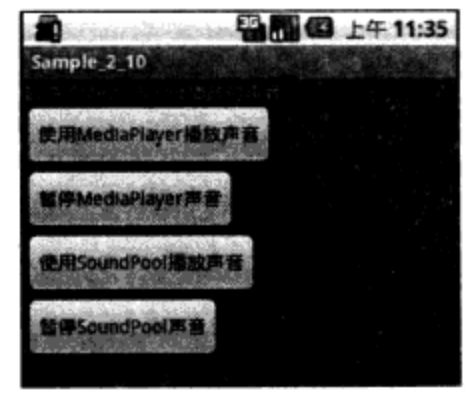
```
5    import android.media.AudioManager;                          //引入 AudioManager 类
6    import android.media.MediaPlayer;                           //引入 MediaPlayer 类
7    import android.media.SoundPool;                             //引入 SoundPool 类
8    import android.os.Bundle;                                   //引入 Bundle 类
9    import android.view.View;                                   //引入 View 类
10   import android.view.View.OnClickListener;                   //引入 OnClickListener 类
11   import android.widget.Button;                               //引入 Button 类
12   import android.widget.TextView;                             //引入 TextView 类
13   public class Sample_2_10 extends Activity implements OnClickListener{
14          Button button1;                                      //四个按钮的引用
15          Button button2;
16          Button button3;
17          Button button4;
18          TextView textView;                                   //TextView 的引用
19     MediaPlayer mMediaPlayer;                                 // MediaPlayer 的引用
20     SoundPool soundPool;                                      // SoundPool 的引用
21     HashMap<Integer, Integer> soundPoolMap;
22       /** Called when the activity is first created. */
23       @Override
24       public void onCreate(Bundle savedInstanceState) {       //重写 onCreate 回调方法
25          super.onCreate(savedInstanceState);
26          initSounds();                                        //初始化声音
27          setContentView(R.layout.main);                       //设置显示的用户界面
28          textView = (TextView) this.findViewById(R.id.textView);
                                                                 //得到 TextView 的引用
29          button1 = (Button) this.findViewById(R.id.button1);//得到 button1 的引用
30          button2 = (Button) this.findViewById(R.id.button2);//得到 button2 的引用
31          button3 = (Button) this.findViewById(R.id.button3);//得到 button3 的引用
32          button4 = (Button) this.findViewById(R.id.button4);//得到 button4 的引用
33          button1.setOnClickListener(this);                    //为按钮添加监听
34          button2.setOnClickListener(this);                    //为按钮添加监听
35          button3.setOnClickListener(this);                    //为按钮添加监听
36          button4.setOnClickListener(this);                    //为按钮添加监听
37        }
38     public void initSounds(){                                 //初始化声音的方法
39          mMediaPlayer = MediaPlayer.create(this, R.raw.backsound);
                                                                 //初始化 MediaPlayer
40          soundPool = new SoundPool(4, AudioManager.STREAM_MUSIC, 100);
41          soundPoolMap = new HashMap<Integer, Integer>();
42          soundPoolMap.put(1, soundPool.load(this, R.raw.dingdong, 1));
43     }
44     public void playSound(int sound, int loop) {          //用 SoundPoll 播放声音的方法
45         AudioManager mgr = (AudioManager)this.getSystemService(Context.AUDIO_
           SERVICE);
46         float streamVolumeCurrent = mgr.getStreamVolume(AudioManager.STREAM_MUSIC);
47         float streamVolumeMax = mgr.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
48         float volume = streamVolumeCurrent/streamVolumeMax;
49         soundPool.play(soundPoolMap.get(sound), volume, volume, 1, loop, 1f);
                                                            //播放声音
50     }
51     public void onClick(View v) {                         //实现接口中的方法
52         // TODO Auto-generated method stub
53         if(v == button1){                                 //单击使用 MediaPlayer 播放声音按钮
54              textView.setText("使用 MediaPlayer 播放声音");
55              if(!mMediaPlayer.isPlaying()){               //如果当前没有音乐正在播放
56                   mMediaPlayer.start();                   //播放声音
57              }
58         }
59         else if(v == button2){                            //单击了暂停 MediaPlayer 声音按钮
60              textView.setText("暂停了 MediaPlayer 播放的声音");
61              if(mMediaPlayer.isPlaying()){                //如果当前有音乐正在播放
62                   mMediaPlayer.pause();                   //暂停声音
```

```
63                  }
64              }
65              else if(v == button3){                    //单击了使用 SoundPool 播放声音按钮
66                  textView.setText("使用 SoundPool 播放声音");
67                  this.playSound(1, 0);                 //播放声音
68              }
69              else if(v == button4){                    //单击了暂停 SoundPool 声音按钮
70                  textView.setText("暂停了 SoundPool 播放的声音");
71                  soundPool.pause(1);                   //暂停 SoundPool 的声音
72              }}}
```

- 第 24 行为 onCreate 回调方法的开始，该方法在 Activity 第一次创建时被调用。
- 第 28～32 行得到程序中会用到的 TextView 及 Button 的引用，第 33～36 行为各个按钮添加监听。
- 第 38～43 行初始化所有声音资源，使用 SoundPool 时一般将声音放进一个 HashMap 中，便于声音的管理与操作。
- 第 44～50 行为用 SoundPool 播放声音的方法，程序的其他地方需要播放声音时，只需调用该方法即可。在该方法中，先对声音设备进行配置，然后调用 SoundPool 的 play 方法来播放声音。
- 第 51～72 行是对各个按钮被按下的事件进行处理，其中第 53～58 行为单击“使用 MediaPlayer 播放声音”按钮的处理代码，先判断当前 mMediaPlayer 是否正在播放，当没有在播放时，播放声音。

提示　SoundPool 初始化的过程是异步的，也就是说，当对 SoundPool 初始化时系统会自动启动一个后台线程来完成初始化工作。因此，并不会影响前台其他程序的运行，但也带来一个问题，调用初始化操作后不能立即播放，需要等待一下，否则可能会出错。另外，SoundPool 可以同时播放多个音频文件，但 MediaPlayer 同一时间却只能播放一个。

（5）运行该程序得到的效果如图 2-15 所示，根据单击按钮的不同采用不同的声音播放方式播放声音。



▲图 2-15　声音播放示例

2.3.2　视频的播放

前面已经介绍了音频的播放，本节将对 Android 中视频的播放进行简单的介绍，同样是通过一个例子来讲解视频的播放方法，详细步骤如下。

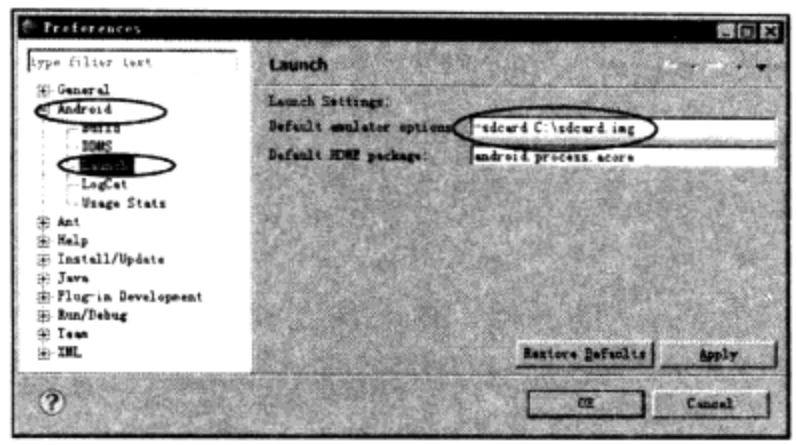
（1）创建 sdcard 镜像文件，创建方法是在命令行下输入 mksdcard 512M C:\sdcard.img 命令。

（2）启动 Eclipse，然后依次单击“Window/Preferences”菜单，弹出如图 2-16 所示的 Preferences 窗口。

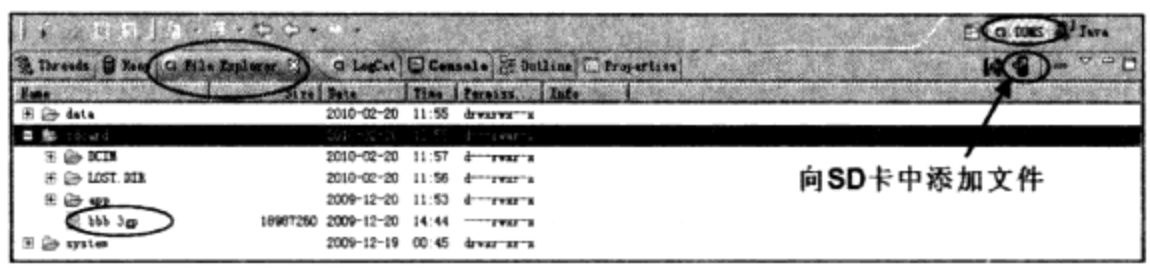
（3）依次单击“Android/Launch”，然后在右侧的“Default emulator options”中填上“-sdcard C:\sdcard.img”，然后单击“OK”按钮，如图 2-16 所示。

（4）运行模拟器，然后将 Eclipse 切换到 DDMS 视图，找到 File Explorer 窗口，通过右上角的

“Push a file onto the device”按钮向 sdcard 中添加程序中用到的视频文件 bbb.3gp，如图 2-17 所示。



▲图 2-16　Preferences 窗口



▲图 2-17　DDMS 视图

（5）创建一个名为 Sample_2_11 的项目。

（6）打开 res/layout 下的 main.xml 文件，更改其代码为下面的代码。

代码位置：见随书光盘中源代码/第二章/Sample_2_11/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>      <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                       <!--添加一个垂直的线性布局 -->
7          <SurfaceView
8          android:id="@+id/surfaceView"
9               android:layout_width="320px"
10              android:layout_height="200px"
11       />                                      <!--添加一个 SurfaceView 用于播放视频 -->
12       <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
13              android:layout_width="fill_parent"
14              android:layout_height="wrap_content"
15       >                                       <!--添加一个线性布局 -->
16         <Button
17         android:id="@+id/play2_Button"
18              android:layout_width="wrap_content"
19              android:layout_height="wrap_content"
20              android:text="播放"
21              />                              <!--添加一个按钮 -->
22              <Button
23              android:id="@+id/pause2_Button"
24                   android:layout_width="wrap_content"
25                   android:layout_height="wrap_content"
26                   android:text="暂停"
27              />                              <!--添加一个按钮 -->
28       </LinearLayout>
```

```
29    </LinearLayout>
```

- 第 7～11 行向线性布局中添加一个 SurfaceView 控件。
- 第 12～28 行在一个线性布局中添加两个按钮。

（7）打开 src/wyf/ytl 下的 Sample_2_11.java 文件，改为如下代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_11/src/wyf/ytl 目录下的 Sample_2_11.java。

```
1     package wyf.ytl;                                          //声明包语句
2     import android.app.Activity;                              //引入 Activity 类
3     ……//该处省略了部分类的引入代码，读者可自行查阅随书光盘中的源代码
4     import android.widget.Button;                             //引入 Button 类
5     public class Sample_2_11 extends Activity implements OnClickListener,SurfaceHolder.
      Callback{
6        String path = "/sdcard/bbb.3gp";                       //视频的路径
7        Button play_Button;                                    //按钮的引用
8        Button pause_Button;
9        boolean isPause = false;
10       SurfaceHolder surfaceHolder;
11       MediaPlayer mediaPlayer;                               //MediaPlayer 的引用
12       SurfaceView surfaceView;
13           public void onCreate(Bundle savedInstanceState) {  //重写的方法
14           super.onCreate(savedInstanceState);
15           setContentView(R.layout.main);                     //设置当前显示的用户界面
16           play_Button = (Button) findViewById(R.id.play2_Button);
17           play_Button.setOnClickListener(this);              //添加监听
18           pause_Button = (Button) findViewById(R.id.pause2_Button);
19           pause_Button.setOnClickListener(this);             //添加监听
20           getWindow().setFormat(PixelFormat.UNKNOWN);
21           surfaceView = (SurfaceView) findViewById(R.id.surfaceView);
22           surfaceHolder = surfaceView.getHolder();
23           surfaceHolder.addCallback(this);                   //添加回调
24           surfaceHolder.setFixedSize(176,144);
25           surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
26           mediaPlayer = new MediaPlayer();                   //创建 MediaPlayer
27       }
28    public void onClick(View v) {
29           if(v == play_Button){                              //按下播放电影按钮
30                isPause = false;
31                playVideo(path);
32           }
33           else if(v == pause_Button){                        //按下暂停按钮，
34                if(isPause == false){                         //如果正在播放则将其暂停
35                     mediaPlayer.pause();
36                     isPause = true;
37                }
38                else{                                         //如果暂停中怎么继续播放
39                     mediaPlayer.start();
40                     isPause = false;
41                }
42           }
43    }
44    private void playVideo(String strPath){                   //自定义播放影片函数
45           if(mediaPlayer.isPlaying()==true){
46                mediaPlayer.reset();
47           }
48           mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
49           mediaPlayer.setDisplay(surfaceHolder);             //设置 Video 影片以
                                                                //SurfaceHolder 播放
50           try{
```

```
51                  mediaPlayer.setDataSource(strPath);          //设置路径
52                  mediaPlayer.prepare();
53              }
54              catch (Exception e){
55                  e.printStackTrace();                         //打印异常信息
56              }
57              mediaPlayer.start();                             //开始播放视频
58          }
59          public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {}
60          public void surfaceCreated(SurfaceHolder arg0) {}
61          public void surfaceDestroyed(SurfaceHolder arg0) {}
62      }
```

- 第 13～27 行重写 onCreate 方法，在方法中得到所需要使用的控件的引用。
- 第 28～43 行为事件响应的方法，当单击“播放”按钮时播放视频，当单击“暂停”按钮时暂停播放视频。
- 第 44～58 行为自定义的视频播放方法，如果视频正在播放则重置视频，设置一些视频的参数后开始播放视频。
- 第 59～61 行为接口中方法的空实现。

（8）运行该项目可在模拟器中看到如图 2-18 所示的效果。

▲图 2-18　视频播放

2.3.3　Camera 图像采集

本节将介绍如何制作一个简单的相机，步骤如下。

（1）创建一个名为 Sample_2_12 的新项目。

（2）为照相添加权限，添加方法是在 AndroidManifest.xml 文件中的</manifest>之前加上<uses-permission android:name="android.permission.CAMERA"/>即可。

（3）打开 res/layout 下的 main.xml 文件，更改其代码为下面的代码。

代码位置：见随书光盘中源代码/第 2 章/Sample_2_12/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                           <!--添加一个垂直的线性布局 -->
7          <SurfaceView
8          android:id="@+id/surfaceView"
9              android:layout_width="320px"
10             android:layout_height="240px"
11       />                                          <!--添加一个 SurfaceView 用于浏览 -->
12       <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
13             android:layout_width="fill_parent"
14             android:layout_height="wrap_content"
15       >                                           <!--添加一个线性布局 -->
16         <Button
17             android:id="@+id/button1"
18                 android:layout_width="wrap_content"
19                 android:layout_height="wrap_content"
20                 android:text="打开"
21         />                                        <!--添加一个按钮 -->
22         <Button
23             android:id="@+id/button2"
```

```
24                  android:layout_width="wrap_content"
25                  android:layout_height="wrap_content"
26                  android:text="关闭"
27          />                                          <!--添加一个按钮 -->
28      </LinearLayout>
29  </LinearLayout>
```

- 第 2～6 行定义一个垂直的线性布局。
- 第 12 行向垂直的线性布局内部添加一个线性布局。
- 第 16～27 向线性布局中添加两个按钮。

（4）打开 src/wyf/ytl 下的 Sample_2_12.java 文件，改为如下代码。

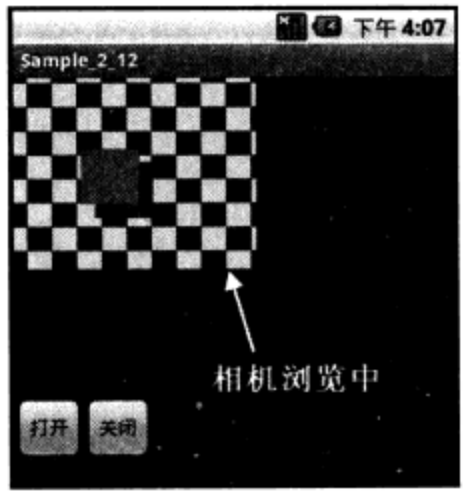
代码位置：见随书光盘中源代码/第二章/Sample_2_12/src/wyf/ytl 目录下的 Sample_2_12.java。

```
1    package wyf.ytl;
2    import java.io.IOException;                                  //引入相关类
3    import android.app.Activity;                                 //引入相关类
4    import android.hardware.Camera;                              //引入相关类
5    import android.os.Bundle;                                    //引入相关类
6    import android.view.SurfaceHolder;                           //引入相关类
7    import android.view.SurfaceView;                             //引入相关类
8    import android.view.View;                                    //引入相关类
9    import android.widget.Button;                                //引入相关类
10   public class Sample_2_12 extends Activity implements SurfaceHolder.Callback{
11       Camera myCamera;                                         //Camera 的引用
12       SurfaceView mySurfaceView;                               //SurfaceView 的引用
13       SurfaceHolder mySurfaceHolder;                           //SurfaceHolder 的引用
14       Button button1;                                          //按钮的引用
15       Button button2;
16       boolean isPreview = false;                               //是否在浏览中
17         public void onCreate(Bundle savedInstanceState) {      //重写的 onCreate 方法
18             super.onCreate(savedInstanceState);
19             setContentView(R.layout.main);                     //设置显示的用户界面
20             mySurfaceView = (SurfaceView) findViewById(R.id.surfaceView);
                                                                  //得到 SurfaceView 的引用
21             button1 = (Button) findViewById(R.id.button1);
22             button2 = (Button) findViewById(R.id.button2);     //得到两个按钮的应用
23             mySurfaceHolder = mySurfaceView.getHolder();       //获得 SurfaceHolder
24             mySurfaceHolder.addCallback(this);
25             mySurfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
26             button1.setOnClickListener(new Button.OnClickListener(){ //打开的按钮监听
27                 public void onClick(View arg0) {                 //单击事件
28                     initCamera();                                //调用初始化方法
29                 }
30             });
31             button2.setOnClickListener(new Button.OnClickListener(){ //关闭的按钮监听
32                 public void onClick(View arg0) {
33                     if(myCamera != null && isPreview){           //正在显示时
34                         myCamera.stopPreview();
35                         myCamera.release();                      //释放 myCamera
36                         myCamera = null;
37                         isPreview = false;                       //设成 false
38                     }
39                 }
40             });
41         }
42         public void initCamera(){                                //初始化相机资源
43                 if(!isPreview){
44                     myCamera = Camera.open();                    //打开 Camera
```

```
45                }
46                    if(myCamera != null && !isPreview){
47                      try {
48                              myCamera.setPreviewDisplay(mySurfaceHolder);
49                              myCamera.startPreview();                    //立即运行 Preview
50                          } catch (IOException e) {
51                              e.printStackTrace();                        //打印错误信息
52                          }
53                          isPreview = true;                               //将标记位设置成 true
54                      }
55        }
56      public void surfaceChanged(SurfaceHolder holder, int format, int width,int height) {}
57      public void surfaceCreated(SurfaceHolder holder) {//实现接口中的方法}
58      public void surfaceDestroyed(SurfaceHolder holder) {//实现接口中的方法}
59    }
```

- 第 17～41 行为重写的 onCreate 回调方法。
- 第 20～22 行得到所需要的控件的引用。
- 第 23～25 行得到 SurfaceHolder 并对其进行类型设置。
- 第 26～41 行为按钮添加监听器。
- 第 44 行打开 Camera，所调用的 open 方法的返回值是一个 Camera 的对象。
- 第 48～49 行将 Preiew 画面呈现在 SurfaceView 里。

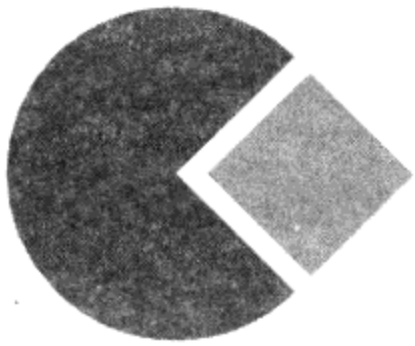
（5）运行该项目，在模拟器中得到的效果如图 2-19 所示。



▲图 2-19　图像采集

2.4 小结

本章对 Android 游戏开发的前台渲染技术进行了详细介绍，通过本章的学习，读者应该能够构建出简单的图形界面。读者可以通过调试和编写本章的示例程序加深对前台渲染技术的理解，从而熟练掌握 Android 中图形的绘制与声音的播放。



第 3 章　Android 游戏开发之交互式通信

在 Android 平台下，应用程序可以方便地调用其他应用程序的功能来实现自己的功能需求，这是 Android 的特性之一。本章将向读者介绍 Android 程序内部或程序之间进行交互式通信的方式，在介绍这些通信方式之前，首先会让读者了解 Android 应用程序的构成框架、了解各个组件的特性以及使用方式。

3.1 Android 应用程序的基本组件

一个 Android 应用程序可以由几个不同的组件构成，本节就来对 Android 应用程序的结构做一个完整的解析。同时，本节还将会对体现各个组件之间关系的 AndroidManifest.xml 配置文件进行简单的介绍。

Android 应用程序的基本组件包括 Activity、Service、BroadcastReceiver 和 ContentProvider 等，不同组件具有不同的特性以及各自的生命周期，下面就来对每个组件做一个简单的介绍。

3.1.1　Activity 组件

Activity 是最常见的一种 Android 组件，每个 Activity 都相当于一个屏幕，其为用户提供了进行交互的可视界面。应用程序可以根据需要包含一个或多个 Activity，这些 Activity 一般都继承自 android.app 包下的 Activity 类，并且这些 Activity 之间的运行是相互独立的。

Activity 的生命周期主要包含 3 个阶段。

- 运行态（running state）

此时 Activity 显示在屏幕前台，并且具有焦点，可以和用户的操作动作进行交互，如向用户提供信息、捕获用户单击按钮的事件并作处理。

- 暂停态（paused state）

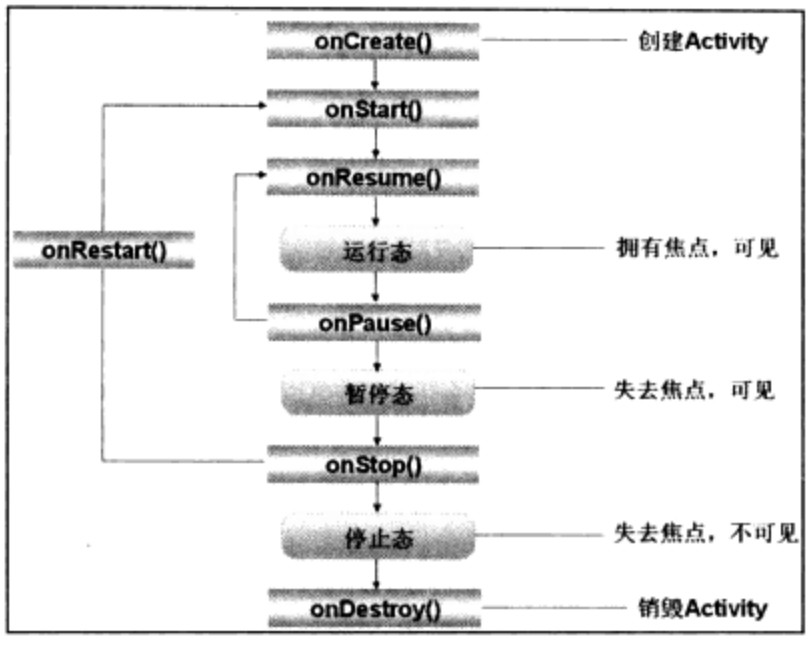
此时 Activity 失去了焦点，并被其他的运行态的 Activity 取代在屏幕前台显示。如果掩盖暂停态 Activity 的运行态，Activity 并不能铺满整个屏幕窗口或者具有透明效果，则该暂停态的 Activity 对用户仍然可见，但是不可以与其进行交互。

暂停态的 Activity 仍然保留其状态和成员等其他信息，当系统的内存非常匮乏时，暂停态的 Activity 会被结束掉以获得更多的资源。

- 停止态（stopped state）

停止态的 Activity 不仅没有焦点，而且是完全不可见的，虽然其也保留状态和成员等信息，停止态的 Activity 会在系统需要的时候被结束。

当 Activity 在不同的状态之间切换时，可以通过重写相应的回调方法来编写状态改变时应该执行的动作，这些状态和相应的回调方法如图 3-1 所示。



▲图 3-1　Activity 的生命周期

Activity 显示的内容可以有两种声明方式，第一种是通过 XML 配置文件来声明，第二种则是将屏幕设置为某一个继承自 View 类的对象。下面分别对这两种方式做一个简单介绍。

- 通过 XML 配置文件声明

Activity 的配置文件位于 res 目录下的 layout 目录中，一个配置文件就相当于一个 View 容器，其中可以添加 Android 平台下的一些内置的 View，如 TextView、ImageView 等，也可以添加继承自 View 类的子类对象，还可以继续添加 View 容器。

布局文件还指定了 View 对象在 View 容器中的排布方式，如线性布局（LinearLayout）、表格布局（TableLayout）等。以下给出了一个简单的通过 XML 配置文件实现 Activity 显示的代码。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_1/src/wyf/wpf 目录下的 Sample_3_1.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.os.Bundle;                                 //引入相关类
4    public class Sample_3_1 extends Activity {
5        @Override
6        public void onCreate(Bundle savedInstanceState) {     //重写 onCreate 方法
7            super.onCreate(savedInstanceState);             //调用父类 onCreate 方法
8            setContentView(R.layout.main);                  //设置所要显示的 XML 配置文件
9        }
10   }
```

代码第 8 行调用 setContentView 方法将屏幕设定为 R.layout.main，其代表位于 res\layout 目录下的 main.xml 文件，由于篇幅有限，该文件代码不予列出，读者可以自行查阅随书光盘。

- 通过 View 的子类对象声明

通过 XML 配置文件将不同的 View 整合到一起非常方便，但是留给开发人员的自主性不够，尤其是当进行游戏编程时，往往 Android 系统中已经存在的 View 无法满足要求，这种情况下一般会通过继承和扩展 View 来开发自己想要的用户界面。下面的例子给出了通过 View 的子类对象来决定 Activity 显示内容的示例，首先给出的是本例中 Activity 的代码。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_2/src/wyf/wpf 目录下的 Sample_3_2.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.os.Bundle;                                 //引入相关类
4    public class Sample_3_2 extends Activity {
5        @Override
6        public void onCreate(Bundle savedInstanceState) {
7            super.onCreate(savedInstanceState);               //调用父类 onCreate 方法
8            MyContentView mcv = new MyContentView(this);      //创建 View 对象
9            setContentView(mcv);                              //设置当前屏幕
10       }
11   }
```

上述代码中使用 MyContentView 作为 Activity 的显示内容，其代码如下。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_2/src/wyf/wpf 目录下的 MyView.java。

```
1    package wyf.wpf;                                                    //声明包
2    import android.content.Context;import android.graphics.Canvas;      //引入相关类
3    import android.graphics.Color;import android.graphics.Paint;        //引入相关类
4    import android.view.View;                                           //引入相关类
5    //继承自 View 的子类
6    public class MyContentView extends View{
7      public MyContentView(Context context) {                           //构造器
8        super(context);
9      }
10     @Override
11     protected void onDraw(Canvas canvas) {                  //重写 View 类绘制时的回调方法
12       Paint paint = new Paint();                            //创建画笔
13       paint.setTextSize(18);                                //设置字体大小
14       paint.setAntiAlias(true);                             //设置抗锯齿
15       paint.setColor(Color.RED);                            //设置字体颜色
16       canvas.drawText("这是通过继承和扩展 View 类来显示的。", 0, 50, paint);
                                                               //绘制字体到屏幕
17     }
18   }
```

尽管一个 Android 应用程序中可以包含多个 Activity，但一般都选择一个作为程序启动后第一个显示在屏幕上的 Activity，其他的 Activity 可以通过当前 Activity 中的 startActivity 方法来启动，这个在之后的小节中将会详细讲解。

3.1.2　Service 组件

与 Activity 不同的是，Service 没有提供与用户进行交互的表示层。Service 是运行在后台的一种 Android 组件，当应用程序需要进行某种不需要前台显示的计算或数据处理时，就可以启动一个 Service 来完成，每个 Service 都继承自 android.app 包下的 Service 类。

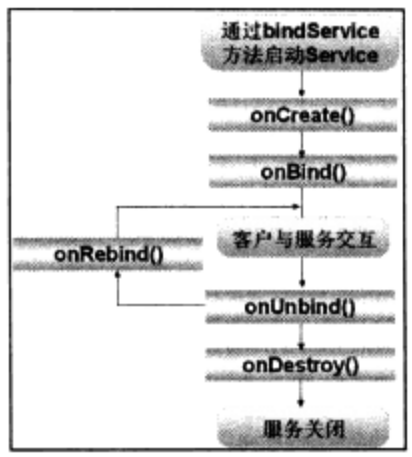
Service 一般由 Activity 或其他 Context 对象来启动，当启动 Service 之后，该 Service 将会在后台运行，即使启动这个 Service 的 Activity 或其他组件的生命周期已经结束，Service 仍然会继续运

行，直到自己的生命周期结束为止。每个 Service 都应该在 AndroidManifest.xml 中进行声明。Service 的启动方式有两种，对应的生命周期也各不相同。

- 通过 startService 方法启动。当系统调用 startService 方法时，如果该 Service 还未启动，则依次调用其 onCreate 方法和 onStart 方法来启动。当其他 Context 对象调用 stopService 方法、Service 调用自身的 stopSelf 或 stopService 方法时才会停止 Service 的执行。
- 通过 bindService 方法启动。当系统调用 bindService 方法时，如果该 Service 未启动，则会调用其 onCreate 方法完成初始化工作，然后会将该 Service 和 Context 对象（如 Activity）进行绑定，当被绑定的 Context 对象被销毁时，与之绑在一起的 Service 也会停止运行。

通过不同的方式启动 Service，其生命周期也不尽相同，两种启动 Service 的方式以及相应阶段会涉及的回调方法如图 3-2 和图 3-3 所示。

▲图 3-2　通过 startService 方法启动 Service



▲图 3-3　通过 bindService 方法启动服务

需要注意的是，尽管存在两种方式启动 Service，但是无论 Service 是通过什么方式启动的，都可以将其与 Context 对象进行绑定。在 Android 平台下启动服务需要涉及 Intent 等方面的知识，本书将在稍后的小节中做详细的介绍。

3.1.3　Broadcast Receiver 组件

Broadcast Receiver 同 Service 一样，并不提供与用户交互的表示层，其是一种负责接收广播消息并对消息做出反应的组件。在 Android 的系统中就存在许多这样的广播，比如电池电量过低或信号过低时，系统就会发出广播进行通知。

应用程序如果需要响应某一个广播消息，应该注册对应的 BroadcastReceiver 对象，该对象继承自 BroadcastReceiver 类，该类位于 android.content 包下。这样一来，当系统或另外的应用程序发出特定广播时，该应用程序就可以接收并做出回应，如启动 Activity 等。

1. BroadcastReceiver 发布广播的方式

发布一个广播比较容易，在需要的地方创建一个 Intent 对象，将信息的内容和用于过滤的信息封装起来，通过调用 Context.sendBroadcast 方法、Context.sendOrderedBroadcast 方法或者 Context.sendStickyBroadcast 方法将该 Intent 对象广播出去，3 种发布广播方式的区别如下。

- 通常，使用 sendBroadcast 或 sendStickyBroadcast 发送出去的 Intent，所有满足条件的

BroadcastReceiver 都会执行其 onReceive 方法。但若有多个满足条件的 BroadcastReceiver，其执行 onReceive 方法的顺序是没有保证的。而通过 sendOrderedBroadcast 方法发送出去的 Intent，会根据 BroadcastReceiver 注册时 IntentFilter 设置的优先级的顺序来执行 onReceive 方法，相同优先级的 BroadcastReceiver 执行 onReceive 方法的顺序是没有保证的。

- sendStickyBroadcast 主要的不同是，Intent 在发送后会一直存在，并且在以后调用 registerReceiver 注册相匹配的 Receiver 时会把这个 Intent 对象直接返回给新注册的 Receiver。

2. BroadcastReceiver 接收广播的方式

发布的广播实体是 Intent，那么接收广播的时候就需要通过 IntentFilter 对象来进行过滤。Broadcast Receiver 的生命周期比较简单，只有一个回调方法——onReceive，该方法在应用程序接收到发给自己广播时候调用，所以 Broadcast Receiver 的使用方法也相对简单，只需要对 onReceive 方法进行合理重写，在适当的地方注册该 Broadcast Receiver 即可。

注册 BroadcastReceiver 对象的方式有以下两种。

- 在 AndroidManifest.xml 文件中声明。注册信息包含在<receiver></receiver>标签中，并在<intent-filter>标签内设定过滤规则。
- 在代码中创建并设置 IntentFilter 对象。该 IntentFilter 对象包含了对广播的过滤规则，然后在需要的地方调用 Context.registerReceiver 方法和 Context.unregisterReceiver 方法进行注册和取消注册，如果采用这种方式注册，当 Context 对象被销毁时，该 BroadcastReceiver 也就不复存在了。

提示　发布广播和使用 BroadcastReceiver 对象注册监听广播的内容涉及了后面的 Intent 对象的知识，将在稍后的小节进行详细探讨。

3.1.4　Content Provider 组件

Content Provider 和其他的应用程序组件有很大的不同，Content Provider 主要用于不同的应用程序之间进行数据共享。在 Android 平台下，每个应用程序都有独立的内存空间，如果某一个应用程序需要使用其他应用程序的数据，就必须采用 ContentProvider 对象。

每个 ContentProvider 都继承自 android.content 包下的 ContentProvider 类，其功能就是提供自己的数据给外部应用程序使用，提供的数据可以存储为 Android 文件、SQLite 数据库文件（将会在后面的章节进行介绍）或其他合法的格式。

ContentProvider 提供数据及访问数据的接口，真正访问数据的是 ContentResolver 对象，该对象可以与 ContentProvider 对象进行通信，以达到共享数据的目的。

提示　ContentProvider 组件所涉及的知识超过了本章涵盖的范围，将会在随后的章节进行详细介绍。

3.1.5　AndroidManifest.xml 文件简介

前面的小节介绍了 Android 平台下应用程序的基本组件，这些组件要想被应用程序使用，需要

在 AndroidManifest.xml 配置文件中进行声明。但 AndroidManifest.xml 文件的作用远不止这些，每个 Android 应用程序都必须包含一个 AndroidManifest.xml 配置文件，而且文件名称不可改变。

除了 Broadcast Receiver 组件既可以在 AndroidManifest.xml 文件中声明，也可以在代码中直接创建之外，其他的应用程序组件必须在 AndroidManifest.xml 文件中进行声明，否则系统将无法使用该组件。AndroidManifest.xml 文件的主要内容包括以下几点。

- 声明应用程序的 Java 包名，该包名将作为该应用程序的惟一标识符。
- 描述应用程序所包含的组件，如 Activity 等。除了描述实现某种组件的类的名称外，还需要声明该组件对于 Intent 对象的过滤规则，即告知系统在何种状态下该组件可以被启动。
- 指出应用程序组件运行在哪个进程，默认情况下所有的组件都运行在主进程中，如果需要让其运行在别的线程，需要在 AndroidManifest.xml 中设置。
- 声明应用程序必须具有的用来访问受保护的 API 或与其他应用程序交互的权限。
- 声明其他应用程序必须具有的用来访问自己组件的权限。
- 列出该应用程序中的 Instrumentation 对象，Instrumentation 的用途是对应用程序的运行进行监控，其只在应用程序的开发过程中起作用，在程序发布前会被移除。
- 声明应用程序所要求的最低 Android API 版本。
- 声明应用程序需要链接到的默认 Android 类库之外的库。

以下代码给出了一个普通项目的 AndroidManifest.xml 文件的结构。由于本书篇幅有限，故只将 AndroidManifest.xml 文件的代码列出。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_3 目录下的 AndroidManifest.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3         package="wyf.wpf"
4         android:versionCode="1"
5         android:versionName="1.0">              <!-- 标记,记录应用程序的包及版本等信息 -->
6       <application android:icon="@drawable/icon" android:label="@string/app_name">
7          <activity android:name=".Sample_3_3"
8                  android:label="@string/app_name"><!-- 声明 Activity 组件 -->
9             <intent-filter>
10               <action android:name="android.intent.action.MAIN" />
11               <category android:name="android.intent.category.LAUNCHER" />
12            </intent-filter>                     <!-- 为 Activity 设置 IntentFilter -->
13         </activity>
14         <service android:name=".MyService"
15              android:process=":remote" >
16         </service>                              <!-- 声明 Service 组件 -->
17      </application>
18      <uses-sdk android:minSdkVersion="7" />   <!-- 指定应用程序运行的最低 SDK 版本 -->
19      <uses-permission android:name="android.permission.INTERNET" />
                                                   <!-- 指定应用程序的权限 -->
20   </manifest>
```

从上边的代码可以看出 AndroidManifest.xml 文件的基本结构，所有的信息写在根标记<manifest>中，<manifest>标记所包含的属性及说明如表 3-1 所示。

在根标记<manifest>中必须包含<application>标记，所有的组件如 Activity 等均在<application>标记中声明。除了<application>标记外，<manifest>标记中还可以包含<uses-permission>等标记。

<manifest>标记中所包含的子标记及其属性和说明如表 3-2 所示。

表 3-1　AndroidManifest.xml 文件主要标记及其说明

标　记	属　性	说　明
<manifest>	package	应用程序的全称包名
	versionCode	内部版本号，值越大版本越新
	versionName	提供给用户的版本号
	sharedUserId	与其他应用程序共享的 Linux 用户 ID，默认每个应用程序拥有惟一的 ID，如果两个应用程序的 ID 相同，则其可以相互访问彼此的数据
	sharedUserLabel	sharedUserId 的可读形式，只有在 sharedUserId 被设置的情况下此属性才有效

表 3-2　<manifest>的子标记及其属性说明

标　记	属　性	说　明
<application>	icon	应用程序的图标，其值必须为 drawable 资源的引用
	label	应用程序的可读名称，其值可以为 string 资源的引用，也可以为原始字符串
	theme	应用程序内部所有 Activity 组件的主题风格，其值为 style 资源的引用
	persistent	是否应用程序应该一直运行，默认为 false，一般对此不设置，该模式只用来描述系统级应用程序
	process	应用程序中所有组件运行的进程名，每个组件可以设置自己的 process 属性来覆盖掉此属性。默认情况下，应用程序在运行第一个组件时创建一个进程，之后其他的组件均运行在此进程中。 可以设置该属性使两个应用程序的组件运行在同一个进程中，这两个应用程序必须具有相同的用户 ID 和 certificate
	permission	应用程序的调用者与应用程序交互所必须具有的权限
<uses -permission>	name	保证应用程序正常运行所必须授予给的权限，该权限在应用程序安装时被授予，并不是运行时
<uses-sdk>	minSdkVersion	应用程序运行的最低 API 版本，默认值为“1”，即与所有的 API 版本兼容
	targetSdkVersion	指明应用程序的目标版本
	maxSdkVersion	应用程序运行的最高 API 版本，如果系统的版本比该属性值高，应用程序将不会被安装

<application>组件中包含了应用程序包含的各种组件的标记，如<activity>、<service>等，这些组件的标记中很多属性值和<application>标记中的属性名称相同，如果进行了设置将会覆盖掉<application>中的同名属性值，如果未设置则取<application>中的同名属性值。<application>标记包含的子标记及其属性说明如表 3-3 所示。

表 3-3　<application>标记所包含的子标记及属性

标　记	属　性	说　明
<activity>、<service>、<receiver>、<provider>共有的属性	name	实现组件类的子类名称，其值可以为子类的全称类名，也可以以“.”开头省略掉应用程序的包名，后面直接加上子类的类名

续表

标　记	属　性	说　明
<activity>、<service>、<receiver>、<provider>共有的属性	process	组件应该运行在哪个进程中，一般情况下不设置时，所有的组件均运行在同一个进程中，如果该值以“:”开头，则会为该组件创建一个私有的新进程；若以小写字母开头将是一个全局的新进程
	permission	启动组件所必须具有的权限，如果该属性未设置，则以<application>标记中的permission 属性为组件的权限
<activity>	screenOrientation	屏幕方向，其可以取的值为：unspecified、landscape、portrait、user、behind、sensor、nosensor
<provider>	readPermission	应用程序的调用者查询 content provider 中数据所必须具有的权限
	writePermission	应用程序的调用者修改 content provider 中数据所必须具有的权限
<uses-library>	name	指明应用程序会链接到的除默认 Android 类库之外的库

在<activity>、<service>和<receiver>等标记中，还可以包含<intent-filter>标记，该标记指明了组件的 intent 过滤规则，<intent-filter>标记的属性及其说明如表 3-4 所示。

表 3-4　　**<intent-filter>标记的属性及说明**

标　记	属　性	说　明
<intent-filter>	icon	代表父组件的图标，必须为 drawable 资源的引用
	label	代表父组件的可读名称，可以为 string 资源的引用，也可以是原始字符串
	priority	在处理 Intent 时具有的优先级，对 Activity 和 broadcast receiver 有效。该属性值越高，优先级越高。 当一个 Intent 可以被多个优先级不同的 Activity 响应时，Android 只会将优先级最高的那些 activity 列入考虑范围。也就是若优先级最高的有多个，则列出来让用户选择，而优先级低的不会列出。 当一个 Intent 可以被多个 broadcast receiver 响应时，将会按照优先级从高到低的顺序执行 onReceive 方法，而相同优先级的 broadcast receiver 执行 onReceive 方法的顺序则没有保证

<intent-filter>标记中的子标记有<action>、<category>和<data>，其中<action>标记是必须包含的，并且可以为多个。这些子标记的属性值如表 3-5 所示。

表 3-5　　**<intent-filter>标记包含的子标记及其属性说明**

标　记	属　性	说　明
<action>	name	为 intent filter 添加一个 action，其值可以为 Intent 类的系统常量，如果为自定义的 action，应该在 action 前加上包名作为前缀
<category>	name	为 intent filter 添加一个 category，其值可以为 Intent 类的系统常量，如果为自定义的 category，应该在 category 前加上包名作为前缀
<data>	scheme	URI 中的 scheme 部分，必须至少设置一个 scheme 属性，否则其他的 URI 属性将会无效
	host	URI 中的 host 部分，必须为小写字母，该属性需要设置了 scheme 属性才有效
	port	URI 中的 port 部分，该属性需要设置了 scheme 和 host 属性才有效

以上为 AndroidManifest.xml 配置文件中常见的一些标记及其属性值说明，其中<intent-filter>

标记中涉及的一些知识将会在随后的小节详细介绍。

3.2 应用程序的内部通信

在 Android 应用程序中，内部通信简单来讲是指主线程和自己开发的子线程之间的通信。在前面的小节中已经提到，在 Android 应用程序运行时，默认情况下会为第一个启动的组件创建一个进程，之后启动的组件都运行在这个进程中。

当为应用程序创建了一个进程后，一个主线程将会被创建。这个主线程主要负责维护组件对象和应用程序创建的所有窗口，如果应用程序中创建了自己的线程，这些线程将无法对主线程控制的内容进行修改，此时就需要使用 Handler 来同主线程进行交互，本节就来简单介绍 Handler 类及其基本的用法。

3.2.1　消息的处理者——Handler 类简介

Handler 类主要用于应用程序的主线程同用户自己创建的线程进行通信，应用程序在主线程中维护一个消息队列。Handler 机制使得线程间的通信通过 Message 和 Runnable 对象来传递和处理。

每个 Handler 对象都与一个线程及其消息队列相关联。当创建一个 Handler 对象时，其便与创建这个 Handler 对象的线程的消息队列绑定，之后该 Handler 对象将会向消息队列传递 message 或 runnable 并处理执行队列中的元素。

Handler 最主要的用法就是安排 Message 和 Runnable 对象使其在未来的某个时刻被处理或运行。

1. 传递消息对象

使用 Handler 传递消息时将消息内容封装到一个 Message 对象中，Message 类中包含了消息的描述和任何形式的可以被 Handler 发送的数据对象。一个 Message 对象中主要的字段如表 3-6 所示。

表 3-6　Message 对象中主要字段及说明

字　段	说　明
arg1	int 类型，当传递的消息只包含整数时，可以填充该字段以降低成本。该字段可以通过成员方法 setData 和 getData 方法访问或修改
arg2	
obj	Object 类型，可以为任意类型
what	int 类型，由用户定义的消息类型码，接收方可以根据该字段值来确定收到的消息是关于什么的

提示

getData 和 setData 返回和接收的都是 Bundle 对象，该对象是以 String 为键、以任意可封装的类型为值的 map。

Handler 发出消息时，既可以指定消息到达后立即被处理，也可以指定其经过特定的时间间隔后被处理，还可以指定一个绝对时间让消息在那之前被处理。不同的发送消息的方法如表 3-7 所示。

表 3-7　　不同的发送消息的方法及说明

方　法　名	说　　明
sendEmptyMessage(int what)	发送一个空的消息，只指定 Message 的 what 字段
sendMessage(Message message)	发送一个消息对象
sendMessageAtTime(Message message,long time)	在指定时间之前发送一个消息
sendMessageDelayed(Message message,long time)	在指定时间间隔之后发送一个消息

无论消息以何种方式发出，接受并处理消息的方法都是 handleMessage 方法，该方法接受的参数为一个 Message 对象，在其中可以根据程序需要对不同的消息进行不同的处理。

提示　虽然 Message 类提供了公有的构造方法，但是最好的获得一个 Message 对象的途径是调用 Message 类的静态方法 obtain 或者是 Handler 类的一系列方法，如 obtainMessage，这样创建的 Message 对象是可复用的。

2. 传递 Runnable 对象

传递 Runnable 对象与传递 Message 对象类似，Runnable 对象为实现了 Runnable 接口的对象，Handler 在传递 Runnable 对象时也可以设置其经过指定的时间间隔或在指定的绝对时间之前被处理。由于篇幅有限，本书在此不再赘述，有兴趣的读者可以自行查阅相关书籍。

3.2.2　使用 Handler 进行内部通信

前面的小节对 Handler 类进行了一个简单的介绍，本小节就来举一个简单的例子说明 Handler 的具体用法。在本例中，用户通过单击屏幕上的按钮，启动一个线程与主线程通过 Handler 进行通信以更新显示数据。运行效果如图 3-4 所示。

本应用程序中只包含 Activity 一个组件，接收用户单击按钮事件和发送 Handler 等功能都在 Activity 中实现。整个程序分为如下 3 个步骤来完成。

▲图 3-4　Sample_3_4 运行效果图

（1）Activity 的主要代码。程序运行后的屏幕中包含一个 Button 和一个 TextView 对象，单击按钮后会在新启动的线程中向主线程发 Handler 消息，主线程收到消息后会根据消息的内容修改 TextView 的显示内容。Activity 的主要代码如下。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_4/src/wyf/wpf 目录下的 Sample_3_4.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.os.Bundle;                                 //引入相关类
4    import android.os.Handler;                                //引入相关类
5    import android.os.Message;                                //引入相关类
6    import android.view.View;                                 //引入相关类
7    import android.widget.Button;                             //引入相关类
8    import android.widget.TextView;                           //引入相关类
9    //继承自 Activity 的子类
10   public class Sample_3_4 extends Activity {
11     public static final int UPDATE_DATA = 0;                //常量，代表更新数据
12     public static final int UPDATE_COMPLETED = 1;           //常量，代表更新数据
13     TextView tv;                                            //TextView 对象的引用
14     Button btnStart;                                        //Button 对象的引用
15     ……                    //此处省略创建 Handler 对象的相关代码，将在后面的步骤中补全
16       @Override
17       public void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.main);              //设置当前屏幕为 R.layout.main 布局文件
20         tv = (TextView)findViewById(R.id.tv);       //获得屏幕中 TextView 对象引用
21         btnStart = (Button)findViewById(R.id.btnStart);
                                                       //获得屏幕中 Button 对象引用
22         ……          //此处省略为按钮添加单击事件监听器的相关代码，将在后面的步骤中补全
23       }
24   }
```

- 代码第 11～12 行定义了用来表示消息类别的静态常量，这样代码的可读性和可维护性都好，建议读者在开发的过程中也采用这种写法。
- 代码第 17～23 行为重写 Activity 的 onCreate 方法，该方法在创建 Activity 组件时被调用。
- 代码第 20～21 行通过调用 findViewById 方法获取布局文件中 Button 和 TextView 对象的引用，以便在后续的代码中修改其属性值。

（2）采用匿名内部类的方式声明并创建一个继承自 Handler 类的子类对象，并在该内部类中重写 handleMessage 方法，下述创建 Handler 对象的代码来自于步骤（1）代码中的第 15 行省略的部分。

```
1    Handler myHandler = new Handler(){
2      @Override
3      public void handleMessage(Message msg) {                  //重写处理消息方法
4         switch(msg.what){                                      //判断消息类别
5         case UPDATE_DATA:                                      //消息为更新的数据
6             tv.setText("正在更新来自线程的数据:"+msg.arg1+"%...");
                                                                 //更新 TextView 显示的内容
7             break;
8         case UPDATE_COMPLETED:                                 //消息为更新完毕
9             tv.setText("已完成来自线程的更新数据！");          //改变 TextView 显示的内容
10            break;
11        }
12     }
13   };
```

说明　上述代码创建了一个与主线程绑定的 Handler 对象，其中重写了 handleMessage (Message msg)方法用来接收和处理消息，主要是采用判断 Message 对象的 what 字段来确定消息类别。

（3）为按钮 btnStart 添加单击事件监听器。在步骤（1）中代码的第 22 行省略的部分添加如下代码，为按钮添加单击事件监听器。

```
1    btnStart.setOnClickListener(new View.OnClickListener() {
                                                          //为 Button 添加单击事件监听器
2         @Override
3         public void onClick(View v) {
4              new Thread(){                              //启动一个新线程
5                   public void run(){
6                        for(int i=0;i<100;i++){
7                             try{                        //睡眠一段时间
8                                  Thread.sleep(150);
9                             }
10                            catch(Exception e){
11                                 e.printStackTrace();
12                            }
13                            Message m = myHandler.obtainMessage();  //创建 Message 对象
14                            m.what = UPDATE_DATA;                   //为 what 字段赋值
15                            m.arg1=i+1;                             //为 arg1 字段赋值
16                            myHandler.sendMessage(m);               //发出 Message 对象
17                       }
18                       myHandler.sendEmptyMessage(UPDATE_COMPLETED); //发更新完毕消息
19                  }
20             }.start();
21        }
22   });
```

- 代码第 3～21 行重写了 onClick(View view)方法，该方法主要的工作是在单击事件发生后启动一个线程，该线程的主要工作是每隔一段时间向主线程发送 Handler 消息，主线程收到消息后会根据其内容修改 TextView 的值。
- 代码第 16 行和第 18 行都是发送 Handler 消息的方法，当所发送的消息内容只需要 what 属性就可以时，可以采用 sendEmptyMessage 方法。

3.3 应用程序组件之间的通信

在 3.1 小节介绍了 Android 应用程序的基本组件，这些基本组件除了 Content Provider 之外，几乎全部都是依靠 Intent 对象来激活和通信的。本节就来介绍 Intent 类的相关知识，并通过示例来说明 Intent 的一般用法。

3.3.1 Intent 类简介

Intent 类的对象是组件间通信的载体，组件之间进行通信就是一个个 Intent 对象在不断地传递。Intent 对象主要作用于运行在相同或不同应用程序的 Activity、Service 和 Broadcast Receiver 组件之间，对于这 3 种组件，其作用机制也不相同。

- 对于 Activity 组件，Intent 主要通过调用 Context.startActivity、Context.startActivityForResult 等方法实现传递，其结果是启动一个新的 Activity 或者使当前的 Activity 开始新的任务。
- 对于 Service 组件，Intent 主要通过调用 Context.startService 和 Context.bindService 方法实现传递，其作用结果是初始化并启动一个服务或者绑定服务到 Context 对象。

- 对于 Broadcast Receiver 组件，Intent 主要通过 sendBroadcast 等一系列发送广播的方法实现传递，其作用结果是将 Intent 组件以广播的形式发出以便合适的组件接收。

一个 Intent 对象就是一组信息，其包含接收 Intent 组件所关心的信息（如 Action 和 Data）和 Android 系统关心的信息（如 Category 等），一般来讲，一个 Intent 对象包含如下内容。

1. Component Name 部分

组件名称指明了未来要处理 Intent 的组件，组件名称封装在一个 ComponentName 对象中，该对象用于惟一标识一个应用程序组件，如 Activity、Service、Content Provider 等。ComponentName 类包含两个 String 成员，分别代表组件的全称类名和包名，包名必须和 AndroidManifest.xml 中 <application>标记中的对应信息一致。

对于 Intent 对象来说，组件名称不是必须的，如果添加了组件名称则该 Intent 为“显式 Intent”，这样 Intent 在传递的时候会直接根据 ComponentName 对象的信息去寻找目标组件。如果不设置组件名称，则为“隐式 Intent”，Android 会根据 Intent 中的其他信息来确定响应该 Intent 的组件是哪个。

2. Action 部分

Action 为一个字符串对象，其描述了该 Intent 会触发的动作。Android 系统中已经预先定义好了一些表征 Action 的常量，如 ACTION_CALL、ACTION_MAIN 等，同时，开发人员也可以自己定义 Intent 的动作描述，一般来讲，自己定义的 Action 字符串应该以应用程序的包名为前缀，如可以定义一个 Action 为“wyf.wpf.StartService”。

因为 Action 很大程度上决定了一个 Intent 的内容（主要是 Data 和 Extras 部分），所以定义自己的 Action 时应该做到见名知义，同时，如果应用程序比较复杂，应该为其定义一个整体的 Action 协议，使所有的 Action 集中管理。

3. Data 部分

Data 描述 Intent 的动作所操作到的数据的 URI 及类型，不同的 Action 对应不同的操作数据，比如 Action 为 ACTION_VIEW 的 Intent 的 Data 应该是“http:”格式的 URI。当为组件进行 Intent 的匹配检查时，正确设置 Data 的 URI 资源和数据类型很重要。

4. Category 部分

Category 为字符串对象，其包含了可以处理 Intent 的组件的类别信息，Intent 中可以包含任意个 Category。同 Action 一样，Android 系统预先定义了一些 Category 常量，但是不可以自行定义 Category。

调用方法 addCategory 用来为 Intent 添加一个 Category，方法 removeCategory 用来移除一个 Category，getCategories 方法返回已定义的 Category。

5. Extras 部分

Extras 是一组键值对，其包含需要传递给目标组件并由其处理的一些额外信息。

6. Flags 部分

一些有关系统如何启动组件的标志位，所有的标志位都已在 Android 系统中预先定义。

3.3.2 应用程序组件——IntentFilter 类简介

当 Intent 在组件之间进行传递时，组件如果需要告知 Android 系统自己能够响应和处理哪些 Intent，就需要使用 IntentFilter 对象。顾名思义，IntentFilter 对象负责过滤掉组件无法响应和处理的 Intent，只将自己关心的 Intent 接收进来进行处理。

IntentFilter 实行“白名单”管理，即只列出组件能接收的 Intent，IntentFilter 只会过滤隐式 Intent，显式的 Intent 会被直接传递到目标组件，一个隐式的 Intent 只有通过了组件的某一个 IntentFilter 的过滤，才可以被组件接收并处理。

诸如 Activity、Service、Broadcast Receiver 这些组件可以有一个或多个 IntentFilter，每个 IntentFilter 相互独立，只需要通过一个即可。每个 IntentFilter 都是 android.content 包下的 IntentFilter 类的对象，除了用于过滤广播的 IntentFilter 可以在代码中动态创建外，其他组件的 IntentFilter 必须在 AndroidManifest.xml 文件中进行声明。

IntentFilter 中具有与 Intent 对应的用于过滤 Action、Data 和 Category 的字段，一个 Intent 对象要想被一个组件处理，必须通过这 3 个环节的检查。

1. 检查 Action

尽管一个 Intent 只可以设置一种 Action，一个 IntentFilter 却可以持有一个或多个 Action 用于过滤，到达的 Intent 对象只需要匹配其中一个 Action 即可。但是 IntentFilter 的 Action 部分不可以为空，如果 Action 部分为空则会阻塞掉所有的 Intent。相反，如果 Intent 的 Action 字段未设置，其将通过所有的 IntentFilter 的 Action 检查。

2. 检查 Data

同 Action 一样，IntentFilter 中的 Data 部分也是可以为一个或多个，而且可以没有。每个 Data 包含的内容为 URI 和数据类型，进行 Data 检查时主要也是对这两点进行比较，比较规则如下。

- 如果 Intent 对象没有设置 Data，只有 IntentFilter 也未作设置时才可通过检查。
- 如果 Intent 对象只设置了 URI 而没有指定数据类型，只有当其匹配 IntentFilter 的 URI，并且 IntentFilter 也没有设置数据类型时该 Intent 对象才可以通过检查。
- 如果 Intent 对象只指定了数据类型而没有设置 URI，只有当其匹配 IntentFilter 的数据类型并且也没有设置 URI 时该 Intent 对象才可以通过检查。
- 如果 Intent 对象既包含 URI 又包含数据类型，只有当其数据类型匹配 IntentFilter 中的数据类型并且通过了 URI 检查时该 Intent 对象才可以通过检查。

3. 检查 Category

IntentFilter 中可以设置多个 Category，检查 Category 时，只有当 Intent 对象中所有的 Category

都匹配 IntentFilter 中的 Category 时该 Intent 对象才可以通过检查，并且 IntentFilter 中的 Category 可以比 Intent 中的 Category 多，但是必须都包含 Intent 对象中所有的 Category。如果一个 Intent 没有设置 Category，则其将通过所有 IntentFilter 的 Category 检查。

IntentFilter 既可以在 AndroidManifest.xml 中声明，也可以在代码中动态创建。如果是在 AndroidManifest.xml 中声明 IntentFilter，需要使用<intent-filter>标记。该标记包含<action>、<data>、<category>子标记，每个子标记中包含的属性以及对应的方法如表 3-8 所示。

表 3-8　<intent-filter>子标记及其属性说明

子 标 记	属　性	说　明	对 应 方 法
<action>	name	字符串，系统预定义或由自己定义	addAction(String)
<category>	name	字符串	addCategory(String)
<data>	mimeType	数据类型，可使用通配符	addDataType(String)
	scheme	这 4 个部分共同组成了 URI，其格式为：scheme://host:port/path，例如：content://wyf.wpf:200/files	addDataScheme(String) addDataAuthority(String) addDataPath(String)
	host		
	port		
	path		

如果一个到来的 Intent 对象通过了不止一个组件（如 Activity、Service 等）的 IntentFilter 的检查，那么系统将会弹出提示，让用户选择激活哪个组件。

3.3.3　示例 1：与 Android 系统组件通信

前面的小节简单介绍了 Intent 类和 IntentFilter 类的相关知识，本小节将通过一个小例子来说明其用法，本例使用 Intent 和 IntentFilter 与 Android 系统预定义组件拨号程序进行通信。首先，通过 Intent 调用系统的拨号程序，程序的主要代码如下。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_5/src/wyf/wpf 目录下的 Sample_3_5.java。

```
1    package wyf.wpf;                                          //声明包
2    import android.app.Activity;                              //引入相关类
3    import android.content.Intent;                            //引入相关类
4    import android.os.Bundle;                                 //引入相关类
5    import android.view.View;                                 //引入相关类
6    import android.widget.Button;                             //引入相关类
7    //继承自 Activity 的子类
8    public class Sample_3_5 extends Activity {
9      Button btnDial;
10       @Override
11       public void onCreate(Bundle savedInstanceState) {     //重写 onCreate 方法
12           super.onCreate(savedInstanceState);
13           setContentView(R.layout.main);                    //设置屏幕显示内容
14           btnDial = (Button)this.findViewById(R.id.btDial); //获得屏幕上按钮对象引用
15           btnDial.setOnClickListener(new Button.OnClickListener(){
                                                               //为按钮添加单击事件的监听器
16               @Override
17               public void onClick(View v) {                  //重写 onClick 方法
18                   Intent myIntent = new Intent(Intent.ACTION_DIAL);
                                                               //创建 Intent 对象
```

```
19                  Sample_3_5.this.startActivity(myIntent);
                                                        //启动 Android 内置的拨号程序
20              }
21          });
22      }
23  }
```

- 代码第 14 行获取屏幕上按钮对象的引用。
- 代码第 15～21 行为按钮添加单击事件的监听器，其中第 19～22 行为重写的 onClick 方法，在其中创建了一个 Intent 并设定其 Action 为 Android 的拨号程序。

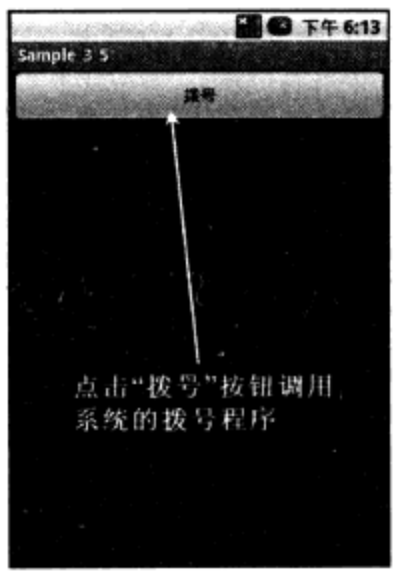
以上代码运行后单击按钮前后的屏幕如图 3-5 和图 3-6 所示。

下面继续完善这个程序，在 AndroidManifest.xml 中的<activity>标记中添加如下代码。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_5 目录下的 AndroidManifest.xml。

```
1   <activity android:name=".Sample_3_5" android:label="@string/app_name">
2       ……<!--  省略不必要的代码，读者可以自行查阅随书光盘  -->
3       <intent-filter>                                <!-- 声明一个 IntentFilter -->
4           <action android:name="android.intent.action.CALL_BUTTON" />
                                                       <!-- 设定 Action -->
5           <category android:name="android.intent.category.DEFAULT" />
                                                       <!-- 设定 Category -->
6       </intent-filter>
7   </activity>
```

这段代码的作用是为 Activity 添加一个 IntetnFilter，即该 Activity 将会对系统按下拨号键进行响应。当在待机状态下按下拨号键时，系统就会提示是启动 Android 系统自带的拨号程序，还是启动自己的应用程序，如图 3-7 所示。



▲图 3-5　程序运行效果图

▲图 3-6　单击按钮后调用系统拨号程序

▲图 3-7　按下拨号键显示的对话框

3.3.4　示例 2：应用程序组件间通信示例 Activity 部分的开发

前面的例子主要的功能是通过 Intent 来启动系统的拨号程序，以及在 AndroidManifest.xml 中设置应用程序的 IntentFilter 来使应用程序可以响应手机拨号键按下的行为。下面将会是一个应用程序内部组件之间通过 Intent 和 Broadcast Receiver 组件通信的例子。

本例中在 Activity 组件中通过单击按钮启动一个 Service，Service 将会启动一个线程，该线程

的工作是定时产生一个随机数，并将其封装到 Intent 对象中传递给 Activity，Activity 接收到 Intent 后提取其中的信息将其显示到 TextView 控件中。在服务运行的时候，可以单击 Activity 上的停止按钮来停止服务。

首先来开发应用程序的 Activity 部分。Activity 的开发通过以下 3 个步骤来完成。

（1）开发 Activity 的主要代码。在 Activity 中定义引用指向屏幕上的 Button 等控件，并为 Button 添加单击事件的监听器，Activity 的主要代码如下。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_6/src/wyf/wpf 目录下的 Sample_3_6.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.content.BroadcastReceiver;                 //引入相关类
4    import android.content.Context;                           //引入相关类
5    import android.content.Intent;                            //引入相关类
6    import android.content.IntentFilter;                      //引入相关类
7    import android.os.Bundle;                                 //引入相关类
8    import android.view.View;                                 //引入相关类
9    import android.view.View.OnClickListener;                 //引入相关类
10   import android.widget.Button;                             //引入相关类
11   import android.widget.TextView;                           //引入相关类
12   //继承自 Activity 的子类
13   public class Sample_3_6 extends Activity {
14     public static final int CMD_STOP_SERVICE = 0;
15     Button btnStart;                                        //开始服务 Button 对象应用
16     Button btnStop;                                         //停止服务 Button 对象应用
17     TextView tv;                                            //TextView 对象应用
18     DataReceiver dataReceiver;                              //BroadcastReceiver 对象
19     @Override
20       public void onCreate(Bundle savedInstanceState) {     //重写 onCreate 方法
21           super.onCreate(savedInstanceState);
22           setContentView(R.layout.main);                    //设置显示的屏幕
23           btnStart = (Button)findViewById(R.id.btnStart);
24           btnStop = (Button)findViewById(R.id.btnStop);
25           tv = (TextView)findViewById(R.id.tv);
26             btnStart.setOnClickListener(new OnClickListener(){
                                                               //为按钮添加单击事件监听
27               @Override
28               public void onClick(View v) {                  //重写 onClick 方法
29                     Intent myIntent = new Intent(Sample_3_6.this, wyf.wpf.MyService.class);
30                     Sample_3_6.this.startService(myIntent);  //发送 Intent 启动 Service
31               }
32           });
33           btnStop.setOnClickListener(new OnClickListener() { //为按钮添加单击事件监听
34               @Override
35               public void onClick(View v) {                  //重写 onClick 方法
36                     Intent myIntent = new Intent();          //创建 Intent 对象
37                     myIntent.setAction("wyf.wpf.MyService");
38                     myIntent.putExtra("cmd", CMD_STOP_SERVICE);
39                     sendBroadcast(myIntent);                 //发送广播
40               }
41           });
42       }
43     ……//此处省略重写的 Activity 的 onStart 和 onStop 方法，将在后面的步骤中补全
44     ……//此处省略声明 BroadcastReceiver 子类的代码，将在后面的步骤中补全
45   }
```

- 上述代码声明了 Activity 中按钮等控件的引用，同时重写了 onCreate 方法为控件对象的引

用赋值并为按钮的单击事件添加监听器。

- 代码的第 29 行和第 36～38 行分别创建了用于启动和停止 Service 的 Intent 对象，并通过 startService 方法和 sendBroadcast 方法发出去。

（2）服务启动后也会向 Activity 发 Intent，所以 Activity 也必须注册一个 Broadcast Receiver 组件用于接收 Intent，注册之前需要编写实现了 BroadcastReceiver 的子类，下面代码即是来自于步骤（1）中代码第 44 行省略的部分。

```
1    private class DataReceiver extends BroadcastReceiver{
                                                        //继承自 BroadcastReceiver 的子类
2         @Override
3         public void onReceive(Context context, Intent intent) {//重写 onReceive 方法
4              double data = intent.getDoubleExtra("data", 0);
5              tv.setText("Service 的数据为:"+data);          //将收到的数据显示在屏幕上
6         }
7    }
```

说明 实现 Broadcast 组件最重要的是重写 onReceive 方法，上述代码的第 4～5 行从接收到的 Intent 对象中提取出 Extra 部分的信息并将其作为 TextView 的显示内容。

（3）编写好 Broadcast Receiver 组件类的代码后就应该在合适的地方对其进行注册和取消注册了。注册和取消 Broadcast Receiver 组件的代码写在 Activity 的 onStart 和 onStop 方法中，下面代码即是步骤（1）中代码第 43 行省略的部分。

```
1    @Override
2    protected void onStart() {                        //重写 onStart 方法
3         dataReceiver = new DataReceiver();
4         IntentFilter filter = new IntentFilter();    //创建 IntentFilter 对象
5         filter.addAction("wyf.wpf.Sample_3_6");
6         registerReceiver(dataReceiver, filter);      //注册 Broadcast Receiver
7         super.onStart();
8    }
9    @Override
10   protected void onStop() {                          //重写 onStop 方法
11        unregisterReceiver(dataReceiver);            //取消注册 Broadcast Receiver
12        super.onStop();
13   }
```

说明 上述代码第 5 行定义了 IntentFilter 的 Action 为自定义的字符串，该字符串必须和发送 Intent 时定义的 Action 字符串一致。

3.3.5 示例 3：应用程序组件间通信示例 Service 部分的开发

到这里 Activity 部分的功能已经开发完毕，下面开发 Service 部分的代码，Service 部分的开发主要分为以下 3 个步骤。

（1）在 Service 的 onCreate 方法中进行一些初始化的工作，在 onStartCommand 和 onDestroy 方法中注册和取消注册 Broadcast Receiver 组件。Service 类中的主要代码如下。

代码位置：见随书光盘中源代码/第 3 章/Sample_3_1/src/wyf/wpf 目录下的 MyService.java。

```
1    package wyf.wpf;                                  //声明包语句
```

```
2    import android.app.Service;                          //引入相关类
3    import android.content.BroadcastReceiver;            //引入相关类
4    import android.content.Context;                      //引入相关类
5    import android.content.Intent;                       //引入相关类
6    import android.content.IntentFilter;                 //引入相关类
7    import android.os.IBinder;                           //引入相关类
8    //继承自 Service 的子类
9    public class MyService extends Service{
10        CommandReceiver cmdReceiver;                    //继承自 BroadcastReceiver 的子类
11        boolean flag;                                   //线程执行的标志位
12        @Override
13        public void onCreate() {                        //重写 onCreate 方法
14             flag = true;                               //线程执行的标志位置为 true
15             cmdReceiver = new CommandReceiver();       //创建 BroadcastReceiver 子类的对象
16             super.onCreate();
17        }
18        @Override
19        public IBinder onBind(Intent intent) {          //重写 onBind 方法
20             return null;
21        }
22        @Override
23        public int onStartCommand(Intent intent, int flags, int startId){
                                                          //重写 onStartCommand 方法
24             IntentFilter filter = new IntentFilter();  //创建 IntentFilter 对象
25             filter.addAction("wyf.wpf.MyService");
26             registerReceiver(cmdReceiver, filter);     //注册 Broadcast Receiver
27             doJob();                                   //调用方法启动线程
28             return super.onStartCommand(intent, flags, startId);
29        }
30        @Override
31        public void onDestroy() {                       //重写 onDestroy 方法
32             this.unregisterReceiver(cmdReceiver);      //取消注册 CommandReceiver
33             super.onDestroy();
34        }
35        ……//此处省略定义 doJob 方法的代码，将在后面的步骤中补全
36        ……//此处省略继承自 BroadcastReceiver 类的 CommandReceiver 类的代码，将在后面的步骤中补全
37   }
```

说明　代码第 25 行 IntentFilter 定义的 Action 为自定义的字符串，其必须和 Activity 端发送 Intent 时设定的 Action 字符串一致。

（2）在步骤（1）中的代码第 27 行调用了 doJob 方法，该方法将启动一个新的线程定时发送 Intent，步骤（1）中代码第 35 行省略的 doJob 方法代码如下。

```
1    public void doJob(){
2         new Thread(){                                   //新建一个线程
3              public void run(){                         //重写线程的 run 方法
4                   while(flag){
5                        try{                             //睡眠一段时间
6                             Thread.sleep(1000);
7                        }
8                        catch(Exception e){
9                             e.printStackTrace();        //捕获并打印异常
10                       }
11                       Intent intent = new Intent();    //创建 Intent 对象
12                       intent.setAction("wyf.wpf.Sample_3_6");
13                       intent.putExtra("data", Math.random());
                                                          //以 data 为键以随机数为值
14                       sendBroadcast(intent);           //发送广播
```

```
15                    }
16                }
17            }.start();
18    }
```

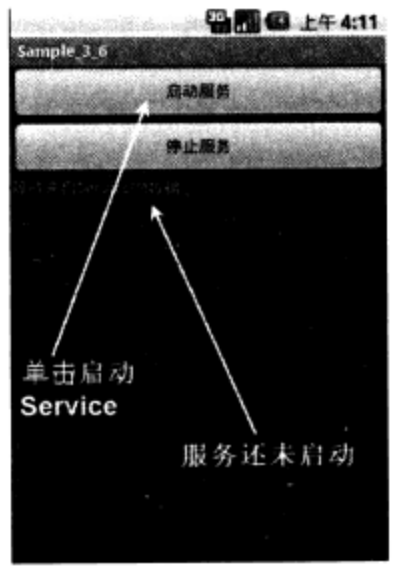
- 在新启动的线程中，每隔 1s 就向 Activity 发一个 Intent，并将其 Action 设置为"wyf.wpf.Sample_3_6"，与 Activity 注册的 IntentFilter 中所包含的 Action 字符串对应。
- 代码第 13 行产生一个随机数并将其赋值给 Intent 的 Extra 部分，代码第 14 行调用 sendBroadcast 方法将广播发出。

（3）为了让 Activity 可以启动和停止 Service，必须为 Service 注册 Broadcast Receiver 组件，需要开发继承自 BroadcastReceiver 的子类，步骤（1）中代码第 36 行省略的代码如下。

```
1    private class CommandReceiver extends BroadcastReceiver{
                                                    //继承自 BroadcastReceiver 的子类
2        @Override
3        public void onReceive(Context context, Intent intent) {
                                                    //重写 onReceive 方法
4            int cmd = intent.getIntExtra("cmd", -1);    //获取 Extra 信息
5            if(cmd == Sample_3_6.CMD_STOP_SERVICE){     //如果发来的消息是停止服务
6                flag = false;                           //停止线程
7                stopSelf();                             //停止服务
8            }
9        }
10   }
```

说明 代码第 4 行从 Intent 对象中的 Extra 部分提取出以“cmd”为键的值，并判断是否为停止服务的消息，如果是则设线程执行标志位为 false 并调用 stopSelf 方法停止服务。

程序运行后未启动服务时程序界面如图 3-8 所示，服务正在运行时程序界面如图 3-9 所示。



▲图 3-8　未启动服务时程序界面

▲图 3-9　服务运行时程序界面

需要注意的是，开发完 Activity 和 Service 部分的代码，还需要在 AndroidManifest.xml 文件中对组件进行声明，对于 Service，如果希望其在 Activity 退出后还能继续运行，需要在<service>标记中添加如下内容。

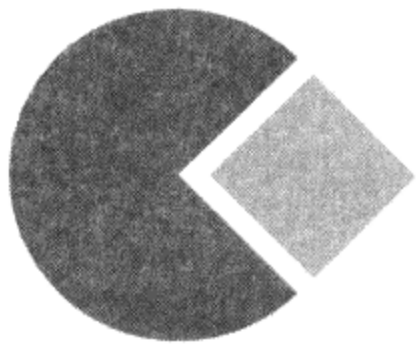
代码位置：见随书光盘中源代码/第 3 章/Sample_3_6 目录下的 AndroidManifest.xml。

```
1     ……//省略掉不必要的代码，读者可以自行查阅随书光盘
2     <service android:name=".MyService" android:process=":remote" >
                                                <!-- 让 Service 运行在远端进程中 -->
3     ……//省略掉不必要的代码，读者可以自行查阅随书光盘
```

这样就可以保证当管理并运行 Activity 等组件的主线程停止后，Service 仍然可以继续在后台运行。

3.4 小结

本章主要介绍了 Android 应用程序交互式通信的原理，首先介绍了 Android 应用程序的基本组件及程序中各线程和各组件之间是如何进行通信的，利用这些交互式通信机制，就可以开发出功能各异的 Android 应用程序。



第 4 章　Android 游戏开发之数据存储和传感器

本章将向读者介绍 Android 平台下的应用程序如何对数据进行存储和读取，这些知识对于开发一个功能完备的应用程序非常有必要。同时，本章还将简单介绍 Android 平台的一个特色——传感器的相关知识，读者可以使用传感器开发出各种新奇的应用。

4.1 在 Android 平台上实现数据存储

本节将会对 Android 平台上的数据存取方式做简单的介绍，主要包括基于文件的流读取、轻量级数据库 SQLite、Content Provider 以及 Preference 的应用介绍。

4.1.1　私有文件夹文件的写入与读取

在介绍如何在 Android 平台上进行文件读取之前，有必要了解 Android 平台上的数据存储规则。在其他的操作系统如 Windows 平台上，应用程序可以自由地或者在特定的访问权限基础上访问或修改其他应用程序名下的文件等资源，而在 Android 平台上，一个应用程序中所有的数据都是私有的，只对自己是可见的。

当应用程序被安装到系统中后，其所在的包会有一个文件夹用于存放自己的数据，仅这个应用程序才具有对这个文件夹的写入权限，这个私有文件夹位于 Android 系统的/data/data/<应用程序包名>目录下，其他的应用程序都无法在这个文件夹中写入数据。除了存放私有数据的文件夹外，应用程序也具有 SD 卡的写入权限。

使用文件 I/O 方法可以直接往手机中储存数据，默认情况下，这些文件不可以被其他的应用程序访问。Android 平台支持 Java 平台下的文件 I/O 操作，主要使用 FileInputStream 和 FileOutputStream 这两个类来实现文件的存储与读取。获取这两个类对象的方式有两种。

- 第一种方式就是像 Java 平台下的实现方式一样，通过构造器直接创建，如果需要向打开的文件末尾写入数据，可以通过使用构造器 FileOutputStream(File file,boolean append)将 append 设置为 true 来实现。需要注意的是，采用这种方式获得 FileOutputStream 对象时，如果文件不存在或不可以写入，程序会抛出 FileNotFoundException 异常。
- 第二种获取 FileInputStream 和 FileOutputStream 对象的方式是调用 Context.openFileInput 和 Context.openFileOutput 两个方法来创建。除了这两个方法外，Context 对象还提供了其他几个用于对文件操作的方法，如表 4-1 所示。

表 4-1　Context 对象中文件操作的 API 及说明

方 法 名	说　明
openFileInput(String filename)	打开应用程序私有目录下的指定私有文件以读入数据，返回一个 FileInputStream 对象
openFileOutput(String name,int mode)	打开应用程序私有目录下的指定私有文件以写入数据，返回一个 FileOutputStream 对象，如果文件不存在就创建这个文件
fileList()	搜索应用程序私有文件夹下的私有文件，返回所有文件名的 String 数组
deleteFile(String fileName)	删除指定文件名的文件，成功返回 true，否则返回 false

在使用 openFileOutput 方法打开文件用以写入数据时，需要指定打开模式。默认为零，即 MODE_ PRIVATE。不同的模式对应的含义如表 4-2 所示。

表 4-2　openFileOutput 方法打开文件时的模式

常　量	含　义
MODE_PRIVATE	默认模式，文件只可以被调用该方法的应用程序访问
MODE_APPEND	如果文件已存在就向该文件的末尾继续写入数据，而不是覆盖原来的数据
MODE_WORLD_READABLE	赋予所有的应用程序对该文件读的权限
MODE_WORLD_WRITEABLE	赋予所有的应用程序对该文件写的权限

下面通过一个例子来说明 Android 平台上的文件 I/O 操作方式，本例主要的功能是在应用程序私有的数据文件夹下创建一个文件，并读取其中的数据显示到屏幕的 TextView 中。本例中应用程序的开发分为以下 4 个步骤。

（1）首先，在应用程序的布局文件 main.xml 中声明屏幕中要显示的 TextView 控件，其代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_1/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       android:orientation="vertical"
4       android:layout_width="fill_parent" android:layout_height="fill_parent"
5       >                        <!-- 声明一个 LinearLayout 线性布局 -->
6      <TextView android:id="@+id/tv"
7         android:layout_width="fill_parent" android:layout_height="wrap_content"
8         />                    <!-- 声明一个 TextView 控件，id 为 tv -->
9    </LinearLayout>
```

说明　默认情况下，当新建一个工程时，会在 main.xml 中自动声明一个 TextView，不过该 TextView 是没有 ID 的，所以需要添加第 6 行的代码进行 ID 的声明。

（2）Activity 主要代码的开发。在 Activity 中主要进行的工作是，调用自己开发的文件写入和读取方法来获得数据信息，并将其内容显示在 TextView 中，其主要的代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_1/src/wyf/wpf 目录下的 Sample_4_1.java。

```
1    package wyf.wpf;                                    //声明包语句
2    import java.io.FileInputStream;                     //引入相关类
```

```
3    import java.io.FileOutputStream;                                  //引入相关类
4    import org.apache.http.util.EncodingUtils;                        //引入相关类
5    import android.app.Activity;                                      //引入相关类
6    import android.os.Bundle;                                         //引入相关类
7    import android.widget.TextView;                                   //引入相关类
8    //继承自 Activity 的子类
9    public class Sample_4_1 extends Activity {
10       public static final String ENCODING = "UTF-8";                //常量，为编码格式
11       String fileName = "test.txt";                                 //文件的名称
12       String message = "你好，这是一个关于文件 I/O 的示例。";        //写入和读出的数据信息
13       TextView tv;                                                  //TextView 对象引用
14         @Override
15         public void onCreate(Bundle savedInstanceState) {           //重写 onCreate 方法
16               super.onCreate(savedInstanceState);
17               setContentView(R.layout.main);                        //设置当前屏幕
18             writeFileData(fileName, message);                       //创建文件并写入数据
19               String result = readFileData(fileName);               //根据 id 获取 TextView 对象
                                                                       //的引用
20               tv = (TextView)findViewById(R.id.tv);                 //设置 TextView 的内容
21               tv.setText(result);                                   //获得从文件读入的数据
22         }
23       ……//此处省略 writeFileData 方法的代码，将在后面的步骤补全
24       ……//此处省略 readFileData 方法的代码，将在后面的步骤补全
25   }
```

- 程序的第 10 行定义了一个常量“ENCODING”，该常量代表编码格式，在随后的读入文件数据并转换成字符串对象时会用到。
- 代码第 11～12 行为需要操作的文件名和要写入的数据信息。
- 代码 18～19 行分别调用了 writeFileData 和 readFileData 方法用于向文件中写入数据和从文件中读取数据，在下面的步骤中将会编写这两个方法的具体代码。

（3）编写对文件进行写入的方法代码，步骤（1）代码第 23 行省略的 writeFileData 方法的代码如下。

```
1    //方法：向指定文件中写入指定的数据
2    public void writeFileData(String fileName,String message){
3      try{
4        FileOutputStream fout = openFileOutput(fileName, MODE_PRIVATE);
                                                  //获得 FileOutputStream
5        byte [] bytes = message.getBytes();      //将要写入的字符串转换为 byte 数组
6        fout.write(bytes);                       //将 byte 数组写入文件
7        fout.close();                            //关闭 FileOutputStream 对象
8      }
9      catch(Exception e){
10       e.printStackTrace();                     //捕获异常并打印
11     }
12   }
```

说明　writeFileData 方法所做的工作比较简单，其接收表示文件名和数据信息的字符串，通过 Context.openFileOutput 方法打开输出流，将字符串转换成 byte 数组写入文件。

（4）最后来编写从文件中读取数据的 readFileData 方法的代码，该方法在 Activity 的 onCreate 方法中被调用，下面代码即是步骤（1）中代码中第 24 行省略的 readFileData 方法的代码。

```
1    //方法：打开指定文件，读取其数据，返回字符串对象
2    public String readFileData(String fileName){
```

```
3        String result="";
4          try{
5               FileInputStream fin = openFileInput(fileName);
                                                        //获得 FileInputStream 对象
6               int length = fin.available();           //获取文件长度
7               byte [] buffer = new byte[length];      //创建 byte 数组用于读入数据
8               fin.read(buffer);                       //将文件内容读入到 byte 数组中
9               result = EncodingUtils.getString(buffer, ENCODING);
                                                        //将 byte 数组转换成指定格式的字符串
10         fin.close();                                 //关闭文件输入流
11         }
12         catch(Exception e){
13              e.printStackTrace();                    //捕获异常并打印
14      }
15         return result;                               //返回读到的数据字符串
16   }
```

- 代码第 5 行调用 Context.openFileInput 打开指定文件名的文件，获取 FileInputStream 对象，并在第 6 行调用其 available 方法获取文件的字节数。available 方法返回文件输入流中从当前位置算起所剩的字节数。
- 代码第 7 行根据获取的字节数创建一个与之大小相等的 byte 数组，在第 8 行通过调用 FileInputStream 的 read 方法将文件中的数据读入到 byte 数组中。
- 代码第 9 行调用了 EncodingUtils 类的静态方法 getString，将接收到的 byte 转换成指定编码格式的字符串。EncodingUtils 中封装了用于字符串编码的一些静态方法。

程序运行后界面如图 4-1 所示。本例采用的是调用 Context 的 openFileInput 和 openFileOutput 方法来获取文件的输入/输出流对象，读者也可以采用构造器的方式来获取文件的输入/输出流对象，在此不再赘述。

▲图 4-1　程序运行界面

4.1.2　读取 Resources 和 Assets 中的文件

在 Android 平台上，除了对应用程序的私有文件夹中的文件进行操作之外，还可以从资源文件和 Assets 中获得输入流读取数据，这些文件分别存放在应用程序的 res/raw 目录和 assets 目录下，这些文件将会在编译的时候和其他文件一起被打包。

需要注意的是，来自 Resources 和 Assets 中的文件只可以读取而不能够进行写操作，下面就通过一个例子来说明如何从 Resource 和 Assets 中的文件中读取信息。首先，分别在 res/raw 和 assets 目录下新建两个文本文件“test1.txt”和“test2.txt”用以读取。

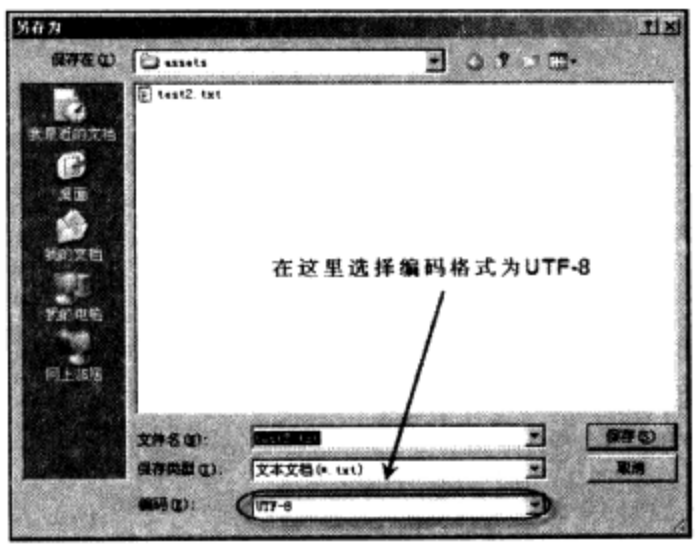
为避免字符串转码带来的麻烦，可以将这两个文本文件的编码格式设置为 UTF-8。设置编码格式的方法有多种，比较简单的一种是用 Windows 的记事本打开文本文件，在另存为对话框中的编码格式中选择“UTF-8”，如图 4-2 所示。

设置好文件的编码格式后，就可以开发源代码了，本例中的程序只包含一个 Activity，其开发步骤如下。

（1）首先，在应用程序的 main.xml 中声明两个 TextView 控件，代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_2/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       android:orientation="vertical"
4       android:layout_width="fill_parent" android:layout_height="fill_parent"
5       >                                          <!-- 声明一个 LinearLayout 线性布局 -->
6      <TextView android:id="@+id/tv1"
7       android:layout_width="fill_parent" android:layout_height="wrap_content"
8       />                                         <!-- 声明一个 TextView 控件，id 为 tv1 -->
9       <TextView android:id="@+id/tv2"
10      android:layout_width="fill_parent" android:layout_height="wrap_content"
11      />                                         <!-- 声明一个 TextView 控件，id 为 tv2 -->
12   </LinearLayout>
```



▲图 4-2　在记事本的另存为对话框中选择编码格式

说明　上述代码声明了 2 个 TextView 控件，将来在程序中将会分别显示读取自 res/raw 目录下的和读取自 asserts 目录下的文件数据。

（2）开发 Activity 的主要代码。Activity 的主要功能是调用方法分别读取来自 res/raw 目录下和 assets 目录下的文件的数据。其主要代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_2/src/wyf/wpf 目录下的 Sample_4_2.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import java.io.InputStream;                               //引入相关类
3    import org.apache.http.util.EncodingUtils;                //引入相关类
4    import android.app.Activity;                              //引入相关类
5    import android.os.Bundle;                                 //引入相关类
6    import android.widget.TextView;                           //引入相关类
7    //继承自 Activity 的子类
8    public class Sample_4_2 extends Activity {
9      public static final String ENCODING = "UTF-8";          //常量，代表编码格式
10     TextView tv1;                                           //TextView 的引用
11     TextView tv2;                                           //TextView 的引用
12       @Override
13       public void onCreate(Bundle savedInstanceState) {
14           super.onCreate(savedInstanceState);
15           setContentView(R.layout.main);                    //设置显示屏幕
16           tv1 = (TextView)findViewById(R.id.tv1);
17           tv2 = (TextView)findViewById(R.id.tv2);
18           tv1.setText(getFromRaw("test1.txt"));             //将 tv1 的显示内容设置为
                                                               //Resource 中的 raw 文件夹的文件
```

```
19          tv2.setText(getFromAsset("test2.txt"));        //将 tv2 的显示内容设置为 Asset
                                                           //中的文件
20        }
21      ……//此处省略 getFromRaw 方法的代码
22      ……//此处省略 getFromAsset 方法的代码
21    }
```

- 代码第 16～17 行调用 findViewById 方法获得屏幕上两个 TextView 控件的对象引用。
- 代码第 18～19 行将 Activity 中的两个 TextView 的显示内容设置为了两个方法 getFromRaw 和 getFromAsset 的返回值，这两个方法的具体代码将在后面的步骤中详细介绍。

（3）编写 getFromRaw 方法的代码。getFromRaw 方法的功能是从 res/raw 目录下读取指定文件名的文件，将其信息以字符串的形式返回，其代码为步骤（1）中第 21 行省略掉的部分，代码如下。

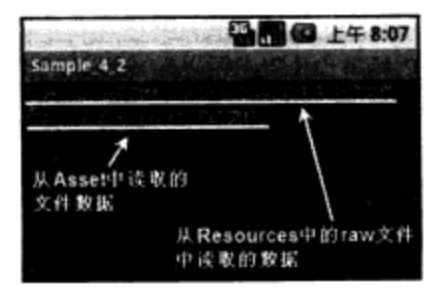
```
1     //方法：从 resource 中的 raw 文件夹中获取文件并读取数据
2     public String getFromRaw(String fileName){
3        String result = "";
4           try{
5            InputStream in = getResources().openRawResource(R.raw.test1);
                                                   //从 raw 中的文件获取输入流
6            int length = in.available();          //获取文件的字节数
7            byte [] buffer = new byte[length];    //创建 byte 数组
8            in.read(buffer);                      //将文件中的数据读取到 byte 数组中
9            result = EncodingUtils.getString(buffer, ENCODING);
                                                   //将 byte 数组转换成指定格式的字符串
10          }
11          catch(Exception e){
12           e.printStackTrace();                  //捕获异常并打印
13          }
14       return result;
15       }
```

- 代码第 5 行通过调用 Resources 的.openRawResource 方法从 raw 目录下的指定文件获取 InputStream 对象。
- 程序的第 6～9 行与 4.1.1 小节的代码比较类似，都是先获取文件的字节数，然后创建相同长度的 byte 数组，并将输入流中的数据读入到 byte 数组中，最后通过指定的编码格式将 byte 数组转换为字符串并将其返回。

（4）编写 getFromAsset 方法的代码。在步骤（1）代码中第 22 行省略的 getFromAsset 方法的代码与 getFromRaw 方法类似，只是获取输入流对象 InputStream 的方式不同，将步骤（2）代码的第 5 行改为以下代码就是 getFromAsset 方法的代码了。

```
1     InputStream in = getResources().getAssets().open(fileName);    //从 Assets 中的文件获
                                                                     //取输入流
```

程序运行后，会通过这两个方法读取文件中的数据显示到屏幕上，如图 4-3 所示。



▲图 4-3　程序运行后效果图

4.1.3　轻量级数据库 SQLite 简介

SQLite 是一款开源的嵌入式数据库引擎，其对多数的 SQL92 标准都提供了支持。相比于同样开源的 MySQL 和 PostgreSQL 来说，SQLite

具有以下独特的地方。

- 处理速度快。传统的数据库引擎采用的是客户端-服务端的访问方式，SQLite 的数据库引擎嵌入到程序中作为其一部分，所以运行的速度要快很多。
- 占用资源少。SQLite 源代码总共不到 3 万行，运行时所占内存不超过 250KB，而且不需要安装部署，并支持多线程访问。
- SQLite 中所有的数据库信息，如表、索引等全部集中存放在一个文件中，SQLite 支持事务，在开始一个新事务时会将整个数据库文件加锁。
- 支持 Windows、Linux 等主流操作系统，可以采用多种语言进行操作，如 Java、PHP 等。

总之，SQLite 的这些特性都非常适合作为移动设备的数据存储。Android 平台提供了对创建和使用 SQLite 数据库的支持，其装载了 sqlite3 组件，任何一个 SQLite 数据库对于创建该数据库的应用程序来说都是私有的，SQLite 的数据库文件位于/data/data/package-name/databases 目录下。

下面介绍在 Android 平台上如何对 SQLite 数据库进行操作，本书把对数据库的操作分成了 3 个步骤，分别是创建数据库对象、操作数据和检索数据。

1. 创建数据库对象

要想对数据库进行操作，首先要获得 SQLiteDatabase 对象，SQLiteDatabase 类提供了一些静态方法用于创建或打开一个数据库。这些方法及说明如表 4-3 所示。

表 4-3　　SQLiteDatabase 类提供的创建或打开数据库的方法

方　法	说　明
openDatabase(String path,SQLiteDatabase.CusorFactory factory,int flags)	以指定的模式打开指定路径名的数据库文件，使用指定的 Cusor 对象用于在查询中返回，如果传入 null 则使用默认
openOrCreateDatabase(File file, SQLiteDatabase.CusorFactory factory)	相当于 openDatabase(file.getPath(),factory, CREATE_IF_NECESSARY)
openOrCreateDatabase(String path, SQLiteDatabase.CusorFactory factory)	相当于 openDatabase(path,factory, CREATE_IF_NECESSARY)

其中 openDatabase 方法中可以通过参数 flags 指定打开的模式，不同的模式及其说明如表 4-4 所示。

表 4-4　　openDatabase 可指定的打开模式及其说明

模式名称	说　明
OPEN_READONLY	打开数据库的方式为只读
OPEN_READWRITE	打开数据库的方式为可读/可写
CREATE_IF_NECESSARY	如果指定的数据库文件不存在，就创建新的数据库
NO_LOCALIZED_COLLAORS	打开数据库时，不对数据进行基于本地化语言的排序

同在 4.1.1 小节文件 I/O 操作类似，Context 对象也提供了用于打开数据库的方法：openOrCreate Database (Sting name,int mode,SQLiteDatabase.CursorFactory factory)，该方法打开/创建指定名称的数据库文件，不过这里传入的 mode 并不是表 4-4 中的几种模式之一，而是 4.1.1 小节中表 4-2 中

的几种模式之一。

除了基于文件系统的数据库外，在 Android 中还可以创建内存数据库。调用 SQLiteDatabase 的静态方法 create(SQLiteDatabase.CursorFactory factory)创建内存数据库，如果对数据库的操作速度要求较高，可以考虑采用内存数据库来实现。

除了上述几种创建或打开数据库的方式外，在实际开发过程中，为方便起见，还可以开发一个数据库辅助类来创建或打开数据库，这个辅助类继承自 SQLiteOpenHelper 类，在该类的构造器中，调用 Context 中的方法创建并打开一个指定名称的数据库对象。继承和扩展 SQLiteOpenHelper 类主要做的工作就是重写以下两个方法。

- onCreate(SQLiteDatabase db)：当数据库被首次创建时执行该方法，一般将创建表等初始化操作放在该方法中执行。
- onUpgrade(SQLiteDatabase dv,int oldVersion,int newVersion)：当打开数据库时传入的版本号与当前版本号不同时会调用该方法。

说明　除了上述两个实现的方法外，还可以选择性地使用 onOpen 方法，该方法会在每次打开数据库时被调用。

SQLiteOpenHelper 类的基本用法是：当需要创建或打开一个数据库并获得数据库对象时，首先根据指定的文件名创建一个辅助对象，然后调用该对象的 getWritableDatabase 或 getReadableDatabase 方法获得 SQLiteDatabase 对象。

提示　调用 getReadableDatabase 方法返回的并不总是只读的数据库对象，一般来说该方法和 getWritableDatabase 方法的返回情况相同，只有在数据库仅开放只读权限或磁盘已满时才会返回一个只读的数据库对象。

2. 操作数据

对数据库的操作一般来讲就是增、删、改、查，SQLiteDatabase 类提供了很多方法用来对数据库进行操作，其中既含有直接接收 SQL 语句作为执行参数的方法，也含有专门用于增、删、改、查的方法。常用的数据库操作方法如表 4-5 所示。

表 4-5　常用的数据库操作方法及参数说明

方　法	方法说明	参数说明
(1)void execSQL(String sql) (2)void execSQL(String sql,Object[] bindArgs)	执行指定的 SQL 语句，不可以为查询语句，不可以传入以“;”隔开的多条 SQL 语句	sql：要执行的 SQL 语句 bindArgs：将来替换 SQL 表达式中“?”的参数列表，目前只支持 byte 数组、String、long 和 double 类型
(1)long insert (String table,String nullColumnHack, ContentValues values) (2)long insertOrThrow (String table,String nullColumnHack, ContentValues values)	向表中插入一行数据，返回插入行的 id。 方法 insertOrThrow 会抛出 SQLException 异常	table：要操作的表名 nullColumnHack：如果传入的 values 参数为 null 或空时，该参数所指定的列将会被赋值为 NULL 插入到表中 values：要插入的数据以“列名-列值”的键值对方式封装到 ContentValues 对象中

续表

方　法	方法说明	参数说明
int update (String table,ContentValues values, String whereClause,String[] whereArgs)	向表中更新数据，返回更新的行数	table：要操作的表名 values：更新后的“列名-列值”的键值对 whereClause：指明更新位置的 where 子句，如果为 null 则更新表中所有行 whereArgs：将来用于替换 whereClause 中“?”的参数列表
int delete (String table,String whereClause, String[] whereArgs)	删除表中满足指定条件的行，返回删除的个数	table：要操作的表名 whereClause：指明删除位置的 where 子句，如果传入“1”则会删除表中所有的行 whereArgs：将来用于替换 whereClause 中的“?”的参数列表

3. 检索数据

对数据的检索涉及的内容稍微多一些，所以在此单独介绍。SQLiteDatabase 中提供了直接解析 SQL 语句的查询方法和专门用于查询的方法，这些方法及说明如表 4-6 所示。

表 4-6　用于查询数据的方法及说明

方　法	方法说明	参数说明
(1)Cursor rawQuery(String sql,String [] args) (2)Cursor rawQueryWithFactory(SQLiteDatabase .CursorFactory factory,String sql,String [] args, String editable)	执行指定的 SQL 查询语句，返回 Cursor 对象作为查询结果	sql：要执行的 SQL 查询语句 args：用于替换 SQL 查询语句中“?”的参数列表 factory：构造 Cursor 对象时所使用的 CursorFactory 对象，为 null 则使用默认 editTable：首个可编辑表的名称
(1)Cursor query(boolean distinct,String table, String[] columns,String selection,String[] selectionArgs, String groupBy,String having, String orderBy,String limit) (2)Cursor query (String table,String[] columns, String selection,String[] selectionArgs,String groupBy,String having,String orderBy) (3)Cursor query(String table,String[] columns, String selection,String[] selectionArgs,String groupBy, String having,String orderBy,String limit) (4)Cursor queryWithFactory(SQLiteDatabase. CursorFactory cursorFactory,boolean distinct, String table,String[] columns,String selection, String[] selectionArgs,String groupBy,String having,String orderBy,String limit)	查询指定的表，返回 Cursor 对象作为查询结果	distinct：是否要求每一行具有惟一性，true 表示要求 table：要操作的表名 columns：指明查询结果中需要返回的列，如果传入 null 则返回所有列 selection：对应 SQL 语句中 where 子句，指明查询结果需要返回的行数，传入 null 会返回所有行 selectionArgs：用于替换 selection 参数中的“?”的参数列表 groupBy：对应 SQL 语句中的 group by 子句，对查询结果分组，传入 null 表示不分类 having：对应 SQL 语句中的 having 子句，需要与 groupBy 参数配合使用，传入 null 表示不对分组结果附加条件 orderBy：对应 SQL 语句中的 order by 语句，传入 null 为默认排序 limit：对应 SQL 语句中的 limit 语句，对查询返回的行数进行限制，传入 null 表示不作限制

说明　表 4-6 列出的一系列的 query 方法，其实是将 SQL 语句进行分解，让整个 SQL 语句的每个部分和子句都独立出来作为供调用者选择的参数。需要注意，如果这些参数对应 SQL 中的某个子句，在传入参数的时候不应该包含这些子句的关键字，如 selection 参数中不应该包含“where”关键字。

4.1.4　SQLite 的使用示例

前面的小节简单介绍了 SQLite 的相关知识，本小节将会举一个简单的例子来说明 SQLite 的用法。本例中使用数据库的辅助类创建和打开数据库，并在 Activity 中调用方法对数据库进行插入、更新和查询等操作，整个程序分以下 4 个步骤来完成。

（1）首先，在应用程序的 main.xml 中声明 TextView 控件。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_3/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent" android:layout_height="fill_parent"
5        >                          <!-- 声明一个 LinearLayout 线性布局 -->
6      <TextView android:id="@+id/tv"
7              android:layout_width="fill_parent" android:layout_height="wrap_content"
8        />                         <!-- 声明一个 TextView 控件，id 为 tv -->
9    </LinearLayout>
```

说明　代码第 6～8 行声明的 TextView 控件将会在程序中负责显示从数据库中检索到的信息。

（2）创建数据库的辅助类。该类继承自 SQLiteOpenHelper 类，并将重写其 onCreate 等方法，其代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_3/src/wyf/wpf 目录下的 MySQLiteHelper. java。

```
1    package wyf.wpf;                                                    //声明包语句
2    import android.content.Context;                                     //引入相关类
3    import android.database.sqlite.SQLiteDatabase;                      //引入相关类
4    import android.database.sqlite.SQLiteOpenHelper;                    //引入相关类
5    import android.database.sqlite.SQLiteDatabase.CursorFactory;        //引入相关类
6    //继承自 SQLiteOpenHelper 的子类
7    public class MySQLiteHelper extends SQLiteOpenHelper{
8         public MySQLiteHelper(Context context, String name, CursorFactory factory,
9                   int version) {
10             super(context, name, factory, version);                   //调用父类的构造器
11        }
12        @Override
13        public void onCreate(SQLiteDatabase db) {                      //重写 onCreate 方法
14             db.execSQL("create table if not exists hero_info("        //调用 execSQL 方法创
                                                                         //建表
15                      + "id integer primary key,"
```

```
16                          + "name varchar,"
17                          + "level integer)");
18          }
19          ……//在此省略重写的 onUpgrade 方法，读者可以自行查阅随书光盘
20    }
```

- 代码第 10 行调用了父类的构造器创建或打开数据库文件。
- 代码第 13～18 行为重写的 onCreate 方法，在该方法中创建了一张名为“hero_info”的表，表中有“id”、“name”和“level”3 个字段。onCreate 方法在第一次创建数据库文件时被调用。
- 代码第 14 行调用了 execSQL 方法，需要注意的是，传入的 SQL 语句字符串中不需要加上分号。

（3）开发 Activity 部分的代码。创建完数据库的辅助对象后，就需要在 Activity 中开发针对数据库具体操作的方法了，Activity 中主要代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_3/src/wyf/wpf 目录下的 Sample_4_3.java。

```
1     package wyf.wpf;                                          //声明包语句
2     import android.app.Activity;                              //引入相关类
3     import android.content.ContentValues;                     //引入相关类
4     import android.database.Cursor;                           //引入相关类
5     import android.database.sqlite.SQLiteDatabase;            //引入相关类
6     import android.os.Bundle;                                 //引入相关类
7     import android.widget.TextView;                           //引入相关类
8     //继承自 Activity 的子类
9     public class Sample_4_3 extends Activity {
10      MySQLiteHelper myHelper;                                //数据库辅助类对象的引用
11      TextView tv;                                            //TextView 对象的引用
12      @Override
13       public void onCreate(Bundle savedInstanceState) {
14           super.onCreate(savedInstanceState);
15           setContentView(R.layout.main);                     //设置显示的屏幕
16           tv = (TextView)findViewById(R.id.tv);              //获得 TextView 对象的引用
17           myHelper = new MySQLiteHelper(this, "my.db", null, 1);
                                                                //创建数据库辅助类对象
18           insertAndUpdateData(myHelper);                     //向数据库中插入和更新数据
19           String result = queryData(myHelper);               //向数据库中查询数据
20           tv.setText("名字\t 等级\n"+result);                //将查询到的数据显示到屏幕上
21         }
22     ……//在此省略 insertAndUpdateData 方法的代码，将在随后补全
23     ……//在此省略 queryData 方法的代码，将在随后补全
24     @Override
25     protected void onDestroy () {
26         SQLiteDatabase db = myHelper.getWritableDatabase(); //获取数据库对象
27         db.delete("hero_info", "1", null);
                                                    //删除 hero_info 表中的所有数据
28         super. onDestroy ();
29     }
30   }
```

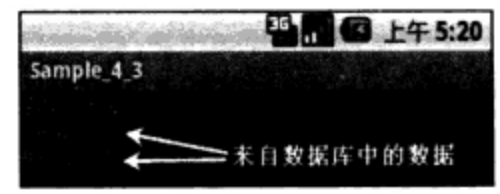
- 代码第 17 行创建了数据库的辅助对象，其主要的作用是，在随后的代码中通过调用 getWritableDatabase 或 getReadableDatabase 方法来获得数据库对象。
- 代码第 25～29 行重写了 onDestroy 方法，即在程序退出时删除 hero_info 表中所有的数据。

（4）编写 insertAndUpdateData 和 queryData 方法的代码。在步骤（2）中的代码第 18 行和第 19 行分别调用了 insertAndUpdateData 和 queryData 方法，这两个方法的功能是对数据库进行各种

操作，其代码如下。

```
1    //方法：向数据库中的表中插入和更新数据
2    public void insertAndUpdateData(MySQLiteHelper myHelper){
3         SQLiteDatabase db = myHelper.getWritableDatabase();      //获取数据库对象
4         //使用 execSQL 方法向表中插入数据
5         db.execSQL("insert into hero_info(name,level) values('Hero1',1)");
6         //使用 insert 方法向表中插入数据
7         ContentValues values = new ContentValues();
                                        //创建 ContentValues 对象存储“列名-列值”映射
8         values.put("name", "hero2");
9         values.put("level", 2);
10        db.insert("hero_info", "id", values);                    //调用方法插入数据
11        //使用 update 方法更新表中的数据
12        values.clear();                                   //清空 ContentValues 对象
13        values.put("name", "hero2");
14        values.put("level", 3);
15        db.update("hero_info", values, "level = 2", null);
                                                            //更新表中 level 为 2 的那行数据
16        db.close();                                       //关闭 SQLiteDatabase 对象
17   }
18   //方法：从数据库中查询数据
19   public String queryData(MySQLiteHelper myHelper){
20        String result="";
21        SQLiteDatabase db = myHelper.getReadableDatabase();      //获得数据库对象
22        Cursor cursor = db.query("hero_info", null, null, null, null, null, "id asc");
                                                                   //查询表中数据
23        int nameIndex = cursor.getColumnIndex("name");           //获取 name 列的索引
24        int levelIndex = cursor.getColumnIndex("level");         //获取 level 列的索引
25        for(cursor.moveToFirst();!(cursor.isAfterLast());cursor.moveToNext()){
                                                                   //遍历结果集，提取数据
26             result = result + cursor.getString(nameIndex)+"    ";
27             result = result + cursor.getInt(levelIndex)+"     \n";
28        }
29        cursor.close();                                          //关闭结果集
30        db.close();                                              //关闭数据库对象
31        return result;
32   }
```

- 代码第 5、第 10 行分别采用不同的方式向表中插入数据，而代码第 15 行通过调用 update 方法将之前插入 level 为 2 的那行数据进行修改。
- 代码第 22 行调用了 query 方法对表进行查询，由于传入的各项条件（如 where 等子句）都为 null，该方法返回的将是表中的所有行，返回的结果按 id 列的升序排列。
- 代码第 25 行是对查询返回的 Cursor 对象进行遍历的代码，moveToFirst 方法将 Cursor 移动到数据的第一行，isAfterLast 方法判断 Cursor 是否已经位于最后一行之后，moveToNext 方法将 Cursor 移动到下一行。
- 代码第 29 行和第 30 行分别调用 Cursor 和 SQLiteDatabase 的 close 方法将其关闭。程序运行后如图 4-4 所示。



▲图 4-4　程序运行效果图

提示　由于本例中只是说明 SQLite 的用法，所以数据库中涉及的表名、列名并没有使用将其定义为常量的做法，建议读者在开发的过程中将这些信息定义为常量，这样代码的可读性和可维护性都会得到提高。

4.1.5 数据共享者——Content Provider 的使用

Content Provider 属于 Android 应用程序的组件之一，这点在本书的第 3 章已经有所介绍。作为应用程序之间惟一的共享数据的途径，Content Provider 主要的功能就是存储并检索数据以及向其他应用程序提供访问数据的接口。

Android 系统为一些常见的数据类型（如音频、视频、图像、手机通信录联系人信息等）内置了一系列的 Content Provider，这些都位于 android.provider 包下。持有特定的许可，可以在自己开发的应用程序中访问这些 Content Provider。

让自己的数据和其他应用程序共享有两种方式：创建自己的 Content Provider（即继承自 ContentProvider 的子类）或者是将自己的数据添加到已有的 Content Provider 中去，后者需要保证现有的 Content Provider 和自己的数据类型相同，并且具有该 Content Provider 的写入权限。对于 Content Provider，最重要的就是数据模型（data model）和 URI。

1. 数据模型（data model）

Content Provider 将其存储的数据以数据表的形式提供给访问者，在数据表中每一行为一条记录，每一列为具有特定类型和意义的数据。每一条数据记录都包括一个“_ID”数值字段，该字段惟一标识一条数据。

2. URI

URI，每一个 Content Provider 都对外提供一个能够惟一标识自己数据集（data set）的公开 URI，如果一个 Content Provider 管理多个数据集，其将会为每个数据集分配一个独立的 URI。所有的 Content Provider 的 URI 都以“content://”开头，其中“content:”是用来标识数据是由 Content Provider 管理的 scheme。

在几乎所有的 Content Provider 的操作中都会用到 URI，因此，如果是自己开发 Content Provider，最好将 URI 定义为常量，这样在简化开发的同时也提高了代码的可维护性。

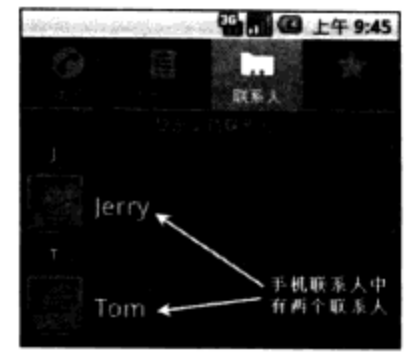
首先来介绍如何访问 Content Provider 中的数据，访问 Content Provider 中的数据主要通过 ContentResolver 对象，ContentResolver 类提供了成员方法可以用来对 Content Provider 中的数据进行查询、插入、修改和删除等操作。以查询为例，查询一个 Content Provider 需要掌握如下的信息。

- 惟一标识 Content Provider 的 URI。
- 需要访问的数据字段名称。
- 该数据字段的数据类型。

提示 如果需要访问特定的某条数据记录，只需该记录的 ID 即可。

查询 Content Provider 的方法有两个：ContentResolver 的 query()和 Activity 对象的 managedQuery()，二者接收的参数相同，返回的都是 Cursor 对象，惟一的不同是使用 managedQuery 方法可以让 Activity 来管理 Cursor 的生命周期。

被管理的 Cursor 会在 Activity 进入暂停态的时候调用自己的 deactivate 方法自行卸载，而在 Activity 回到运行态时会调用自己的 requery 方法重新查询生成 Cursor 对象。如果一个未被管理的 Cursor 对象想被 Activity 管理，可以调用 Activity 的 startManagingCursor 方法来实现。



▲图 4-5　联系人（Contacts）程序中存在的联系人信息

下面通过一个例子来说明访问 Content Provider 的方式，本例中通过 ContentResolver 对象访问 Android 中存储了联系人信息的 Content Provider 并将数据显示到 TextView 上，其开发步骤如下。

（1）首先在手机模拟器上运行“联系人(Contacts)”程序，在其中添加两个联系人“Tom”和“Jerry”，添加成功后如图 4-5 所示。

（2）在应用程序的 main.xml 文件中声明 TextView 控件，代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_4/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent" android:layout_height="fill_parent"
5        >                          <!-- 声明一个 LinearLayout 线性布局 -->
6      <TextView android:id="@+id/tv"
7            android:layout_width="fill_parent" android:layout_height="wrap_content"
8          />                       <!-- 声明一个 TextView 控件，id 为 tv -->
9    </LinearLayout>
```

说明　代码第 6～8 行声明的 TextView 控件将会在程序中负责显示从 Content Provider 中查询到的信息。

（3）开发 Activity 的代码。本程序中 Activity 的主要功能是得到持有联系人信息的 Content Provider 中的数据，其代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_4/src/wyf/wpf 目录下的 Sample_4_4.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.content.ContentResolver;                   //引入相关类
4    import android.database.Cursor;                           //引入相关类
5    import android.net.Uri;                                   //引入相关类
6    import android.os.Bundle;                                 //引入相关类
7    import android.provider.Contacts.People;                  //引入相关类
8    import android.widget.TextView;                           //引入相关类
9    //继承自 Activity 的子类
10   public class Sample_4_4 extends Activity {
11       String [] columns = {                        //查询 Content Provider 时希望返回的列
12            People._ID,
13            People.NAME,
14       };
15       Uri contactUri = People.CONTENT_URI;         //访问 Content Provider 需要的 Uri
16       TextView tv;                                 //TextView 对象引用
17        @Override
18        public void onCreate(Bundle savedInstanceState) {    //重写 onCreate 方法
19           super.onCreate(savedInstanceState);
```

```
20              setContentView(R.layout.main);
21              tv = (TextView)findViewById(R.id.tv); //获得 TextView 对象引用
22              String result = getQueryData();       //调用方法访问 Content Provider
23              tv.setText("ID\t 名字\n"+result);      //将查询到的信息显示到 TextView 中
24        }
25        //方法：获取联系人列表信息，返回 String 对象
26        public String getQueryData(){
27        String result = "";
28        ContentResolver resolver = getContentResolver();
                                                      //获取 ContentResolver 对象
29          Cursor cursor = resolver.query(contactUri, columns, null, null, null);
                                                      //调用方法查询 Content Provider
30              int idIndex = cursor.getColumnIndex(People._ID);
                                                      //获得_ID 字段的列索引
31              int nameIndex = cursor.getColumnIndex(People.NAME);
                                                      //获得 NAME 字段的列索引
32              for(cursor.moveToFirst();(!cursor.isAfterLast());cursor.moveToNext()){
                                                      //遍历 Cursor， 提取数据
33                   result = result + cursor.getString(idIndex)+ "\t";
34                   result = result + cursor.getString(nameIndex)+ "\t\n";
35              }
36              cursor.close();                       //关闭 Cursor 对象
37              return result;
38        }
39    }
```

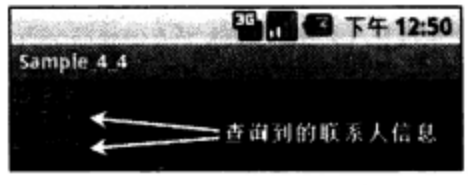
- 代码第 11 行定义了一个 String 数组，该数组中包含了查询 Content Provider 时希望返回的列，其将会作为 ContentResolver 的 query 方法的一个参数传入。
- 代码第 15 行获取了存储联系人信息的 Content Provider 的数据表的 URI。URI 对于 Content Provider 很重要，ContentResolver 中几乎所有方法的第一个参数都是一个 URI 对象，URI 对象告诉了 ContentResolver 应该和哪个 Content Provider 交互、应该与其中的哪个数据表关联。
- 代码第 22 行调用了 getQueryData 方法，该方法在代码的第 25～38 行定义，在 getQueryData 方法中首先获得 ContentResolver 对象，然后调用其 query 方法查询给定 URI 的数据表，最后遍历返回的 Cursor 对象，将内容提取并存储到一个字符串中并返回。

（4）最后还需要在 AndroidManifest.xml 中为应用程序声明访问联系人信息的权限，否则在运行时会抛出异常，声明权限的代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_4 目录下的 AndroidManifest.xml。

```
1     ……  //此处省略不相关代码，读者可以自行查阅随书光盘
2     <uses-permission android:name="android.permission.READ_CONTACTS" />
3     ……  //此处省略不相关代码，读者可以自行查阅随书光盘
```

程序运行后如图 4-6 所示。



▲图 4-6　程序运行后效果图

由上面的程序不难看出，访问一个 Content Provider 并不复杂，只需要牢牢掌握 3 个要素：URI、数据字段、数据类型即可。

提示　对于向 Content Provider 中添加新的数据记录以及修改、删除数据记录，其实现方式同查询类似，只不过需要用到 ContentValues 对象来存放数据表的“列名-列值”的映射，在此不再赘述。

如果需要创建一个 Content Provider，则需要进行的工作主要分为以下 3 个步骤。

（1）建立数据的存储系统。

数据的存储系统可以由开发人员任意决定，一般来讲，大多数的 Content Provider 都通过 Android 的文件存储系统或 SQLite 数据库建立自己的数据存储系统。

（2）扩展 ContentProvider 类。

开发一个继承自 ContentProvider 类的子类代码来扩展 ContentProvider 类，这个步骤主要的工作是将要共享的数据包装并以 ContentResolver 和 Cursor 对象能够访问到的形式对外展示。具体来说需要实现 ContentProvider 类中的 6 个抽象方法。

- Cursor query(Uri uri,String[] projection,String selection,String[] selectionArgs,String sortOrder)：将查询的数据以 Cursor 对象的形式返回。
- Uri insert(Uri uri,ContentValues values)：向 Content Provider 中插入新数据记录，ContentValues 为数据记录的列名和列值的映射。
- int update(Uri uri,ContentValues values,String selection,String[] selectionArgs)：更新 Content Provider 中已存在的数据记录。
- int delete(Uri uri,String selection,String[] selectionArgs)：从 Content Provider 中删除数据记录。
- String getType(Uri uri)：返回 Content Provider 中数据的（MIME）类型。
- boolean onCreate( )：当 Content Provider 启动时被调用。

以上方法将会在 ContentResolver 对象中被调用，所以很好地实现这些抽象方法会为 ContentResolver 提供一个完善的外部接口。除了实现抽象方法外，还可以做一些提高可用性的工作。

- 定义一个 URI 类型的静态常量，命名为 CONTENT_URI。必须为该常量对象定义一个惟一的 URI 字符串，一般的做法是将 ContentProvider 子类的全程类名作为 URI 字符串，如“content://wyf.wpf.MyProvider”。
- 定义每个字段的列名，如果采用的数据存储系统为 SQLite 数据库，数据表列名可以采用数据库中表的列名。不管数据表中有没有其他的惟一标识一条记录的字段，都应该定义一个“_id”字段来惟一标识一条记录。一般将这些列名字符串定义为静态常量，如“_id”字段名定义为一个名为“_ID”值为“_id”的静态字符串对象。

（3）在应用程序的 AndroidManifest.xml 文件中声明 Content Provider 组件。

创建好一个 Content Provider 必须要在应用程序的 AndroidManifest.xml 中进行声明，否则该 Content Provider 对于 Android 系统将是不可见的。声明一个 Content Provider 组件的方法在第 3 章已经有所介绍，如果有一个名为 MyProvider 的类扩展了 ContentProvider 类，声明该组件的代码如下。

```
1    <provider name="wyf.wpf.MyProvider"
2            authorities="wyf.wpf.myprovider"
3            . . . />                <!--  为<provider>标记添加 name、authorities 属性  -->
4    </provider>
```

其中 name 属性为 ContentProvider 子类的全称类名，authorities 属性惟一标识了一个 ContentProvider。除了这些，还有设置读写权限等内容，在此将不赘述，读者可以查阅第 3 章的相关知识或其他书籍。

提示	创建一个 Content Provider 所要进行的工作比较复杂，涉及的代码量较多，且不属于本书的研究范畴，由于篇幅有限，便不对 Content Provider 的创建进行举例。

4.1.6 简单的数据存储——Preferences 的使用

Preferences 是一种应用程序内部轻量级的数据存储方案。Preferences 主要用于存储和查询简单数据类型的数据，这些简单数据类型包括 boolean、int、float、long 以及 String 等，存储方式为以键值对的形式存放在应用程序的私有文件夹下。

Preferences 一般用来存储应用程序的设置信息，如应用程序的色彩方案、文字字体等。在应用程序中获取 Preferences 的方式有如下两种。

- 调用 Context 对象的 getSharedPreferences 方法获得 SharedPreferences 对象。需要传入 SharedPreferences 的名称和打开模式，名称为 Preferences 文件的名称，如果不存在则创建一个以传入名称为名的新的 Preferences 文件；打开模式为 PRIVATE、MODE_WORLD_ READABLE 和 MODE_WORLD_WRITEABLE 其中之一。
- 调用 Activity 对象的 getPreferences 方法获得 SharedPreferences 对象。需要传入打开模式，打开模式为 PRIVATE、MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE 其中之一。

通过两种不同途径获得的 SharedPreferences 对象并不是完全相同的，区别如下所列。

- 通过 Context 对象的 getSharedPreferences 方法获得的 SharedPreferences 对象可以被同一应用程序中的其他组件共享。
- 使用 Activity 对象的 getPreferences 方法获得的 SharedPreferences 对象只能在相应的 Activity 中使用。

SharedPreferences 对象中提供了一系列的 get 方法用于接收键返回对应的值。如果需要对 Preferences 文件中存储的键值对进行修改，首先需要调用 SharedPreferences 的 edit 方法获得一个 Editor 对象，该对象可以用来修改 Preferences 文件中存储的内容。下面将通过一个小例子来说明 Preferences 的用法，整个例子的开发分为如下几个步骤。

（1）本例的 Activity 中有一个 EditText 控件，需要在布局文件 main.xml 中声明它，实现代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_5/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent" android:layout_height="fill_parent"
5        >                                  <!--  创建一个线性布局 LinearLayout  -->
6      <EditText android:id="@+id/et"
7          android:layout_width="fill_parent" android:layout_height="wrap_content"
8          />                               <!-- 创建一个 EditText 控件，id 为 "et"  -->
9    </LinearLayout>
```

说明	上述代码在线性布局中声明了一个 EditText 控件，并为 EditText 控件指定 id，以便在 Activity 的代码中可以通过 id 找到该 EditText 控件。

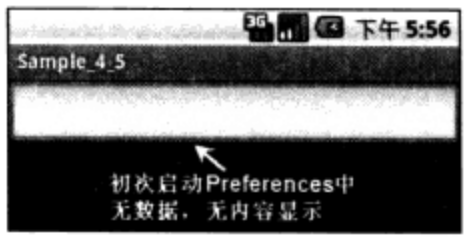
（2）每次启动 Activity 时，会从应用程序的 Preferences 文件中读取数据并显示出来；每次退出 Activity 时，将 EditText 控件当前的内容存储到 Preferences 文件中，程序的 Activity 代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_5/src/wyf/wpf 目录下的 Sample_4_5.java。

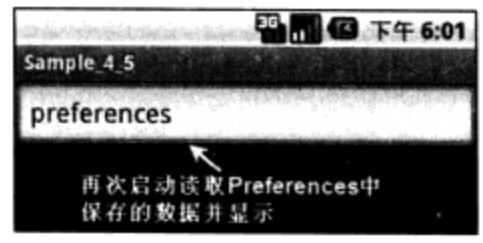
```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.content.SharedPreferences;                 //引入相关类
4    import android.os.Bundle;                                 //引入相关类
5    import android.widget.EditText;                           //引入相关类
6    //继承自 Activity 的子类
7    public class Sample_4_5 extends Activity {
8         EditText etPre;                                      //EditText 对象的引用
9         SharedPreferences sp;                                //SharedPreference 对象的引用
10        public final String EDIT_TEXT_KEY = "EDIT_TEXT";//定义 Preferences 文件中的键
11       @Override
12       public void onCreate(Bundle savedInstanceState) { //重写 onCreate 方法
13          super.onCreate(savedInstanceState);
14          setContentView(R.layout.main);
15          etPre = (EditText)findViewById(R.id.et);  //获得屏幕中 EditText 对象引用
16          sp = getPreferences(MODE_PRIVATE);        //获得 SharedPreferences 对象
17          String result = sp.getString(EDIT_TEXT_KEY, null);
18          if(result != null){                       //判断获取的值是否为空
19          etPre.setText(result);                //EditText 对象显示的内容设置为读取的数据
20          }
21        }
22       @Override
23       protected void onDestroy() {              //重写 onDestroy 方法
24          SharedPreferences.Editor editor = sp.edit();
                                                   //获得 SharedPreferences 的 Editor 对象
25          editor.putString(EDIT_TEXT_KEY, String.valueOf(etPre.getText()));
                                                   //修改数据
26          editor.commit();                       //必须调用该方法以提交修改
27          super.onDestroy();
28       }
29   }
```

- 代码第 9 行声明了 SharedPreferences 对象的引用，第 10 行定义了 String 类型的常量，用来表示 Preferences 中的一个键。
- 代码第 16 行调用 Activity 的 getPreferences 方法获得 SharedPreferences 对象，这种方式获得的 SharedPreferences 对象只能被 Activity 使用。
- 代码第 17 行调用 SharedPreferences 对象的 getString 方法读取以 EDIT_TEXT_KEY 为键的值，第二个参数 null 表示如果该键值对不存在则返回 null。
- 代码第 23～28 行重写了 Activity 的 onDestroy 方法，在该方法中首先获得 Editor 对象，然后使用 Editor 对象修改 Preferences 中的数据。数据修改完毕后一定要调用 Editor 对象的 commit 方法类提交修改。

启动应用程序后，在 EditText 控件中输入一些内容，然后关闭程序，再次启动程序，EditText 控件中显示的内容为上次退出时的内容。两次启动程序时屏幕的显示内容如图 4-7 和图 4-8 所示。



▲图 4-7　第一次启动程序时屏幕的显示



▲图 4-8　第二次启动程序时屏幕的显示

说明　其实对于每个 SharedPreferences 对象，Android 都在应用程序的私有文件夹下建立了一个以 SharedPreferences 对象名称命名的 XML 文件（对于通过 Activity 的 getPreferences 创建的 Preferences 文件，其文件名为 Activity 的名称），键值对都存放在相应的标记中。

4.2 Android 平台下传感器应用的开发

Android 平台中有许多令用户津津乐道的特性，其中最令人眼前一亮的莫过于 Android 平台下的传感器了，利用传感器可以开发出包括游戏在内的很多新奇的应用。

Android 平台下的传感器包括加速度传感器（Accelerometer）、姿态传感器（Orientation）、磁场传感器（Magnetic Field）和光传感器（Light）等。在对主要的几种传感器做一个简单的介绍之前，有必要先了解 Android 平台下传感器应用开发流程。

4.2.1　传感器应用开发流程

Android 平台下传感器应用开发是通过监听器机制来实现，要针对某一种或多种传感器开发应用，主要的步骤如下。

1. 创建 SensorManager 对象

通过 SensorManager 可以访问到手持设备的传感器，同时 SensorManager 对象还提供了一些方法用于对捕获的数据进行一些计算等处理。在程序中，通过调用 Context.getSystemService 方法传入参数“SENSOR_SERVICE”来获得 SensorManager 对象。

2. 实现 SensorListener 接口

这是开发传感器应用最主要的工作，实现 SensorListener 接口主要需要实现以下两个方法。

- void onAccuracyChanged(int sensor,int accuracy)：该方法在传感器的精确度发生变化时调用，SensorManager 提供了 3 种精确度，由高到低分别为 SENSOR_STATUS_ACCURACY_ HIGH、SENSOR_STATUS_ACCURACY_MEDIUM 和 SENSOR_STATUS_ACCURACY_LOW。参数 accuracy 为新的精确度。
- void onSensorChanged(int sensor,float[] values)：该方法在传感器的数据发生变化时调用，开发传感器应用的主要的业务代码应该放在这里执行，如读取数据并根据数据的变化进行相应的操作

等。方法传入的参数 sensor 表示为传感器类型的常量，values 为 float 类型数组，其长度和内容因传感器类型的不同而变化。

说明

应用开发中主要使用的是 values 数组，在后面的小节将陆续介绍不同的传感器事件发生时 values 中数据所代表的不同含义。

3. 注册 SensorListener

开发完 SensorListener 之后，剩下的工作就是在程序的适当位置注册监听和取消注册监听了，在这里调用步骤 1 中获得的 SensorManager 对象的 registerListener 方法来注册监听器，其接收的参数为监听器对象、传感器类型以及传感器事件传递的频度。

取消注册 SensorListener 时调用 SensorManager 的 unregisterListener 方法，一般来讲，注册和取消注册的方法应该成对出现，如果在 Activity 的 onResume 方法中注册 SensorListener 监听，就应在 onPause 方法中取消注册。

4.2.2　常用传感器简介

1. 加速度传感器 Accelerometer

加速度传感器主要感应手机的运动，在注册了传感器监听器后，加速度传感器主要捕获 3 个参数 values[0]、values[1]和 values[2]，这 3 个数据代表的含义如下。

- values[0]：空间坐标系中 x 轴方向上的加速度减去重力加速度在 x 轴上的分量。
- values[1]：空间坐标系中 y 轴方向上的加速度减去重力加速度在 y 轴上的分量。
- values[2]：空间坐标系中 z 轴方向上的加速度减去重力加速度在 z 轴上的分量。

上述 3 个数据的单位均为 m/s^2，在对其上述的含义做详细解释之前，首先了解一下 Android 平台下手机的空间坐标系，如图 4-9 所示。

从图 4-9 中可以看出，整个空间坐标系的原点位于手机屏幕的左下角，这点和手机的平面坐标系不同。当手机平放到桌面上时，z 轴为垂直于手机屏幕竖直向上的方向；x 轴为垂直于 z 轴代表手机屏幕的水平方向的轴，向右为正；y 轴为垂直于 z 轴和 x 轴代表手机屏幕的垂直方向的轴，向前为正。

下面举例说明 3 个方向上的加速度值的获取。

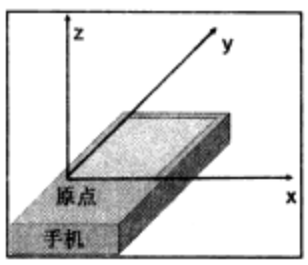
- 当手机平放到桌面静止时，加速度为重力加速度 g，通过 0 减去$-g$（重力加速度 g 方向为 z 轴反向，故为负值）得到 values[2]为 g。
- 如果把手机水平向右推，此时手机 x 方向上的加速度为正，即 values[0]为正。
- 当把手机以 a m/s^2 的加速度竖直向上举时，values[2] 的返回值为$(a+g)$ m/s^2，通过加速度 a 减去负的重力加速度$-g$ 得到。

2. 姿态传感器 Orientation

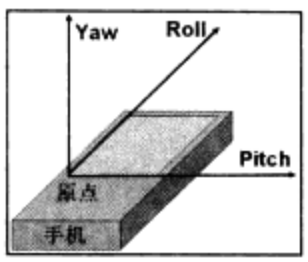
姿态传感器主要感应手机方位的变化，其每次读取的都是静态的状态值，在注册了传感器监听

器后，姿态传感器主要捕获 3 个参数 values[0]、values[1]和 values[2]，分别代表手机沿 Yaw 轴、Pitch 轴和 Roll 轴转过的角度。

研究姿态传感器的 Yaw、Pitch 和 Roll 轴所代表的含义之前，也先了解其与空间坐标系中的 x、y 和 z 轴的区别。当手机平放到桌面上时，姿态传感器的 3 个方向轴 Yaw、Pitch 和 Roll 如图 4-10 所示。



▲图 4-9　空间坐标系 xyz 轴示意图



▲图 4-10　姿态传感器的 3 个方向轴示意图

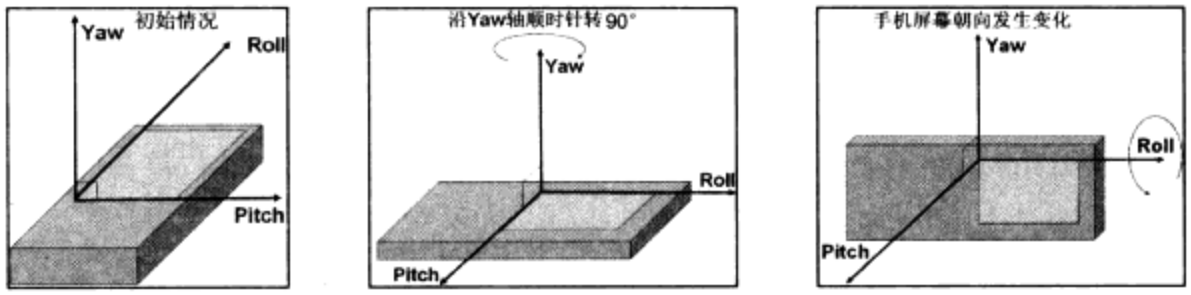
对比图 4-9 和图 4-10，会发现 Yaw、Pitch 和 Roll 轴和空间坐标系中的 z、x、y 轴是等同的，但是当手机不再处于这种初始状态时，姿态传感器的 3 个方向轴就和空间坐标系中的 3 个轴大不一样了。

- Yaw 轴

Yaw 是 3 个方向轴中惟一不变的轴，其方向总是竖直向上，和空间坐标系中的 z 轴是等同的，也就是重力加速度 g 的反方向。

- Pitch 轴

Pitch 轴方向依赖于手机沿 Yaw 轴的转动情况，即当手机沿 Yaw 转过一定角度后，Pitch 轴也相应围绕 Yaw 轴转过相同的角度，如图 4-11 所示。Pitch 轴的位置依赖于手机沿 Yaw 轴转过的角度，好比 Yaw 轴和 Pitch 轴是两根焊死在一起成 90° 夹角的铁棍。



▲图 4-11　关于 Pitch 轴的解说

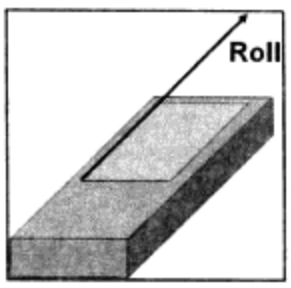
> **提示**　要特别注意的是，Pitch 轴只与手机沿 Yaw 轴转动的角度有关，与手机的屏幕朝向无关。如图 4-11 中最右边所示，当手机屏幕朝向发生变化，但没有沿 Yaw 轴转动时，Pitch 轴不发生变化。也就是说，Pitch 轴与手机间的相对关系是不固定的。

- Roll 轴

Roll 轴的方向虽然依赖于手机沿 Yaw 轴和 Pitch 轴转动的情况，但是 Roll 轴的确定并不复杂，Roll 轴与手机之间的相对关系是固定的。其总是沿着手机屏幕长边正方向，如图 4-12 所示。

> **提示**　由于 Roll 轴与手机之间的相对关系是固定的，所以在手机围绕 Yaw 轴或 Pitch 轴转动的过程中，Roll 轴必然也会跟随手机一起转动，而不是永远保持着与空间坐标系中 y 轴相同的方向。

看了前面的介绍读者可能会觉得 Yaw、Pitch、Roll 轴的定义很奇怪，这确实与人们习惯的方位系统有所不同。这一套方位系统其实是借鉴了飞行员驾驶飞机时使用的方位系统，具体如下。



▲图 4-12　Roll 轴示意图

- Yaw 轴永远是竖直向上的方向，因为在驾驶飞机的时候无论飞机如何旋转、翻滚，飞行员都必须确定向上的绝对方向。
- Pitch 轴会随着 Yaw 轴旋转，实际上 Pitch 轴方向可以通过推算表示飞机的朝向，绕 Pitch 轴旋转表示飞机上拉（抬起机头）或下推（机头向下）。
- Roll 轴表示贯穿机身的轴，与飞机是相对静止的，飞机绕 Roll 轴旋转表示左倾或右倾。

提示　之所以选择飞行员的方位系统估计和美国人出门乘坐飞机机会多，大量人持有私人飞行驾照有关，读者也要逐渐习惯。简而言之，读者应把手机想象成你驾驶的飞机，把你想象成飞机上的飞行员即可。

3. 其他传感器

除了前面介绍到的常用的加速度传感器和姿态传感器外，Android 平台下还提供了许多其他类型的传感器，下面简单介绍一下这些传感器。

- 磁场传感器（Magnetic Field）

磁场传感器主要用于感应周围的磁感应强度，注册监听器后其主要捕获 3 个参数：values[0]、values[1]和 values[2]。3 个参数分别代表磁感应强度在空间坐标系中 3 个方向轴上的分量。所有数据的单位为 uT，即微特斯拉。

- 光传感器（Light）

光传感器用于感应周围的光强，注册监听器后只捕获一个参数：values[0]。该参数代表周围的光照强度，单位为勒克斯（lux）。

- 距离传感器（Proximity）

距离传感器用于感应周围靠近的物体，在注册监听器后只捕获一个参数：values[0]。该参数代表靠近传感器的物体距离，以厘米为单位。

4.2.3　传感器应用开发示例

介绍完常用的传感器及其捕获的参数含义，下面通过一个例子来说明如何获取加速度传感器变化时的数据，其他传感器获取数据方式与之类似，只是对取到的数据后的处理方式有所不同。本例程序的开发步骤如下。

（1）首先，在程序的布局文件 main.xml 中声明要显示的 TextView 控件，其代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_6/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent" android:layout_height="fill_parent"
```

```
5        >                    <!-- 声明一个 LinearLayout 线性布局 -->
6      <TextView android:id="@+id/tvX"
7          android:layout_width="fill_parent" android:layout_height="wrap_content"
8          />                   <!-- 声明一个 TextView 控件，id 为 tvX -->
9      <TextView android:id="@+id/tvY"
10         android:layout_width="fill_parent" android:layout_height="wrap_content"
11         />                   <!-- 声明一个 TextView 控件，id 为 tvY -->
12     <TextView android:id="@+id/tvZ"
13         android:layout_width="fill_parent" android:layout_height="wrap_content"
14         />                   <!-- 声明一个 TextView 控件，id 为 tvZ -->
15   </LinearLayout>
```

说明 上述代码第 6～14 行定义了 3 个 TextView 控件，用于显示输出传感器捕获到的数据。

（2）本例中包含一个 Activity，其功能是为加速度传感器注册监听器，并在传感器数据发生变化时，把变化后的数据显示到 TextView 控件中，其主要代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_6/src/wyf/wpf 目录下的 Sample_4_6. java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.hardware.Sensor;                           //引入相关类
4    import android.hardware.SensorEvent;                      //引入相关类
5    import android.hardware.SensorEventListener;              //引入相关类
6    import android.hardware.SensorManager;                    //引入相关类
7    import android.os.Bundle;                                 //引入相关类
8    import android.widget.TextView;                           //引入相关类
9    //继承自 Activity 的子类
10   public class Sample_4_6 extends Activity {
11         SensorManager mySensorManager;                      //SensorManager 对象引用
12         TextView tvX;                                       //TextView 对象引用
13         TextView tvY;                                       //TextView 对象引用
14         TextView tvZ;                                       //TextView 对象引用
15           @Override
16           public void onCreate(Bundle savedInstanceState) {
17             super.onCreate(savedInstanceState);
18             setContentView(R.layout.main);
19             tvX = (TextView)findViewById(R.id.tvX);         //获得屏幕上 TextView 控件的引用
20             tvY = (TextView)findViewById(R.id.tvY);         //获得屏幕上 TextView 控件的引用
21             tvZ = (TextView)findViewById(R.id.tvZ);         //获得屏幕上 TextView 控件的引用
22             mySensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
23           }
24         ……//在此省略 onResume 方法代码，将在随后补全
25         ……//在此省略 onPause 方法代码，将在随后补全
26         ……//在此省略实现了 SensorEventListener 接口的监听器代码，将在随后补全
27   }
```

- 代码第 11 行声明了一个 SensorManager 对象，该对象是在代码第 22 行通过 Context 对象的 getSystemService 方法获得的。
- 代码第 12～14 行声明了 3 个 TextView 对象，在第 19～21 行通过 findViewById 获得屏幕上 3 个 TextView 对象的引用。

（3）创建了 SensorManager 对象只是第一步，下面是实现 SensorEventListener 接口的监听器的代码，步骤（1）中代码中第 26 行省略掉的代码如下。

```
1    private SensorListener mySensorListener = new SensorListener(){
2         @Override
3         public void onAccuracyChanged(int sensor, int accuracy) {
                                                        //重写 onAccuracyChanged 方法
4         }
5         @Override
6         public void onSensorChanged(int sensor, float[] values) {
                                                        //重写 onSensorChanged 方法
7              if(sensor == SensorManager.SENSOR_ACCELEROMETER){
                                                        //判断是否为加速度传感器变化产生
8                   tvX.setText("x 轴方向上的加速度为: "+values[0]);
                                                        //将提取的数据显示到 TextView
9                   tvY.setText("y 轴方向上的加速度为: "+values[1]);
                                                        //将提取的数据显示到 TextView
10                  tvZ.setText("z 轴方向上的加速度为: "+values[2]);
                                                        //将提取的数据显示到 TextView
11             }
12        }
13   };
```

- 上述代码创建了一个实现 SensorListener 接口的传感器监听器对象，主要重写的方法是 onAccuracyChanged 方法和 onSensorChanged 方法。
- 第 5～12 行为重写的 onSensorChanged 方法，首先判断产生传感器事件的来源是否为加速度传感器，随后简单地从 values 数组中读取数值显示到 Activity 的 TextView 控件中去，这些操作未来会被复杂的业务逻辑所替代。

（4）创建好监听器对象之后，剩下的工作就是在适当的位置注册和取消注册监听器了，在 Activity 主要代码中第 24 和 25 行省略掉的代码如下。

```
1    @Override
2    protected void onResume() {                                    //重写 onResume 方法
3         mySensorManager.registerListener(                         //注册监听器
4                   mySensorListener,                               //监听器对象
5                   SensorManager.SENSOR_ACCELEROMETER,             //传感器类型
6                   SensorManager.SENSOR_DELAY_UI                   //传感器事件传递的频度
7                   );
8         super.onResume();
9    }
10   @Override
11   protected void onPause() {                                     //重写 onPause 方法
12        mySensorManager.unregisterListener(mySensorListener);    //取消注册监听器
13        super.onPause();
14   }
```

说明　代码第 6 行传入的参数为 Android 系统传递传感器事件的频度，这里设定的频度只是建议值，一般来讲系统真正的频度比传入的设置值要高。

4.2.4　使用 SensorSimulator 模拟传感器变化

按照 4.2.3 小节所介绍的步骤开发出程序后，在 Android 提供的手机模拟器上运行会发现屏幕什么都没有显示，这是因为想要获取传感器的数据，必须使手机的传感器感应到变化，而如果在 Android 提供的 Emulator 而不是真机上调试程序，是无法让手机模拟器产生诸如加速度改变、姿态

翻转等变化的。

为了能够在手机模拟器上也能调试包含了传感器的应用程序，需要使用一个小工具 SensorSimulator。该工具开源免费，读者可以从 http://code.google.com/p/openintents/wiki/SensorSimulator 找到下载链接。

下载下来的文件为 sensorsimulator-1.0.0-beta1.zip，将该压缩包解压到磁盘任意目录，本例中为 C:\sensorsimulator-1.0.0-beta1 目录。下载完毕并将程序解压到指定目录后，需要进行如下步骤才可以使用该工具模拟传感器的变化。

- 在手机模拟器上安装 SensorSimulatorSettings.apk。
- 在 Eclipse 中为项目添加 jar 包。
- 修改程序的源代码。
- 启动 sensorsimulator.jar 和手机模拟器上的 SensorSimulatorSettings 进行必要的配置。
- 运行修改过的程序，在启动后的 sensorsimulator.jar 程序中模拟传感器变化。

1. 在手机模拟器上安装 SensorSimulatorSettings.apk

在 Android 模拟器运行的情况下，打开 CMD 命令行窗口，使用“cd”命令定位当前目录到 C:\sensorsimulator-1.0.0-beta1\bin 下。该目录下有一个 SensorSimulatorSettings.apk 文件，输入下述命令将 SensorSimulatorSettings.apk 安装到模拟器中。

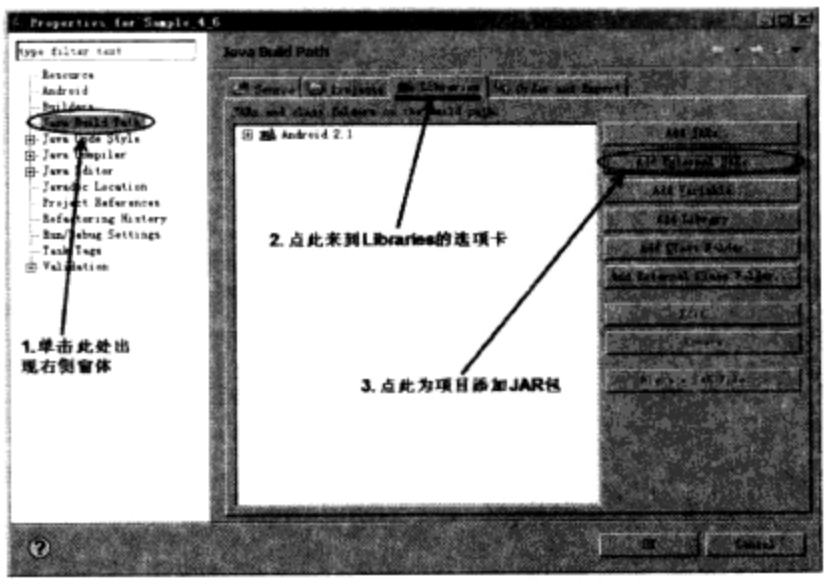
```
adb install SensorSimulatorSettings.apk
```

输入命令后，如果出现如图 4-13 所示的提示信息，则表明已成功安装 APK 文件到模拟器中。

2. 在 Eclipse 中为项目添加 jar 包

在 Eclipse 的 Package Explorer 中找到 Sample_4_6，在项目名称上单击鼠标右键，在弹出的菜单中选择“Properties”选项，弹出如图 4-14 所示的窗口。

▲图 4-13　成功安装 APK 文件的窗口



▲图 4-14　单击 Properties 选项弹出的窗口

在弹出的窗口左侧单击“Java Build Path”，然后从右侧的窗体中选择“Libraries”标签，在其

选项卡中单击 Add External JARs... 按钮，在随后弹出的“JAR Selection”对话框中找到 C:\sensorsimulator-1.0.0-beta1\lib 目录下的 sensorsimulator-lib.jar，将其添加到项目中。

3. 修改程序的源代码

使用 SensorSimulator 来模拟传感器变化需要对正常程序的源代码进行修改，本例中将会对 4.2.3 小节中编写好的项目 Sample_4_6 进行修改，使其能够利用 SensorSimulator 模拟传感器变化。需要修改的地方有如下几处。

- 声明 SensorManager

使用 SensorSimulator 时不再声明 SensorManager 类对象，而是 SensorManagerSimulator 对象，即将代码中的“SensorManager mySensorManager;”替换为如下代码。

```
1    SensorManagerSimulator mySensorManager;      //声明 SensorManagerSimulator 对象
```

- 获得 SensorManager 对象

既然对象的声明类型已经不同，获得的方式也不会相同，将代码中的“mySensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);”替换成如下代码。

```
1    mySensorManager = SensorManagerSimulator.getSystemService(this, SENSOR_SERVICE);
2    mySensorManager.connectSimulator();          //与 Simulator 服务器连接
```

- 修改 AndroidManifest.xml 文件

使用 SensorSimulator 时需要进行网络通信，所以必须在 AndroidManifest.xml 中为应用程序声明 Internet 的权限，实现代码如下。

```
1    <!-- 为应用程序添加 Internet 权限，否则将无法与 SensorSimulator 进行通信 -->
2    <uses-permission android:name="android.permission.INTERNET"/>
```

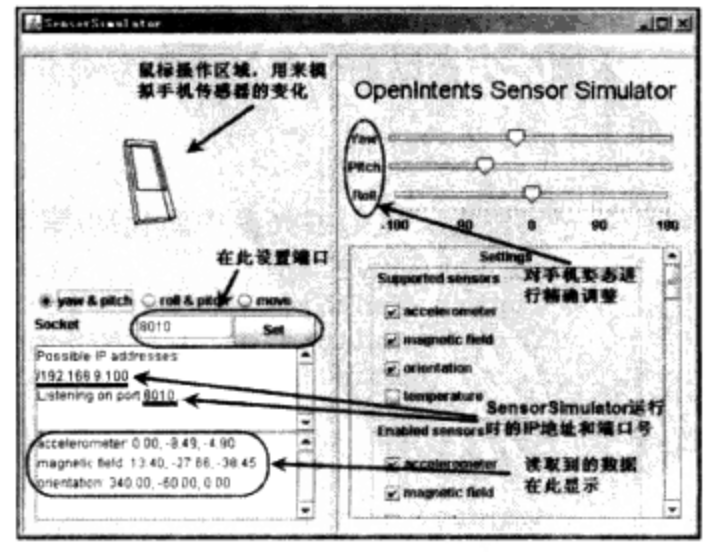
说明　以上 3 处代码的修改在随书光盘中的源代码/第 4 章/Sample_4_6/src/wyf/wpf 目录下的 Sample_4_6.java 文件中已在需要修改的地方用注释标明。可以自行查阅随书光盘。

4. 启动 sensorsimulator.jar 和手机模拟器上的 SensorSimulatorSettings 进行必要的配置

在 C:\sensorsimulator-1.0.0-beta1\bin 目录下找到 sensorsimulator.jar 并启动，运行后的界面如图 4-15 所示。

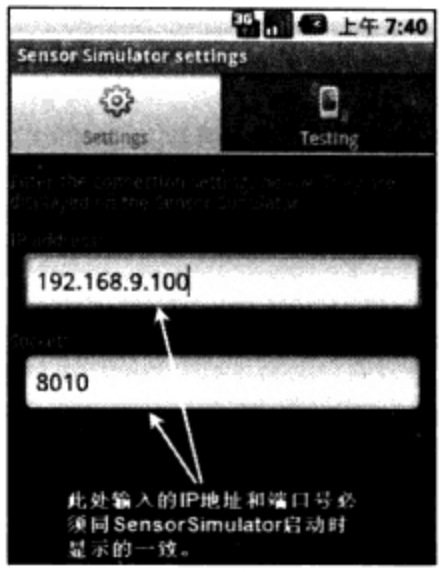
SensorSimulator 程序的主要功能和用法已在图 4-15 中说明，下面进行手机模拟器和 Sensor Simulator 的连接配置工作，运行手机模拟器上安装好的 SensorSimulatorSettings.apk，如图 4-16 所示。

在图 4-16 中输入 SensorSimulator 启动时显示的 IP 地址和端口号，单击屏幕右上角的“Testing”标签，转到如图 4-17 的测试连接界面。单击屏幕上的“Connect”按钮，进入如图 4-18 所示的界面。

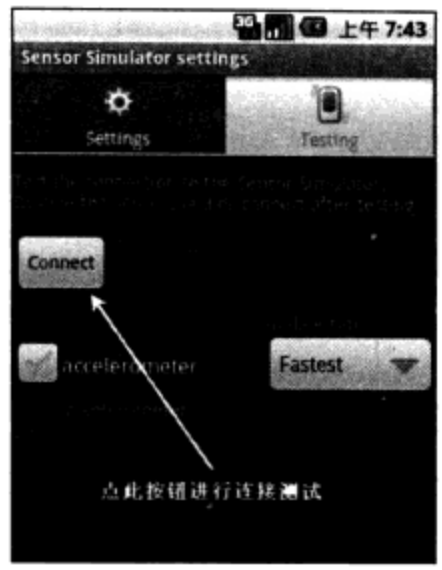


▲图 4-15　sensorsimulator.jar 运行界面

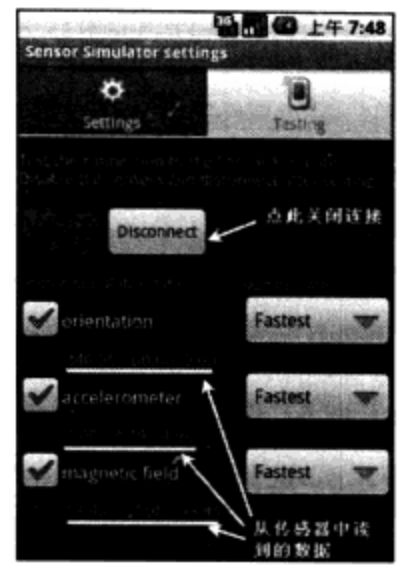
在图 4-18 所示的界面里选择需要监听的传感器，如果能够从传感器中读取到数据，说明 SensorSimulator 与手机模拟器连接成功，可以测试自己开发的应用程序了。



▲图 4-16　SensorSimulatorSettings.apk 配置界面



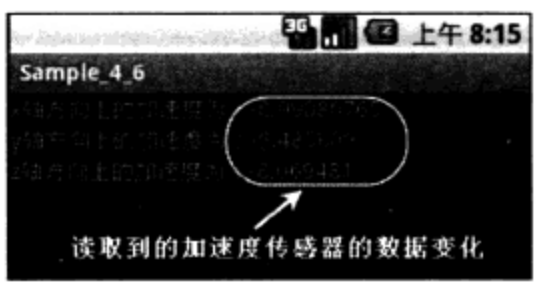
▲图 4-17　SensorSimulatorSettings.apk 测试连接界面 1



▲图 4-18　SensorSimulatorSettings.apk 测试连接界面 2

5. 运行修改过的程序，在启动后的 SensorSimulator.jar 程序中模拟传感器变化

测试过 SensorSimulator 与手机模拟器的连接之后，退出 SensorSimulatorSettings.apk，运行修改过的 Sample_4_6，同时在 SensorSimulator 程序中通过鼠标拖动左上角区域的手机或拖动屏幕右侧的姿态传感器滑块，可以模拟手机的各种动作，如翻转、上下移动等，这些操作带来的加速度传感器的变化都会以数据的形式显示在 Sample_4_6 的屏幕上，如图 4-19 所示。



▲图 4-19　修改过的 Sample_4_6 程序运行界面

4.2.5　使用新版本的 API 开发传感器应用

在 4.2.3 小节和 4.2.4 小节中，为了配合 SensorSimulator 的使用，在开发传感器应用时使用了较低版本的 API，原因就是 SensorSimulator 并没有为较高的 API 版本推出自己的更新版本，两个版本的区别不是很大，本小节就来简单介绍一下如何使用较高版本的 API 开发传感器应用。

本例中仍然以加速度传感器为例介绍如何获取传感器发生变化时的数据。Activity 类的主要代码如下。

代码位置：见随书光盘中源代码/第 4 章/Sample_4_7/src/wyf/wpf 目录下的 Sample_4_7. java。

```
1    package wyf.wpf;                                    //声明包语句
2    import android.app.Activity;                        //引入相关类
3    import android.hardware.Sensor;                     //引入相关类
4    import android.hardware.SensorEvent;                //引入相关类
5    import android.hardware.SensorEventListener;        //引入相关类
6    import android.hardware.SensorManager;              //引入相关类
7    import android.os.Bundle;                           //引入相关类
8    import android.widget.TextView;                     //引入相关类
9    //继承自 Activity 的子类
10   public class Sample_4_7 extends Activity {
```

```
11          SensorManager mySensorManager;                    //SensorManager 对象引用
12          TextView tvX;                                     //TextView 对象引用
13          TextView tvY;                                     //TextView 对象引用
14          TextView tvZ;                                     //TextView 对象引用
15            @Override
16            public void onCreate(Bundle savedInstanceState) {
17                super.onCreate(savedInstanceState);
18                setContentView(R.layout.main);
19                tvX = (TextView)findViewById(R.id.tvX);     //获得屏幕上 TextView 控件的引用
20                tvY = (TextView)findViewById(R.id.tvY);     //获得屏幕上 TextView 控件的引用
21                tvZ = (TextView)findViewById(R.id.tvZ);     //获得屏幕上 TextView 控件的引用
22                mySensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
23            }
24          ……//在此省略 onResume 方法代码，将在随后补全
25          ……//在此省略 onPause 方法代码，将在随后补全
26          ……//在此省略实现了 SensorEventListener 接口的监听器代码，将在随后补全
27      }
```

上述代码与 Sample_4_6 的代码类似，故在此不再赘述，新旧版本 API 的主要不同在于传感器的监听器对象的创建，上述代码 26 行省略掉的代码如下。

```
1     private SensorEventListener mySensorEventListener = new SensorEventListener(){
2           @Override
3           public void onAccuracyChanged(Sensor sensor, int accuracy) {
                                                              //重写 onAccuracyChanged 方法
4           }
5           @Override
6           public void onSensorChanged(SensorEvent event) {
7                 if(event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
                                                              //判断是否为加速度传感器变化
8                       float [] values = event.values;       //获得捕获的数据
9                       tvX.setText("x 轴方向上的加速度为: "+values[0]);
                                                              //将提取的数据显示到 TextView
10                      tvY.setText("y 轴方向上的加速度为: "+values[1]);
                                                              //将提取的数据显示到 TextView
11                      tvZ.setText("z 轴方向上的加速度为: "+values[2]);
                                                              //将提取的数据显示到 TextView
12                }
13          }
14    };
```

- 在新版本中已不再使用 SensorListener 而是使用 SensorEventListener，该接口中需要实现的方法和接收的参数也和 SensorListener 有所不同，例如，标识传感器类型的不再是一个 int 类型的常量，而是一个封装好的 Sensor 对象。
- 新版本的 API 中的 onSensorChanged 方法接收的参数为一个 SensorEvent 对象，该对象中封装了如表 4-7 所示的信息。

表 4-7　　SensorEvent 对象的字段及其说明

字　段	说　明
sensor	Sensor 对象，代表传感器类型
accuracy	int 类型，传感器事件的精确度
timestamp	long 类型，事件发生的时间，以纳秒为单位
values	float 类型数组，其长度和内容因传感器类型的不同而变化

除了监听器对象的创建不同之外，注册和取消注册监听器的实现方式也略有差别。注册和取消注册监听器的代码如下。

```
1    @Override
2    protected void onResume() {                    //重写 onResume 方法
3      mySensorManager.registerListener(            //调用方法为 SensorManager 注册监听器
4              mySensorEventListener,               //实现了 SensorEventListener 接口的监听器对象
5              mySensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                                                    //Sensor 对象
6              SensorManager.SENSOR_DELAY_UI        //系统传递 SensorEvent 事件的频度
7              );
8      super.onResume();
9    }
10   @Override
11   protected void onPause() {                     //重写 onPause 方法
12     mySensorManager.unregisterListener(mySensorEventListener);
                                                    //取消注册监听器
13     super.onPause();
14   }
```

- 对于 registerListener 和 unregisterListener 方法，其接收的参数都不再是实现了 SensorListener 接口的监听器，而是实现了 SensorEventListener 接口的监听器对象。
- 代码第 3 行调用 registerListener 方法为 SensorManager 对象注册监听器，同 onAccuracyChanged 方法一样，对于传感器类型，该方法接收的参数不再是代表类型的 int 常量，而是通过 SensorManager 对象的 getDefaultSensor 方法获得 Sensor 对象。

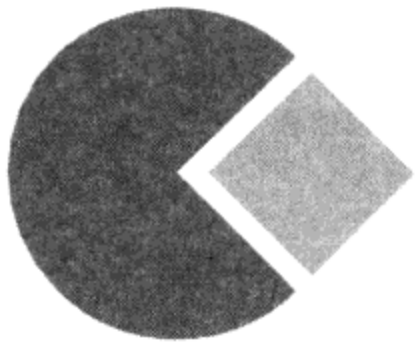
提示 应用程序的布局文件 main.xml 与 Sample_4_6 中的 main.xml 相比没有改变，故在此不予列出，读者可以自行查阅随书光盘。

总结一下，其实新旧版本 API 中主要的不同在于新版本使用 Sensor 对象取代旧版本中的 int 常量来标识传感器的类型，同时，新版本中使用 SensorEventListener 而不是 SensorListener 作为监听器。由于 SensorSimulator 工具无法和较高版本的 API 进行通信，故本书在此将不演示本例程序。

4.3 小结

本章主要介绍了在 Android 平台下如何进行数据的存储和共享。数据的存储包括文件存储、SQLite 数据库存储等；数据的共享方式有在应用程序间共享数据的 Content Provider，也有在应用程序内部的组件之间共享数据的 Preference，利用这些数据存储的处理方式使开发出的应用功能上更完备。

本章还介绍了 Android 平台的一大亮点——传感器应用的开发，并介绍了如何使用 Sensor Simulator 工具模拟传感器变化，这些内容将会在后面的案例中再次提到。



第 5 章　Android 游戏开发之网络编程

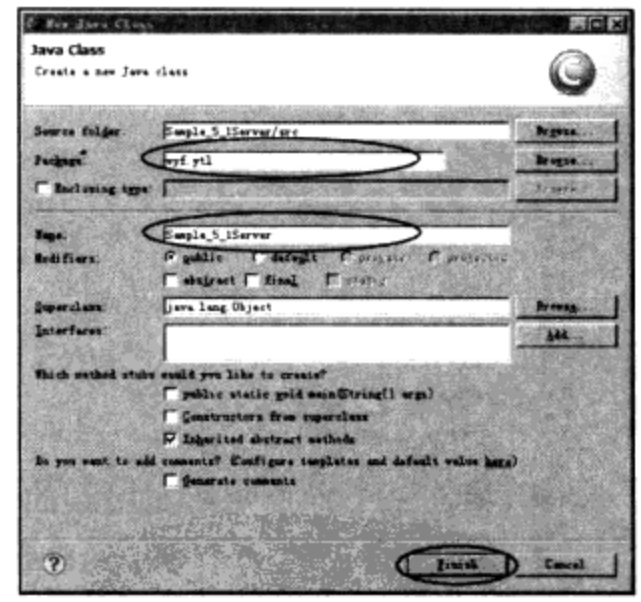
如今的游戏基本上已经离不开网络了，手机游戏也逐步向网络型发展，网络编程能力对于一个游戏开发人员来说是必不可少的。本章将对 Android 平台下的网络编程进行简单介绍，希望通过本章的学习，读者能够了解到 Android 平台下网络开发的特点。

5.1 基于 Socket 套接字的网络编程

说到网络开发，首先想到的一定是 Socket 编程，在游戏开发中，应用最多的网络编程技术也是 Socket，本节将通过一个简单的实例讲解如何在 Android 平台下完成基于 Socket 的网络应用开发，其具体开发步骤如下。

（1）首先是服务器端的开发，在 Eclipse 中新建一个名为 Sample_5_1Server 的普通 Java 项目，创建方法是在 Eclipse 中依次单击菜单“File/New/Java Project”。

（2）在新建的项目下的 src 上单击鼠标右键，在弹出的菜单中依次单击“New/Class”，在弹出的新建类的窗口中填入文件名 Sample_5_1Server 及包名 wyf.ytl，单击“Finish”按钮，如图 5-1 所示。



▲图 5-1　新建 Java 类

（3）开发 Sample_5_1Server 类，其代码如下。

代码位置：见随书光盘源代码/第 5 章/Sample_5_1Server/src/wyf/ytl 目录下的 Sample_5_1Server.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import java.io.DataInputStream;                           //引入相关类
3    import java.io.DataOutputStream;                          //引入相关类
4    import java.net.ServerSocket;                             //引入相关类
5    import java.net.Socket;                                   //引入相关类
6    public class Sample_5_1Server{
7         public static void main(String[] args){              //主方法
8              ServerSocket ss = null;                         //ServerSocket 的引用
9              Socket s = null;                                //Socket 的引用
10             DataInputStream din = null;
11             DataOutputStream dout = null;
12             try{
13                  ss = new ServerSocket(8888);               //监听到 8888 端口
```

```
14                  System.out.println("已监听到 8888 端口！");
15              }
16              catch(Exception e){
17                  e.printStackTrace();                    //打印异常信息
18              }
19              while(true){
20                  try{
21                      s = ss.accept();                    //等待客户端连接
22                      din = new DataInputStream(s.getInputStream());
23                      dout = new DataOutputStream(s.getOutputStream());
                                                            //得到输入输出流
24                      String msg = din.readUTF();         //读一个字符串
25                      System.out.println("ip: " + s.getInetAddress());
                                                            //打印客户端 IP
26                      System.out.println("msg: " + msg);   //打印客户端发来的消息
27                      System.out.println("====================");
28                      dout.writeUTF("Hello Client!");  //向客户端发送消息
29                  }
30                  catch(Exception e){
31                      e.printStackTrace();                //打印异常信息
32                  }
33                  finally{                                //用 finally 语句块确保动作执行
34                      try{
35                          if(dout != null){
36                              dout.close();               //关闭输出流
37                          }
38                          if(din != null){
39                              din.close();                //关闭输入流
40                          }
41                          if(s != null){
42                              s.close();                  //关闭 Socket 连接
43                          }
44                      }
45                      catch(Exception e){
46                          e.printStackTrace();            //打印异常信息
47                      }
48                  }
49              }
50          }
51      }
```

- 第 9～11 行声明后面用到的 DataInputStream 和 DataOutputStream 等对象的引用。
- 第 13 行创建一个 ServerSocket 并监听到 8888 端口。第 19 行进入死循环，等待接收客户端的连接请求。
- 第 21～23 行接收客户端的连接请求并根据连接的 Socket 创建输入/输出流。
- 第 24～26 行接收客户端发来的消息并打印，第 27 行打印分割线。
- 第 28 行向客户端发送字符串“Hello Client!”。
- 第 33～48 行关闭输入/输出流以及 Socket。

（4）接下来是客户端的开发，首先新建一个名为 Sample_5_1 的 Android 项目。

（5）为新建的 Sample_5_1 项目添加网络权限，打开 AndroidManifest.xml，在其</manifest>标记之前添加语句<uses-permission android:name="android.permission.INTERNET"/>。

（6）开发 main.xml，其代码如下。

代码位置：见随书光盘源代码/第 5 章/Sample_5_1/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       android:orientation="vertical"
4       android:layout_width="fill_parent"
5       android:layout_height="fill_parent"
6     >                                               <!--添加一个垂直的线性布局-->
7      <EditText
8          android:id="@+id/editText"
9               android:layout_width="fill_parent"
10              android:layout_height="wrap_content"
11              android:text="Hello Server!"
12     />                                              <!--添加一个输入文本框-->
13     <Button
14              android:id="@+id/button1"
15              android:layout_width="wrap_content"
16              android:layout_height="wrap_content"
17              android:text="发送信息到服务器"
18     />                                              <!--添加一个发送按钮 -->
19         <TextView
20              android:id="@+id/textView"
21              android:layout_width="wrap_content"
22              android:layout_height="wrap_content"
23              android:text="服务器发来的消息："
24         />                                  <!--添加一个文本控件显现服务器发来的信息-->
25   </LinearLayout>
```

- 第 2～6 行添加一个垂直的线性布局。
- 第 7～24 行向线性布局中添加 3 个控件，并为各个控件指定 ID。

（7）开发 Sample_5_1.java，用下面的代码代替 Sample_5_1.java 中原有的代码。

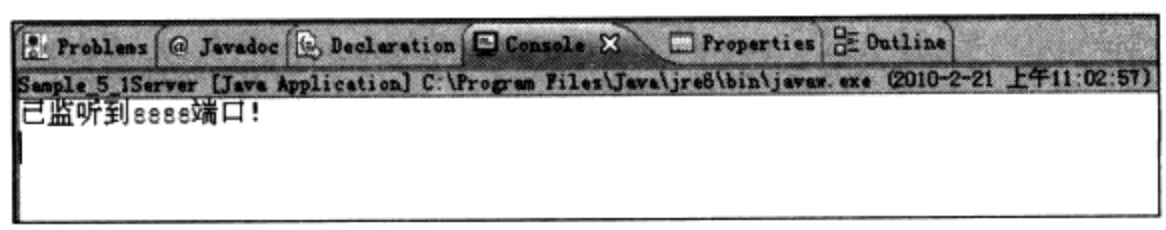
代码位置：见随书光盘源代码/第 5 章/Sample_5_1/src/wyf/ytl 目录下的 Sample_5_1.xml。

```
1    package wyf.ytl;
2    import java.io.DataInputStream;                    //引入 DataInputStream 类
3    import java.io.DataOutputStream;                   //引入 DataOutputStream 类
4    import java.net.Socket;                            //引入 Socket 类
5    import android.app.Activity;                       //引入 Activity 类
6    import android.os.Bundle;                          //引入 Bundle 类
7    import android.view.View;                          //引入 View 类
8    import android.view.View.OnClickListener;          //引入 OnClickListener 类
9    import android.widget.Button;                      //引入 Button 类
10   import android.widget.EditText;                    //引入 EditText 类
11   import android.widget.TextView;                    //引入 TextView 类
12   public class Sample_5_1 extends Activity implements OnClickListener{
13     Button button1;                                  // Button 对象的引用
14     EditText editText;                               // EditText 对象的引用
15     TextView textView;                               // TextView 对象的引用
16       public void onCreate(Bundle savedInstanceState) { //重写的 onCreate 回调方法
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.main);               //设置当前的用户界面
19         button1 = (Button) findViewById(R.id.button1);     //得到布局中的按钮引用
20         editText = (EditText) findViewById(R.id.editText);
                                                        //得到布局中的 EditText 引用
21         textView = (TextView) findViewById(R.id.textView);
                                                        //得到布局中的 TextView 引用
22         button1.setOnClickListener(this);            //添加监听
```

```
23      }
24    public void onClick(View v) {
25         Socket s = null;
26         DataOutputStream dout = null;
27         DataInputStream din = null;
28         if(v == button1){                                    //单击的是按钮
29             try {
30                 s = new Socket("192.168.9.102", 8888);       //连接服务器
31                 dout = new DataOutputStream(s.getOutputStream());
                                                                //得到输出流
32                 din = new DataInputStream(s.getInputStream());
                                                                //得到输入流
33                 dout.writeUTF(editText.getText().toString());
                                                                //向服务器发送消息
34                 textView.setText("服务器发来的消息: " + din.readUTF());
                                                                //接收服务器发来的消息
35             } catch (Exception e) {
36                 e.printStackTrace();                         //打印异常信息
37             } finally {
                                                        //用 finally 语句块确保动作执行
38                 try{
39                     if(dout != null){
40                         dout.close();                        //关闭输入流
41                     }
42                     if(din != null){
43                         din.close()     ;                    //关闭输入流
44                     }
45                     if(s != null){
46                         s.close();                           //关闭 Socket 连接
47                     }
48                 }
49                 catch(Exception e){
50                     e.printStackTrace();                     //打印异常信息
51                 }}}}}
```

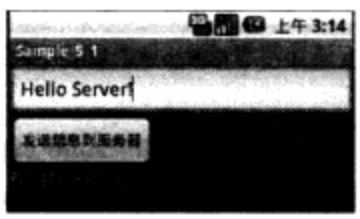
- 第 16～23 行重写了 onCreate 方法，在方法中得到各个控件的引用并为按钮添加监听器。
- 第 24 行实现了接口中的 onClick 方法。
- 第 30～32 行当单击按钮时，连接服务器的 8888 端口，并创建输入/输出流。
- 第 33～34 行先向服务器发送 editText 中的文本信息，再将从服务器接收到的信息通过 textView 显示出来。
- 第 37～51 行关闭输入/输出流及 Socket 连接。

（8）先运行 Sample_5_1Server，在 Console 窗口中会看到“已监听到 8888 端口！”的字样，说明服务器已经启动完毕，如图 5-2 所示。



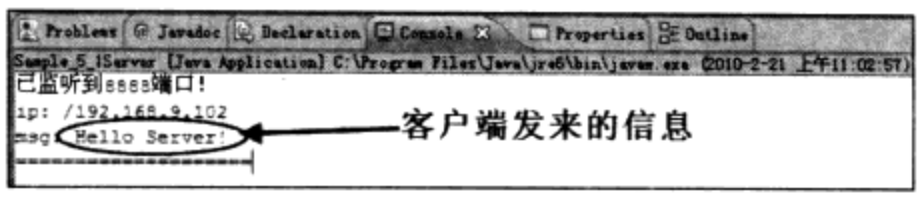
▲图 5-2　Console 窗口

（9）再运行 Sample_5_1，在模拟器中可看到如图 5-3 的效果，当单击按钮后，便观察到从服务器发送过来的信息，如图 5-4 所示，而在服务器端的 Console 窗口中可看到客户端发来的消息，如图 5-5 所示。



▲图 5-3　网络编程客户端效果图

▲图 5-4　客户端接收到服务器的消息



▲图 5-5　服务端的打印信息

5.2 基于 HTTP 协议的网络编程

前一小节已经对 Android 平台的 Socket 编程进行了介绍，本小节将对另一种网络编程技术 HTTP 协议进行简单介绍。

5.2.1　通过 URL 获取网络资源

HTTP 协议最简单的应用就是通过 URL 获取网络资源，下面的例子是在 Android 平台下，通过 URL 获取谷歌主页的源代码并显示在可滚动的 View 中，该例子的开发过程如下。

（1）新建一个名为 Sample_5_2 的 Android 项目。

（2）为该项目添加网络权限，打开 AndroidManifest.xml 文件，在</manifest>标记之前加上 <uses-permission android:name="android.permission.INTERNET"/>。

（3）开发 Sample_5_2.java 文件，打开 src/wyf/ytl 目录下的 Sample_5_2.java 文件，将原有代码替换成如下代码。

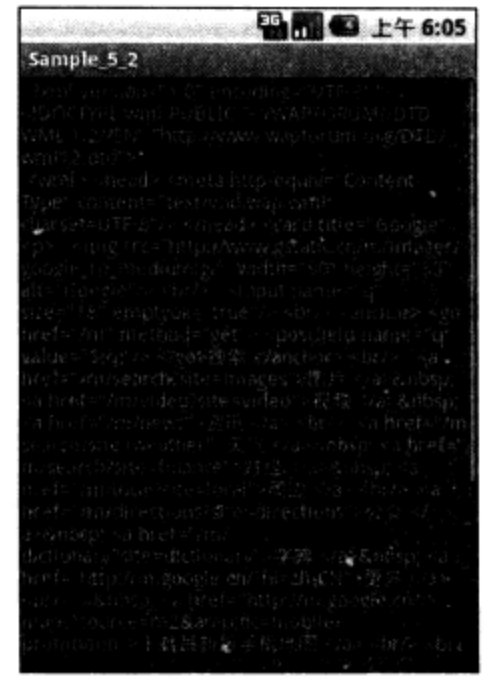
代码位置：见随书光盘源代码/第 5 章/Sample_5_2/src/wyf/ytl 目录下的 Sample_5_2.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import java.io.BufferedInputStream;                       //引入 BufferedInputStream 类
3    import java.net.URL;                                      //引入 URL 类
4    import java.net.URLConnection;                            //引入 URLConnection 类
5    import org.apache.http.util.ByteArrayBuffer;              //引入 ByteArrayBuffer 类
6    import org.apache.http.util.EncodingUtils;                //引入 EncodingUtils 类
7    import android.app.Activity;                              //引入 Activity 类
8    import android.os.Bundle;                                 //引入 Bundle 类
9    import android.widget.ScrollView;                         //引入 ScrollView 类
10   import android.widget.TextView;                           //引入 TextView 类
11   public class Sample_5_2 extends Activity {
12        TextView textView = null;                            //声明一个 TextView 的引用
13        ScrollView scrollView = null;                        //声明一个 ScrollView 的引用
14          public void onCreate(Bundle savedInstanceState) {  //重写的 onCreate 方法
15          super.onCreate(savedInstanceState);
16          textView = new TextView(this);                     //初始化 textView
17          scrollView = new ScrollView(this);                 //初始化 scrollView
18          URLConnection ucon = null;
```

```
19          BufferedInputStream bin = null;
20          ByteArrayBuffer bab = null;
21          try {
22               URL myURL = new URL("http://www.google.cn/");//初始化 URL
23               ucon = myURL.openConnection();                //打开连接
24               bin = new BufferedInputStream(ucon.getInputStream());
                                                               //通过连接得到输入流
25               int current = 0;
26               bab = new ByteArrayBuffer(1000);
27               while((current=bin.read()) != -1){
28                   bab.append((char)current);   //将收到的信息添加到 ByteArrayBuffer 中
29               }
30          } catch (Exception e) {
31               e.printStackTrace();             //打印异常信息
32          } finally {                           //用 finally 语句块确保动作执行
33               try{
34                    if(bin != null){
35                         bin.close();  bin=null;            //关闭输入流
36                    }
37               }
38               catch(Exception e){
39                    e.printStackTrace();                   //打印异常信息
40               }
41          }
42          textView.setText(EncodingUtils.getString(bab.toByteArray(),"UTF-8"));
                                                  //设置 textView 中的内容
43          scrollView.addView(textView);         //将 textView 添加到 scrollView 中
44          this.setContentView(scrollView);      //设置当前显示的用户界面为 scrollView
45     }
46 }
```

- 第 12～13 行声明 TextView 和 ScrollView 的引用。第 16～17 行初始化 TextView 和 ScrollView。
- 第 22 行初始化 URL 资源，在构造器中传入的参数为谷歌主页的 URL 地址。
- 第 23～24 行打开连接，并通过连接获得输入流。第 25～29 行通过输入流接收数据，并将接收到的数据添加到 ByteArrayBuffer 中。第 32～41 行关闭并释放输入流。
- 第 42 行将 ByteArrayBuffer 中的数据转换成 UTF-8 编码形式后设置到 textView 中。
- 第 43 行将 textView 添加到可滚动的 scrollView 中。
- 第 44 行将 scrollView 设置成当前显示的用户界面。

（4）运行该项目，在模拟器中观察其运行效果，如图 5-6 所示，可通过上下键或者鼠标滚动 scrollView 来查看更多信息。



▲图 5-6　获取谷歌主页源代码

提示　因为该例子需要通过网络获取主页的信息，所以在运行该例子时，应该保证所使用计算机的网络是良好的。

该例子中并没有像之前各章节的例子使用 XML 布局文件进行布局，而是直接在 Java 源代码中初始化各个 View，直接将初始化后的 View 设置成当前显示的用户界面。在游戏的开发过程中，这

种方式使用较普遍，这种方式更灵活、更简单。

5.2.2　在 Android 中解析 XML

前面已经对如何获取网络资源进行了讲解，接下来将介绍在 Android 中如何对在网络上获取的 XML 文件的内容进行解析，得到所需要的内容。

XML 是一种通用的数据交换格式，它的平台无关性、语言无关性、系统无关性为数据集成与交换带来了很大的方便，XML 文件的解析对 Android 来说必不可少。

Android 中解析 XML 的方式有两种，一种叫 DOM，另一种叫 SAX，下面将对这两种解析方式进行简单的介绍。

- DOM 方式。DOM 是一种基于 XML 文档树结构的解析，解析之前通常需要加载整个 XML 文档和构造树结构，之后的解析就是在 Java 代码中通过 DOM 接口操作之前构造的树结构。DOM 方式解析 XML 的优点是，整个 XML 的树结构存在于内存中，可以很方便地操作，而且可以很轻松地删除和修改 XML 树结构。但是 DOM 方式每次必须将整个 XML 文档调入内存，而往往只是对其中很小一部分进行操作，这就浪费了很多资源。
- SAX 方式。SAX 是一种基于事件流的解析方式，解析之前不需要加载整个 XML 文档，而是当解析器遇到某些标记时触发一些事件，开发人员编写响应这些事件的代码来做相应的处理。所以 SAX 方式解析 XML 比较迅速，而且节省资源。但因为在解析过程中并没有加载整个 XML，所以 SAX 方式解析的数据不持久，如果没有及时保存，很容易丢失。又因为 SAX 方法得到的文本并不知道属于哪个元素下的，所以操作有些局限。

提示

因为在游戏开发中，基本不会使用到 XML 的解析技术，所以在此并没有深入地讲解，有兴趣的读者可以查看相关资料进行学习与研究。

5.3　Android 平台下的 Google Map

Google 除了依靠超强的搜索引擎技术成就了整个企业的辉煌，还拥有许多酷炫实用的网络服务，Google Map 就是其中的一种。Google Map 的使用是完全免费的，开发人员只需简单的几句调用代码就可以将 Google Map 添加到自己的应用中。本节将详细介绍在 Android 平台中如何开发带有 Google Map 的应用程序。

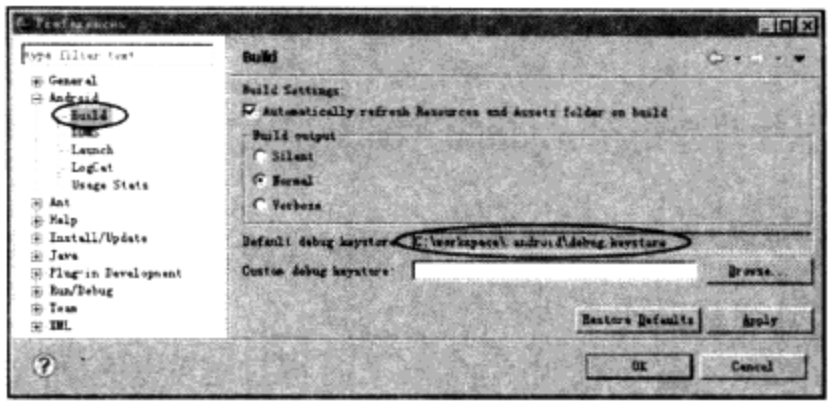
5.3.1　定位类 GenPoint 与显示地图类 MapView 的搭配使用

本节将通过一个实例讲解在 Android 平台下如何开发带有 Google Map 的应用程序。该实例中主要用到的两个类是 GenPoint 与 MapView，基本思路是根据用户输入的经度与纬度通过 GenPoint 来定位，然后通过 MapView 来显示地图的正确位置，该例子的详细开发步骤如下。

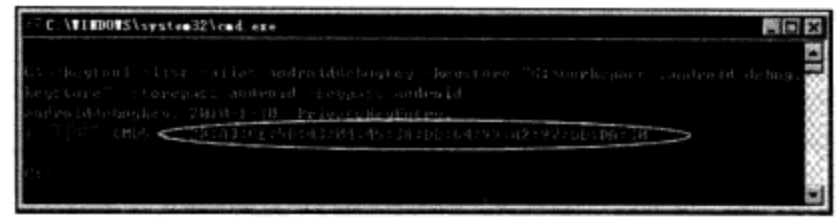
（1）启动 Eclipse，依次单击 Eclipse 的菜单“Window/Preferences/Android/Build”，得到 debug.keystore 的路径，如图 5-7 所示。

（2）启动 CMD 命令行，输入命令“keytool-list-alias androiddebugkey-keystore "C:\workspace\.

android\debug.keystore" -storepass android-keypass android”得到 MD5，如图 5-8 所示。



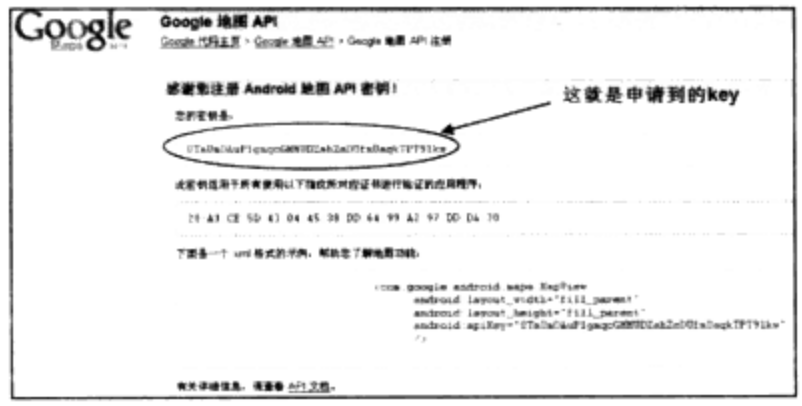
▲图 5-7　debug.keystore 的路径



▲图 5-8　得到 MD5

提示　keytool 为 JDK 安装目录下 bin 的一个工具，如果读者没有配置环境变量，需要将目录切换到 JDK 安装目录的 jre/bin 后再运行上述命令。

（3）用浏览器登录到 Google Map API signup 网站：http://code.google.com/android/maps-api-signup.html，输入之前得到的 MD5 编码，即可申请到由 Google 签署过的 key，复制得到的 key，在之后的程序中会用到，如图 5-9 所示。



▲图 5-9　获得 API Key

提示　在申请 Key 的过程中，需要登录 Google 账号，如果读者没有 Google 账号，可以到 https://www.google.com/accounts/NewAccount 网站注册一个。

（4）创建名为 Sample_5_3 的 Android 项目，在选择 Build Target 时应该选择 Google APIs 的 2.1 版。

（5）打开 AndroidManifest.xml 文件，声明该应用将应用地图库，声明方法是在</application>的上面添加语句<uses-library android:name="com.google.android.maps"/>。

（6）因为 Google Map 应用必须应用网络资源，所以应继续添加网络权限，在</manifest>标记的上面添加语句<uses-permission android:name="android.permission.INTERNET"/>。

（7）打开 res/layout 下的 main.xml 文件，将其代码替换成下列代码。

代码位置：见随书光盘源代码/第 5 章/Sample_5_3/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>      <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                          <!--定义一个线性布局 -->
7        <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
8             android:orientation="horizontal"
9             android:layout_width="fill_parent"
10            android:layout_height="wrap_content"
11       >                                          <!-- 添加一个水平的线性布局 -->
12         <TextView
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="经度"
16         />                                       <!--在水平的线性布局中添加 TextView -->
17         <EditText
18             android:id="@+id/editView1"
19         android:layout_width="wrap_content"
20         android:layout_height="wrap_content"
21         android:text="116.46"
22         />                                       <!--在水平的线性布局中添加 EditText -->
23     </LinearLayout>
24     <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
25         android:orientation="horizontal"
26         android:layout_width="fill_parent"
27         android:layout_height="wrap_content"
28     >                                            <!-- 添加一个水平的线性布局 -->
29           <TextView
30           android:layout_width="wrap_content"
31           android:layout_height="wrap_content"
32           android:text="维度"
33           />                                     <!--在水平的线性布局中添加 TextView -->
34           <EditText
35               android:id="@+id/editView2"
36           android:layout_width="wrap_content"
37           android:layout_height="wrap_content"
38           android:text="39.92"
39         />                             <!--在水平的线性布局中添加 EditText -->
40     <Button
41               android:id="@+id/button1"
42           android:layout_width="wrap_content"
43           android:layout_height="wrap_content"
44           android:text="查询"
45           />                             <!--在水平的线性布局中添加一个按钮 -->
46     /LinearLayout>
47     <com.google.android.maps.MapView
48               android:id="@+id/myMapView"
49               android:layout_width="fill_parent"
50               android:layout_height="wrap_content"
51               android:clickable="true"
52               android:apiKey="0TaUmOAuP1gmqcGMMUDZahZeDUfmUaqkTPT91kw"
53       />                                 <!--在水平的线性布局中添加 MapView 用来显示地图 -->
54   </LinearLayout>
```

- 第 1～46 行向垂直的线性布局中填入水平线性布局，然后再向水平线性布局中填入需要的控件，其原理与前面各个例子类似，在此就不再赘述。
- 第 47 行向线性布局中添加一个 MapView 控件。
- 第 51 行设置显示的地图是否支持触屏操作。

- 第 52 行设置之前申请的 API Key，该属性必须设置。

（8）打开 src/wyf/ytl 下的 Sample_5_3.java 文件，将其代码替换成如下代码。

代码位置：见随书光盘源代码/第五章/Sample_5_3/src/wyf/ytl 目录下的 Sample_5_3.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.os.Bundle;                                 //引入 Bundle 类
3    import android.view.View;                                 //引入 View 类
4    import android.widget.Button;                             //引入 Button 类
5    import android.widget.EditText;                           //引入 EditText 类
6    import com.google.android.maps.GeoPoint;                  //引入 GeoPoint 类
7    import com.google.android.maps.MapActivity;               //引入 MapActivity 类
8    import com.google.android.maps.MapController;             //引入 MapController 类
9    import com.google.android.maps.MapView;                   //引入 MapView 类
10   public class Sample_5_3 extends MapActivity {
11     MapView myMapView = null;                               //声明 MapView 的引用
12     MapController myMapController = null;                   //声明 myMapController 的引用
13     Button button1 = null;                                  //声明 Button 的引用
14     EditText editView1 = null;                              //声明 EditText 的引用
15     EditText editView2 = null;                              //声明 EditText 的引用
16     double jingDu = 116.46;
17     double weidu = 39.92;
18       public void onCreate(Bundle savedInstanceState) {
19           super.onCreate(savedInstanceState);
20           setContentView(R.layout.main);
21           myMapView = (MapView) this.findViewById(R.id.myMapView);
                                                             //得到 myMapView 的引用
22           button1 = (Button) this.findViewById(R.id.button1);
                                                             //得到 button1 的引用
23           editView1 = (EditText) this.findViewById(R.id.editView1);
                                                             //得到 editView1 的引用
24           editView2 = (EditText) this.findViewById(R.id.editView2);
                                                             //得到 editView2 的引用
25           myMapController = myMapView.getController();    //获得 MapController
26           setGeoPoint();                                  //设置地图的经纬度
27           button1.setOnClickListener(new Button.OnClickListener() {   //添加监听
28               public void onClick(View v) {
29                   jingDu= Double.parseDouble(editView1.getText().toString());
                                                             //取得 editView1 中的数据
30                   weidu = Double.parseDouble(editView2.getText().toString());
31                   setGeoPoint();                          //设置地图的经纬度
32               }
33           });
34       }
35       public void setGeoPoint(){                          //设置地图的经纬度的方法
36           GeoPoint gp = new GeoPoint((int)(weidu*1E6), (int)(jingDu*1E6));
37           myMapController.animateTo(gp);                  //设置经纬度
38           myMapController.setZoom(18);                    //设置放大等级
39       }
40     protected boolean isRouteDisplayed() {
41           return false;
42     }
43   }
```

- 第 11～15 行声明程序中会用到的各个对象的引用，第 21～25 行得到在 XML 中配置的各个对象的引用。
- 第 25 行通过 MapView 获得 MapController，MapController 是用来对地图进行设置的。
- 第 27～34 行为查询按钮添加监听，当单击按钮时，取得两个 EditText 中的数据设置经纬度。

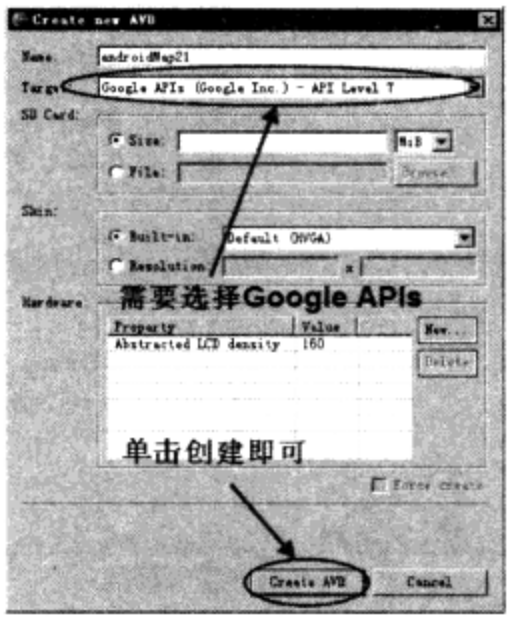
- 第 36 行根据经纬度的值创建 GeoPoint 对象。
- 第 37 行设置地图显示的经纬度，第 38 行设置地图的放大等级。

（9）还需重新创建一个 AVD，依次单击 Eclipse 菜单的“Window/Android SDK and AVD Manager/New”，在弹出的对话框中的 Target 项中需要选择“Google APIs”，如图 5-10 所示。

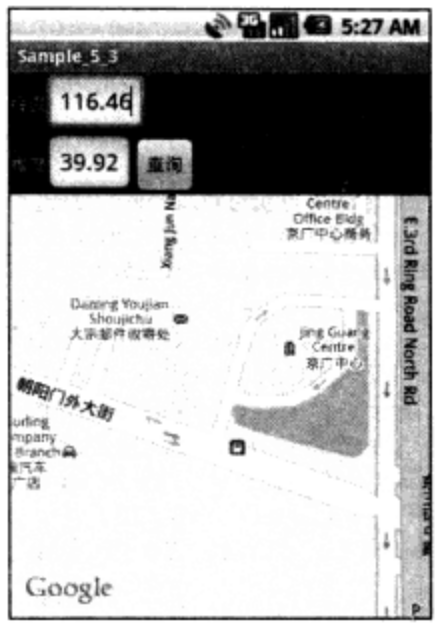
（10）在新建的 AVD 上运行该项目得到的效果如图 5-11 所示。

5.3.2　位置监听器——LocationListener 的使用

前面已经介绍了 Google Map 的基本使用方法，接下来的例子将介绍，如何通过 GPS 坐标的改变来时时更改地图显示的位置，即当手机的地理位置发生变化时，应用程序根据变化的经纬度重新定位地图的显示位置。



▲图 5-10　创建 Google APIs AVD



▲图 5-11　Google Map

具体开发步骤如下。

（1）创建名为 Sample_5_4 的 Android 项目。

（2）打开 AndroidManifest.xml 文件，声明该应用将应用地图库，声明方法是在</application>的上面添加语句<uses-library android:name="com.google.android.maps"/>。

（3）和前面小节的例子一样，还需要为应用程序添加网络权限，添加方法是在</manifest>标记的上面添加语句<uses-permission android:name="android.permission.INTERNET"/>。

（4）还需在 AndroidManifest.xml 文件中的</manifest>标记的前面添加语句<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>。

（5）打开 res/layout 下的 main.xml 文件，将其代码替换成下列代码。

代码位置：见随书光盘源代码/第 5 章/Sample_5_4/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>        <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                          <!--定义一个线性布局 -->
7        <com.google.android.maps.MapView
```

```
8                 android:id="@+id/myMapView"
9                 android:layout_width="fill_parent"
10                android:layout_height="wrap_content"
11                android:apiKey="0TaUmOAuPlgmqcGMMUDZahZeDUfmUaqkTPT91kw"
12        />                                        <!--在线性布局中添加MapView用来显示地图 -->
13    </LinearLayout>
```

- 第 2～6 行添加一个垂直的线性布局。
- 第 7～12 行向线性布局中添加一个 MapView。一定要指定 API Key，API Key 的值使用的还是前一节例子开发前申请的 key。

（6）打开 src/wyf/ytl 下的 Sample_5_4.java 文件，将其原有代码替换成如下代码。

代码位置：见随书光盘源代码/第 5 章/Sample_5_4/src/wyf/ytl 目录下的 Sample_5_4.java。

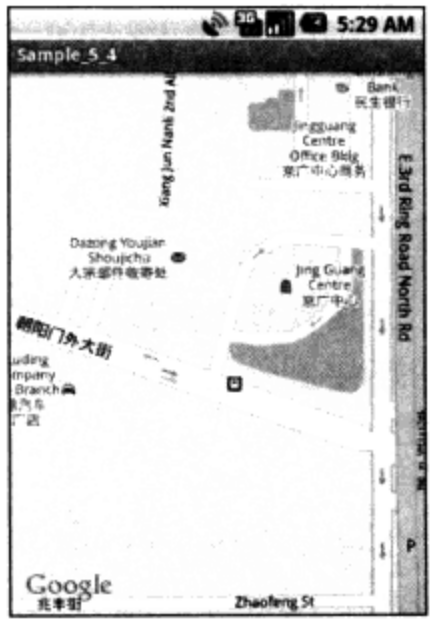
```
1     package wyf.ytl;                                          //声明包语句
2     import android.content.Context;                           //引入 Context 类
3     import android.location.Location;                         //引入 Location 类
4     import android.location.LocationListener;                 //引入 LocationListener 类
5     import android.location.LocationManager;                  //引入 LocationManager 类
6     import android.os.Bundle;                                 //引入 Bundle 类
7     import com.google.android.maps.GeoPoint;                  //引入 GeoPoint 类
8     import com.google.android.maps.MapActivity;               //引入 MapActivity 类
9     import com.google.android.maps.MapController;             //引入 MapController 类
10    import com.google.android.maps.MapView;                   //引入 MapView 类
11    public class Sample_5_4 extends MapActivity {
12      MapView myMapView = null;                               //声明 MapView 的引用
13      MapController myMapController = null;                   //声明 myMapController 的引用
14      LocationManager mLocationManager = null;
15      double jingDu = 116.46;
16      double weidu = 39.92;
17         public final LocationListener mLocationListener = new LocationListener(){
                                                               //监听器类
18         public void onLocationChanged(Location location){
19              weidu = location.getLatitude();
20              jingDu = location.getLongitude();
21              setGeoPoint();                                 //设置地图的经纬度
22         }
23         public void onProviderDisabled(String provider){     //空实现
24         }
25         public void onProviderEnabled(String provider){      //空实现
26         }
27         public void onStatusChanged(String provider, int status, Bundle extras){
                                                                //空实现
28         }
29     };
30     public void onCreate(Bundle savedInstanceState) {
31         super.onCreate(savedInstanceState);
32         setContentView(R.layout.main);
33         myMapView = (MapView) this.findViewById(R.id.myMapView);
                                                   //得到 myMapView 的引用
34         myMapController = myMapView.getController();         //获得 MapController
35         setGeoPoint();                                  //设置地图的经纬度
36                                                         //取得系统 LOCATION 服务
37         mLocationManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
38                                                         //监听 Location 更改时的事件,更新 MapView
39         mLocationManager.requestLocationUpdates(
40                   mLocationManager.GPS_PROVIDER, 2000, 10, mLocationListener);
```

```
41      }
42      public void setGeoPoint(){                                  //设置地图的经纬度的方法
43          GeoPoint gp = new GeoPoint((int)(weidu*1E6), (int)(jingDu*1E6));
44          myMapController.animateTo(gp);                          //设置经纬度
45          myMapController.setZoom(18);                            //设置放大等级
46      }
47     protected boolean isRouteDisplayed() {
48          return false;
49      }
50    }
```

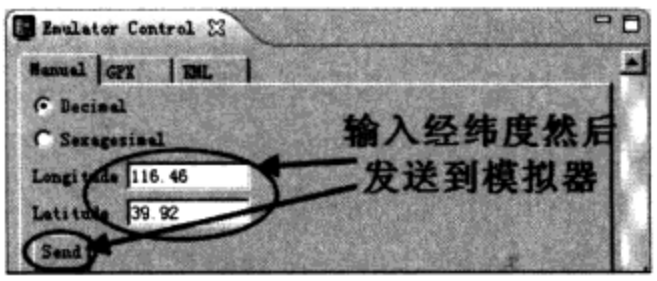
- 第 12～14 行声明程序中用到的各个对象的引用。
- 第 17～29 行创建一个 final 的监听器类，实现其 onLocationChanged 方法。当手机物理位置发生变化时会调用该方法，在方法中取得新的经纬度后重新设置地图显示的位置。
- 第 37 行取得系统的 LOCATION 服务。
- 第 39～40 行添加 Location 监听，当 Location 发生改变时更新 MapView。

（7）运行该程序，在模拟器中观察到的效果如图 5-12 所示。

（8）将 Eclipse 切换到 DDMS 状态，通过改变右下侧 Manual 选项卡中的 Longitude 和 Latitude 的值来模拟手机经纬度的变化，如图 5-13 所示。



▲图 5-12　GPS 与 Google Map 结合图



▲图 5-13　DDMS 中模拟 GPS 的经纬度

5.4 其他网络通信方式

Android 平台除了之前介绍过的网络通信方式以外，还有两种网络通信方式使用的比较频繁，一种是基于 WiFi 的网络通信，另一种是基于蓝牙的网络通信，本节将对这两种技术进行简单的介绍。

5.4.1　使用 WiFi 进行开发

WiFi（Wireless Fidelity）俗称无线宽带，其实也就是 IEEE 802.11 的统称，是一种短程无线传输技术，一般工作在 2.4Ghz 频段，所支持的速度最高达 11Mbit/s。支持 WiFi 的手机在覆盖无线网络的地方可以通过 WiFi 连接上网络，使手机与 PC 共享宽带上网，通过 WiFi 上网的手机与 PC 并没有本质区别。

WiFi 技术之所以备受亲睐，是因为它拥有如下突出的优势。

- 无线电波覆盖范围广。
- 传输速度快，理论上可达 54Mbit/s。
- 进入该领域的门槛低，无需大量资金即可覆盖很多地方。
- 无需布线，不用考虑布线的条件限制。
- 健康安全，其发射功率不会超过 100mW，而普通手机的发射功率在 200mW～1W。

WiFi 是目前无线接入的主流标准，而在 Intel 公司的强力支持下，已经出现全面兼容 WiFi 的 WiMAX(802.16x)，与现有 WiFi 相比 WiMAX 拥有更远的传输距离、更宽的频段选择、更高的接入速度等优点，预计在未来的几年中 WiMAX 技术将会成为无线网络的主流技术。

通过使用 WiFi 技术进行开发与普通的 Socket 开发完全一样，开发人员无需了解应用层之下是通过 WiFi 连接的还是其他什么方式，只考虑应用层之间的 Socket 连接即可。

5.4.2 借助于蓝牙（Bluetooth）技术进行开发

蓝牙是由瑞典爱立信公司于 1994 年发起研发的一种支持设备短距离通信的无线电技术。它能在手机、PDA、无线耳机、笔记本电脑等设备之间进行无线信息交换。采用的是分散式网络结构以及快跳频和短包技术，支持点对点或者是点对多点的通信，一般工作在 2.4Ghz ISM 频段，所支持的传输速度为 1Mbit/s，并采用时分双工传输方案实现全双工传输。

蓝牙技术的广泛应用主要取决与它所拥有的如下优势。

- 全球可用，使用 Bluetooth 技术不需要支付任何费用。
- 设备选择范围广，从手机、汽车到医疗设备都集成了该技术。
- 易于使用，不需要固定的基础设施，易于安装和配置，只要拥有支持该技术的设备即可。
- 全球通用的规格，符合标准的蓝牙设备都可以轻易连接到一起。

开发基于蓝牙技术的网络程序步骤如下。

（1）为应用程序添加蓝牙通信权限以及蓝牙开关权限。

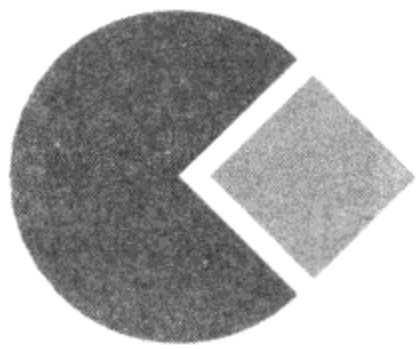
（2）搜索附近可用的蓝牙设备。

（3）建立连接，蓝牙的通信分为服务器端和客户端，分别使用 BluetoothSocket 类的不同方法来获取数据，作为服务器端。通过 BluetoothServerSocket 类来处理客户端发来的信息，服务端套接字在接受（accepted）一个客户端发来的 BluetoothSocket 时做出回应。而作为客户端应先获取本地的 BluetoothDevice 对象，再连接服务器。

（4）进行通信，这就和其他网络通信方式没有任何区别了，同样是通过 IO 进行通信和传输。

5.5 小结

本章对 Android 的网络编程进行了详细介绍，通过本章的学习，读者应该了解了开发网络应用程序的思路。基于 Socket 的网络编程技术一般应用于开发网络游戏、简单的商业程序等。基于 HTTP 协议的网络编程技术则主要应用于与网络服务器进行交互的应用程序，如手机浏览器等。本章还介绍了 Google Map 的应用程序的开发，让读者学会运用 Google Map 技术构建属于自己的地图程序。



第6章 不一样的游戏，一样的精彩

前面的章节介绍了 Android 应用程序开发的相关知识，而在形形色色的手机应用中，游戏占了很大的比重。目前，随着手持设备的性能逐步提升和移动网络的不断完善，手机游戏带给玩家的体验也越来越好，手机游戏已经成长为和电脑游戏同等重要的产业了。

当今市面上流行的手机游戏类型繁多，不同类型的游戏自然有其独特的设计方式以及独到的吸引玩家的地方，本章将结合目前手机游戏产业的现状介绍一些与游戏开发相关的知识，对主要的几种游戏类型做一个简单的介绍。

6.1 射击类游戏

射击类游戏（Shooting Game）是一种比较古老的游戏类型，手机游戏中的射击游戏也很流行，目前市面上的射击类游戏最多的是飞行射击游戏，比较著名的有雷电系列，还有一些是诸如坦克大战之类的操作性要求较高的射击游戏，本小节就来简单介绍一下射击类手机游戏的相关知识。

6.1.1 游戏玩法

下面从玩家人数、操作方式和取胜条件等几个方面分析射击类游戏的玩法。

- 玩家人数

射击类游戏通常为单人游戏，很少以二人对战或多人在线的方式进行，而且一般来说，射击游戏节奏比较快，要求玩家通过快速的反应与游戏进行交互，所以射击游戏大都属于单机游戏。

- 操作方式

射击类游戏的操作方式比较单一，主要是控制游戏角色的行走方向以及向目标开火或施放特殊技能，有些射击游戏为了提高游戏速度，会让玩家控制的角色自动射击或提供选项让玩家选择是否开启自动射击，自动射击尤其在飞行类射击游戏中比较普遍。

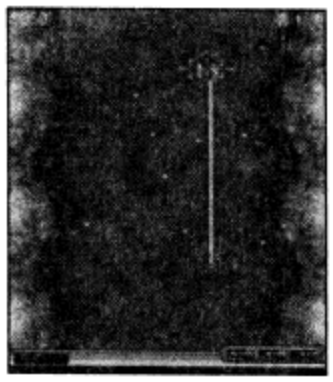
- 取胜条件

射击类游戏一般在游戏开始时会为玩家分配若干条生命用以进行后续的游戏，当耗费光所给的生命数目后就会结束游戏。射击类游戏多数为关卡类游戏，即玩家用有限的生命挑战难度不断提升的关卡，关卡一般也是有限的，但是有些游戏的关卡可以在程序中生成。

6.1.2 视觉效果

- 视角问题

对于占多数的飞行类射击游戏来说，其视角几乎都是 2D 视角，玩家从高空俯瞰整个游戏界面，如著名的雷电系列（如图 6-1 所示）的游戏就是这样。还有一些游戏会像“永远的坦克大战”（如图 6-2 所示）这样采用 2.5D 视角进行游戏。



▲图 6-1　飞行射击游戏“雷电”的 2D 视角

还有一些射击类游戏正逐步向桌面电脑中的第一人称视角的射击游戏靠拢，如比较著名的电脑游戏“反恐精英”、“荣誉勋章”和“使命召唤”等都有了其在手机端的实现版本，这样极大丰富了手机射击类游戏的可玩性。

▲图 6-2　“永远的坦克大战”的 2.5D 视角

- 游戏背景

为射击游戏提供的最简单的游戏背景就是滚动的卷轴式背景，刚才提到的雷电等许多飞行射击类游戏都是采用的这种背景显示模式。卷轴式背景实现方法就是使用一幅比游戏屏幕长的图片首尾相接作为游戏背景，在游戏的过程中通过不断循环滚动显示来达到背景变换的效果。

对于飞行类射击游戏之外的射击游戏，其游戏背景由多个小图片拼接而成，如代表房子、树木的小图片，这些小图片被称为图元，采用图元技术可以轻易地搭建出 2D、斜 45°2.5D（如图 6-2 所示的游戏屏幕）、90°2.5D 的游戏场景。

6.1.3 游戏内容设计

- 剧情设计

虽然射击类游戏要求玩家快速反应、游戏的快节奏发展、眼花缭乱的爆炸效果以及随之而来的音效，但是如果射击类游戏千篇一律地都是开枪“打兔子”，那么肯定不会吸引大批的玩家，所以为射击游戏设计合理的剧情也显得非常重要。

设计剧情的方式有很多，比如增加一段背景故事，塑造一个游戏主人公，在游戏中适当地出现人物的对话等，也可以把相互独立的关卡用背景故事串联起来。

- 游戏规则

由于射击类游戏的玩法比较单一，所以游戏规则也相对简单，除了限制玩家的生命数目之外，还要在游戏运行的过程中不断出现一些奖励，如增加生命数目、增加玩家控制角色的伤害输出等。同时对于这种操作比较简单的游戏，设置积分机制也是激励玩家不错的方法。

6.2 竞速类游戏

竞速类游戏不同于其他类型的游戏，竞速类游戏的内容比较单一，就是驾驶一种交通工具进行比赛。竞速游戏主要吸引玩家的地方在于令玩家体会到高速移动时所带来的视觉和听觉上的享受，以及冲破重重障碍到达终点的成就感。

对于目前手机平台下的竞速游戏来说，大部分使用的比赛交通工具为赛车，很少有竞速游戏会采用宇宙飞船或是舰艇等作为比赛工具。

6.2.1　游戏玩法

- 玩家个数

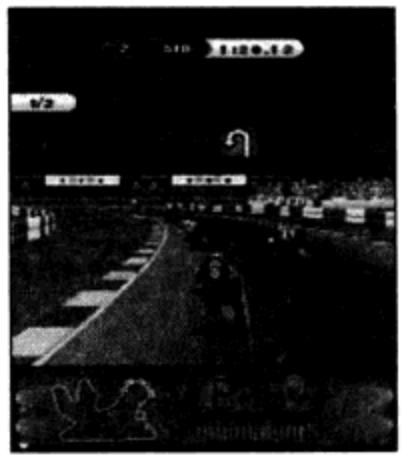
手机平台下的竞速游戏不能像电脑游戏那样方便地进行局域网互联，所以主要以单机版的竞速游戏为主，这也使得手机平台下的竞速游戏不像其他游戏类型那样火爆。

- 游戏模式

竞速游戏的趣味性不仅仅存在于驾驶，不同游戏模式下的竞速游戏也会给玩家带来不一样的体验，下面列举几个常见的游戏模式。

（1）夹杂打斗的竞速游戏，有些竞速游戏在进行中会允许玩家和其他的选手进行简单的战斗，这样可以使游戏更加紧张刺激。

（2）以职业联赛为主的竞速游戏，玩家每成功赢得一场比赛，其等级就会提升，高等级可以参加高级的比赛以获取更多的金钱和等级。如在手机游戏“世界摩托大奖赛”（如图 6-3 所示）中就采用大赛的模式来进行游戏。



▲图 6-3　“世界摩托大奖赛”游戏画面

（3）以任务为驱动的竞速游戏，这种模式下的竞速游戏由任务系统来控制游戏流程，很多时候因为任务之间的前后关系，玩家并不能随心所欲地选择比赛。

- 取胜条件

要想在竞速游戏中取胜，方法只有一个：获得比赛的胜利，不过也有一些竞速游戏并不要求玩家的名次，只要完成了比赛就算获胜。

6.2.2　视觉效果

- 游戏视角

竞速游戏的视角最初为从正上方俯视赛场，这种方式很适合躲避障碍物。不过目前大部分的竞速游都将数据显示成为了第一人称视角，这样更容易给玩家身临其境的感觉。

- 游戏画面

竞速游戏的画面一般分为两个部分，第一部分就是游戏的主画面，游戏的主画面渲染了赛场的

情景，如驾驶赛车时屏幕不断后撤的效果；第二部分就是用来模拟特定交通工具的操纵界面的控制面板。除了这两个部分，还有些竞速游戏会把任务列表或缩略地图也显示到屏幕上。

6.2.3 游戏内容设计

- 交通工具的设计

交通工具的选择决定了整个游戏的发展方向，目前市面上以赛车为题材的竞速游戏较多，但是游戏的设计者不应该只看到赛车这一种交通工具，其他交通工具可能会更加具有可玩性。

（1）水上交通工具，水上的交通工具有很多，如摩托艇、机动艇、帆船等，而且船类交通工具的驾驶对于玩家来说可能也会更加新鲜。对于赛道的设计也可以更具特色，为其添加一些有趣的障碍，如漩涡、台风甚至是水怪等。

（2）空中交通工具，主要是飞机和飞船两种，驾驶交通工具在赛道上不会有太多文章可做（除非驾驶飞船遇到小行星群），倒是这两种交通工具的驾驶方式可以好好设计一下，因为驾驶飞机尤其是飞船的方式比较复杂，可以将操作方式简化一下展示给玩家。

- 剧情设计

虽然竞速类游戏大部分时间都让玩家处于高度紧张的驾驶状态，但是，在游戏中适当地穿插剧情也会为整个游戏加分不少，尤其是对于那些任务驱动型的竞速游戏。

6.3 益智类游戏

益智游戏（Puzzle Game）是另外一种深受用户欢迎的游戏类型，很多人把益智游戏称作休闲游戏，但实际上很多益智游戏玩起来并不会很“休闲”，如一些需要频繁思考的诸如数独之类的游戏。而休闲游戏中很大一部分游戏并不属于“益智”的范畴，如后面会提到的养成类游戏一般也划为休闲游戏。

益智类游戏的特色就是，游戏中会更多地依靠智力去解决问题，而现实生活中能够锻炼智力的游戏有很多，如纸牌类游戏、棋类游戏等都属于益智类游戏。

6.3.1 游戏玩法

不同的益智类游戏由于设计的内容差很多，所以各自有各自的玩法，下面主要来谈谈益智类游戏中相同的地方。

- 玩家个数

一般来讲，益智解谜类的游戏大都为单人游戏，如经典游戏“吃豆子”（如图 6-4 所示）、推箱子、走出迷宫和拼图等，这类单人版游戏也有一个共同点，那就是都以关卡作为提升难度的手段。关卡可以是有限的，也可以是由程序自动生成的，如生成迷宫游戏中的地图就有很多成熟的算法。

还有一些益智类游戏要求多人对战的模式，如各种棋牌类游戏，经典游戏“大富翁”（如图 6-5 所示）就是多人联网进行，这里的“人”不一定非得是玩家，也可以是电脑。

- 取胜条件

益智游戏的取胜条件一般很简单，逃出迷宫、在棋（牌）局中赢得胜利等，很多的益智游戏都

会有限时功能，或者把消耗的时间作为计算积分时考量的因素。不过也有一些益智类游戏取胜条件虽然简单，但是很难实现，如大富翁系列的游戏，很难在短时间内取得游戏的胜利。

▲图 6-4　经典游戏“吃豆子”

6.3.2　视觉效果

- 游戏视角

很少有益智游戏的视角为第一人称，一般的益智游戏的视角可以看到整个游戏场景，通常都是平面游戏，近来很多平面游戏都被改造成为了具有 3D 效果的益智游戏，如 3D 版的手机推箱子，游戏视角的改变可以提高玩家的体验，相比于其他吸引玩家的手段，改变视角更容易实现。

- 游戏画面设计

除了刚才提到的手机大富翁系列，普通的益智游戏的场景不会很大，一般为手机屏幕大小，如各种棋牌游戏的画面基本不需要滚屏，多数的关卡型益智游戏的关卡场景也不会超过手机屏幕的大小，如经典游戏“吃豆子”（Pac-Man）。当然也有例外，如曾风靡一时的“掘金者“（Gold Runner）游戏的每一关的场景都需要进行滚屏，这也和游戏节奏的快慢有关。

▲图 6-5　经典游戏“大富翁”

益智游戏场景比较固定（尤其是棋牌游戏），为了提高玩家的体验，在画面的变换上就应该多下功夫，应该根据手机平台的性能，尽量让画面为整个游戏加分，如欢迎动画、游戏中出现的各种提示和动作都可以尽量漂亮些。

6.3.3　游戏内容设计

- 剧情设计

前面也已经提到，益智游戏最大的魅力在于对智力的挑战，所以玩益智游戏的玩家往往对游戏的剧情并不是非常的感兴趣，就连经典游戏“大富翁”也没有在剧情上花太多功夫。

- 难度调节

益智类游戏的难度等级必须是可调节的，这样才可以吸引玩家勇攀高峰。对于关卡类的益智游戏，一般通过关卡提升难度就足够了。而对于不设关卡的益智游戏来说，通常在游戏开始的时候提示玩家选择一个难度等级，如棋牌类游戏等。

- 游戏规则

益智游戏的游戏规则设计需要把握好 3 个方面：一是入门难易程度，游戏开始时不能让玩家感觉到太简单，也不能让玩家太沮丧；二是趣味性，益智游戏吸引人的地方不仅在于对智力的挑战，还必须在趣味性上大作文章；三是耐玩度，怎样让玩家愿意再玩一次，也是需要在设计的时候多做考虑的。

6.4　角色扮演游戏

角色扮演游戏（Role Playing Game）是手机游戏中的另外一个大阵营，不同于益智游戏等即时

性的游戏，角色扮演游戏一般要求玩家投入较多的注意力和较长的关注时间，同时一款优秀的角色扮演游戏的开发投入也相对较高。

6.4.1 游戏玩法

- 玩家人数

电脑平台下的角色扮演游戏即有单机版的，也有局域网对战版和网络多人在线形式的，手机平台下角色扮演游戏也不局限于单人模式，很多的大型网络在线手机游戏都是角色扮演性质的，但是论数量，还是单击模式的 RPG 游戏占多数。

- 游戏主线

单机版的角色扮演游戏的主线比较明朗，单机模式的 RPG 往往会把玩家控制的角色定义成为“救世主”的形象，所以整个游戏都会围绕这个角色展开，由玩家控制的角色来串接故事情节并影响游戏的发展方向，任务系统在角色扮演游戏中比较常见，好的任务系统可以对游戏剧情起到推动的作用。

网络版的角色扮演游戏一般对单个玩家没有这么高的定位，所以对于玩家来说，主要游戏主线在于控制自己的玩家进行各种探险、战斗并以此来提升自己属性，网络游戏中也可以通过复杂的任务系统来让玩家体会到整个游戏剧情发展。

- 取胜条件

网络版的角色扮演游戏一般没有取胜条件，而单机版的角色扮演游戏的取胜条件由剧情来决定，通常是以解除危机、打败最终魔王为游戏胜利的条件，有些角色扮演游戏还会有不同的结局。

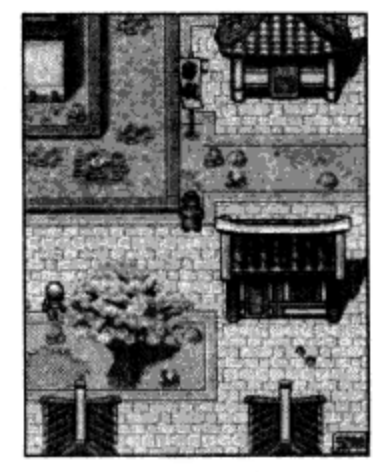
6.4.2 视觉效果

- 游戏视角

角色扮演游戏基本上不会以 2D 的视角来呈现，通常游戏视角都是 2.5D 或者 3D，而对于 2.5D 又有斜 45° 俯视和正 90° 俯视两种。如一款移植自电脑游戏的“仙剑奇侠传”采用的就是斜 45° 俯视视角，如图 6-6 所示，而手机游戏“游戏仙侣情缘之麒麟劫”的视角则是正 90° 俯视，如图 6-7 所示。

▲图 6-6 “仙剑奇侠传”的游戏视角

不管是斜 45° 和正 90°，都是采用图元技术加上多个图层叠加实现的，所以这类角色扮演游戏中地图设计将会是一个非常重要的环节。目前的角色扮演游戏以正 90° 俯视视角居多，其实现方式也比斜 45° 俯视要简单一些。



▲图 6-7 “游戏仙侣情缘之麒麟劫”的游戏视角

由于 2.5D 视角更容易让玩家习惯，很多 3D 视角的角色扮演游戏也会在其中穿插 2.5D 的场景。

- 游戏界面

角色扮演游戏的界面不应该只有游戏场景和菜单这么简单，出于剧情和玩家需要，必须为游戏创建不同用途的界面，如对于常见的武侠题材的 RPG 游戏，就需要为玩家创建角色属性面板、物品及装备面板、技能面板等界面。

6.4.3　游戏内容设计

- 情节设计

对于角色扮演游戏，故事情节的好坏在很大程度上影响了游戏带给玩家的体验，所以在游戏设计初期必须选好一个题材。

通常角色扮演游戏的题材背景会选择在一个不同于普通人生活的世界，比较多的是来自武侠文学如金庸等大师的作品，或者是西方的玄幻文学如指环王或吸血鬼题材，还有一批游戏是来自于电影或其他科幻小说。

确定了题材，还需要丰富整个游戏的剧情，一般来说，角色扮演的游戏方式主要包括探险、接收任务以及战斗，合理的分配这 3 种游戏方式，可使游戏的可玩性达到最高。

- 角色设计

角色扮演游戏中，尤其是单机模式下的角色扮演游戏，角色设计的重要性是不容忽视的，除了玩家控制的“救世主”角色，还要设计其他的辅助角色，如用来指引“救世主”走向强大的导师，一起进行探险的伙伴，要消灭的最终 Boss 等。

对于主要的角色，还需要设计其详细的属性，如为玩家控制的“救世主”以及并肩作战的伙伴设计战斗时用的属性。以武侠题材的 RPG 为例，需要为主要角色设计的属性有技能、血量、法力、等级等，对于大型的 RPG，还需要设计职业及装备等。

- 主角成长

玩家控制的角色在游戏中不断成长是游戏的趣味性之一，同时也是游戏情节发展的主线。所以在设计游戏时需要根据故事情节让主角不断成长，这种成长包括个人属性的提升以及游戏剧情的逐步铺开，主角的成长方向同时也是吸引玩家坚持玩到底的原因之一。

- 游戏存储

对于一般玩家来讲，角色扮演游戏很少能够在短时间内通关，所以必须为游戏增加存储功能。游戏中可以采用到指定地方才可以存储的模式，也可以用菜单选项让玩家随时存储。

6.5　闯关动作类游戏

这里要介绍的闯关动作类游戏，是区别于射击类游戏和格斗游戏的，闯关动作类游戏的节奏一般比较轻快，玩家的成就感主要来源于完成一个个关卡的挑战胜利，更具趣味性。闯关类动作游戏的设计重点不在战斗，而是在闯关，这样适应的玩家人群会更广，比较经典的闯关类动作游戏有“超级玛丽”（如图 6-8 所示）和冒险岛（如图 6-9 所示）等。

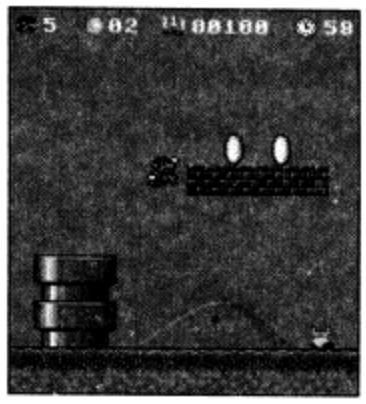
6.5.1　游戏玩法

- 玩家人数

玩家玩闯关动作类游戏的主要目标一般都在于过关斩将，并不十分需要别的玩家的阻挠或协助，所以通常都为单机游戏。

- 操作方式

闯关动作类游戏一般为老少皆宜的游戏，所以其操作方式不会太复杂，只是简单地控制游戏角色移动和释放技能即可。对于有些闯关类动作游戏，玩家大部分时间都在命令游戏角色不停地跳跃，如“超级玛丽”等游戏。



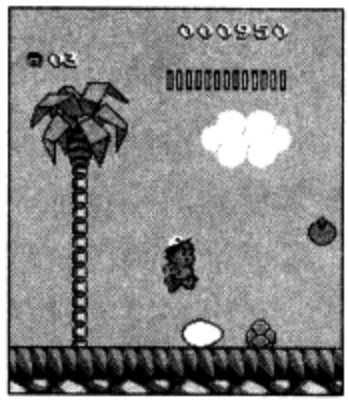
▲图 6-8　手机游戏“超级玛丽”游戏画面

- 取胜条件

闯关动作类游戏的取胜条件就是将一个个的关卡挑战成功，一般最后的关卡中会出现游戏的最终 Boss，打败这个 Boss，就宣告玩家成功通关。

6.5.2　视觉效果

对于闯关类动作游戏来说，其背景最为经典的游戏画面就是横向滚动式，这种游戏的视角可以让玩家做到“旁观者清”，能够较早地看到前面可能要遇到的障碍和挑战。当游戏的节奏比较快时，玩家可以有时间做好准备。



▲图 6-9　手机游戏“冒险岛”游戏画面

6.5.3　游戏内容设计

- 剧情设计

闯关动作类游戏一般并不会大费周章地以华丽的背景故事去吸引玩家，但是也会有一个简单的背景故事，在背景故事中会向玩家简单介绍闯关的动机和玩家所要追求的终极目标。之后玩家的注意力应该被层出不穷个关卡所牵引，而不是对剧情念念不忘。

- 关卡设计

前面也曾提到，闯关动作类游戏的操作方式不会特别复杂，游戏剧情也不会特别引人入胜，那么要想让玩家在简单的游戏中获得乐趣，关卡的设计就显得尤为重要。

提到关卡设计，肯定要考虑其难易度的把握，增加难度的方式有多种，除了重新设计高难度的关卡之外，对游戏进行微小的改动也可以实现难易程度的改变，如给玩家更短的时间来通过挑战、将游戏中的奖励物品放置在更危险的地方等。

除了在关卡的难易程度上做文章外，还可以根据关卡的不同制定不同的游戏规则，这样不容易使玩家产生游戏疲劳感，如第一关中，游戏角色在森林中进行跑跳，而第二关就可以把场景设到水下，玩家可以在水中自由移动，但是要记得到水面上换气。

6.6　冒险游戏

冒险游戏（Adventure Game）是另外一种需要故事情节的游戏，冒险游戏与角色扮演游戏类似，不同的是冒险游戏是在故事中添加了游戏元素，而角色扮演或其他游戏是在游戏中穿插故事。冒险游戏一般也不需要什么策略或技巧，玩冒险游戏就像是读历险记一样。

6.6.1　游戏玩法

- 冒险模式

传统的冒险游戏坚持的原则主要是“说故事”，其曾是非常热门的游戏类型。在传统的冒险游

戏中，玩家主要的工作是探索发现以及根据现有的状态进行选择。

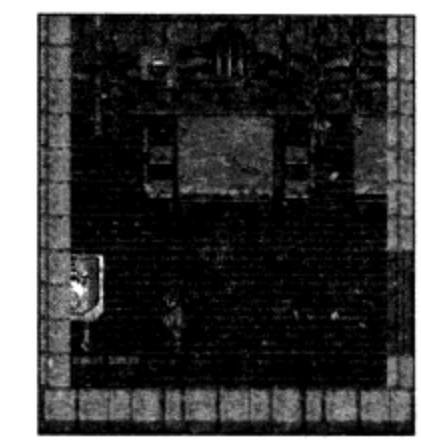
随着游戏类型的不断衍变，很多其他类型的游戏中都会或多或少地添加一些冒险的成分，而冒险游戏中也出现了一些其他游戏类型的元素，如动作类冒险游戏就是一种，动作类冒险游戏在冒险过程中增加了快节奏的打斗等场景，使游戏更加刺激和紧张。

▲图 6-10　“寂静岭”游戏画面

- 面临的挑战

不像动作游戏，玩家在冒险游戏中控制的角色不会具有太强的侵略性，但仍然会遇到一些冲突或阻碍。这些困难不会让玩家手忙脚乱、应接不暇，但也会使玩家将思绪投入其中，乐此不疲。

不管是解谜类的冒险游戏，如“寂静岭”（如图 6-10 所示），还是逃脱类的冒险游戏，如“越狱 24 小时”（如图 6-11 所示），游戏中遇到的困难基本上类似，如打开一扇需要钥匙的门、需要完成 NPC 的任务、需要找齐全套的物品用于组合等。在冒险游戏中，玩家通常不是靠快速反应或长时间思考来解决问题的，而是依靠已获得的游戏线索或物品道具甚至技能来排除阻碍的。


▲图 6-11　“越狱 24 小时”游戏画面

提示　冒险类游戏有一个其他类型游戏所没有的特点，那就是很难让玩过一次的玩家再去经历一次冒险，就像是很少有人会去把看过的电影再看一遍一样。这也体现了故事情节对于冒险游戏的重要性，一旦故事情节已经了然于胸，冒险游戏就再也无法给人以刺激和惊奇了。

- 取胜条件

既然冒险游戏就像是在讲故事，故事讲完也就尘埃落定了。除非是为续集留悬念，一般的冒险游戏在最后会出现类似大团圆的结局，如逃出困境、解救了爱人或朋友、找到了丢失的宝藏等。

6.6.2　视觉效果

- 游戏视角

冒险游戏的视角很多情况下与角色扮演游戏的视角类似，前面曾提到的“越狱 24 小时”就是正 90° 俯视的 2.5D 的视角，这样玩家可以对整个游戏场景有充分的掌握。不过也有很多手机上的冒险游戏以第一人称视角来呈现，为的是增强玩家身临其境的感觉。

- 画面和地图

一个冒险游戏就是一个历险故事，而讲述这个故事时除了文字，还有画面，所以，游戏的画面不应该拖故事情节的后腿，应该尽量提高玩家的空间感。另外，不管是 2.5D 的俯视视角还是第一人称视角，在冒险游戏中增加一个缩略地图都不失为一种不错的选择，这样玩家不至于迷路，也不会重复探索已经走过的地方。

6.6.3 游戏内容设计

- 故事发展

冒险游戏如果没有跌宕起伏的故事内容，就基本失去了可玩性，而如果只是像念书那样讲故事，也会大大降低可玩性。游戏中展示的是场景而不是故事，所以在将故事改编成游戏或为游戏增加故事情节时，要注意规划游戏的故事结构，把各种剧情插入到不同的场景之中，场景和场景之间又相互关联，最后形成相互串连的游戏场景。

- 关卡设计

如果冒险故事情节比较漫长，通常要将故事切成若干个关卡，或者称为章节。这些关卡的连接方式有多种，如单线索方式是将所有关卡连成一条线循序渐进地向玩家铺开；多线索方式则是在进入新关卡前让玩家选择，或者根据玩家现在的游戏状态进入相应的关卡。

- 对话设计

在游戏中主要的对话对象是 NPC，一般来说游戏中加入对话主要是为了增加趣味性，使玩家更能体会到游戏的互动性。而在冒险游戏中，对话还有另外的用途，那就是为玩家提供重要的游戏线索，所以在设计冒险游戏时应该对对话进行合理的设计。

6.7 策略游戏

手机平台下的策略游戏来源于电脑端的策略游戏，其最初是模拟类游戏的一个分支。随着策略游戏的不断发展，其也衍生出了很多其他不同的形式，如回合制策略游戏和即时策略游戏。通常，即时战略游（Real-Time Strategy）也被认为是从策略游戏发展而来。

6.7.1 游戏玩法

- 玩家个数

在其他游戏中，玩家往往通过在游戏中控制一个角色来参与游戏，而在策略游戏中，玩家常常没有具体的角色，或者说玩家控制不止一个角色。在策略游戏中玩家扮演的角色是统筹各个方面的总司令。这在一定程度上也增加了游戏的复杂度。

电脑平台下的策略游戏一般不会限制玩家的个数，玩家既可以同电脑对战，也可以和其他玩家一起游戏，或者合作，或者对抗。而在手机平台下，大部分策略游戏都是单人模式的，玩家主要和电脑中的敌人或朋友一起游戏。

- 操作方式

策略游戏的玩法非常简单，那就是探索、发展、征服。所以不管是回合制策略游戏、即时策略游戏还是即时战略游戏，玩家的目标和实现目标的方式都比较单一。简单地讲就是，在初期探索未知的世界，然后建立自己的基地并不断发展扩大，当然在发展的过程中也伴随着进攻和防守。

- 取胜条件

不同于其他类型的游戏，策略游戏更偏重于思考和谋划，因此，策略游戏所消耗的时间有可能会很长，要显示的信息量也很大。同时有些即时战略游戏包含任务系统，这些连续的任务也不是短

时间内能够完成的。

按照策略游戏的原则，策略游戏的取胜条件在于征服，即完全消灭掉游戏中的敌人或者不幸地被敌人消灭掉则宣告游戏结束。

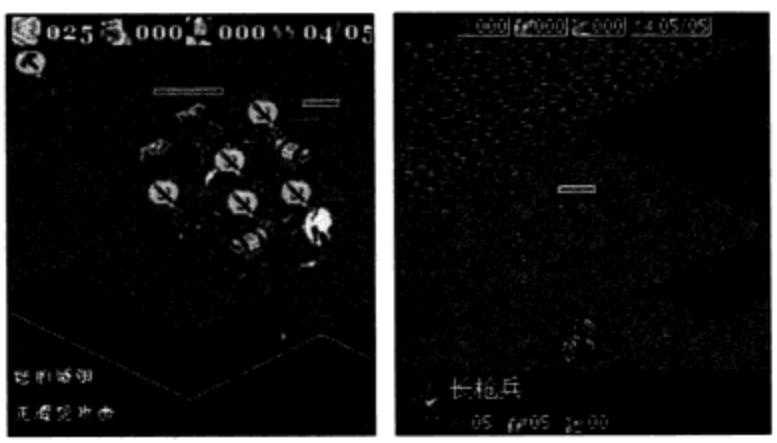
6.7.2　视觉效果

- 游戏视角

电脑平台下的策略游戏一般为 2.5D 视角，或者是可以在 2.5D 视角和第一人称视角之间切换。手机平台下的策略游戏由于不容易变换视角，所以通常都采用 2.5D 俯视视角，这样不仅玩家的视野会比较开阔，操作游戏中不同对象也比较容易，如“帝国时代”手机版（如图 6-12 所示）采用的就是斜 45° 俯视视角。

- 缩略地图

策略游戏中的地图都非常大，这样能够保证开展游戏的多方阵营能够在初期平稳地生存。在这种情况下缩略地图对于玩家来说就是一个非常有用的工具了。利用缩略地图，玩家可以迅速把镜头从一个地方切换到另一个地方，同时也可以迅速了解游戏局势。

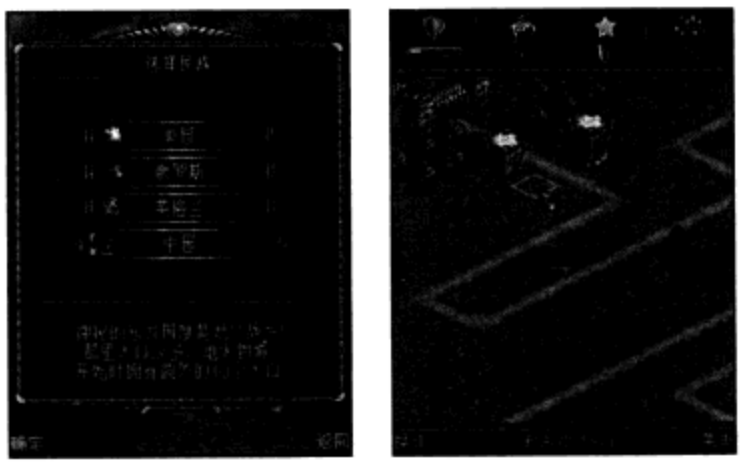


▲图 6-12　手机游戏“帝国时代”的视角

6.7.3　游戏内容设计

- 题材设计

策略游戏的题材形式并不多，一般都与战争有关，只是选择的背景不同，除了选用历史题材外，通常魔幻和玄幻题材的故事也比较多。往往一个成功的策略游戏，其引人入胜的背景故事功不可没，一些策略游戏的背景故事甚至还会改编成电影或其他文学作品。



▲图 6-13　“文明”游戏画面

- 游戏平衡设计

策略游戏中往往会出现许多的阵营，如移植自电脑同名游戏的“文明”（如图 6-13 所示），不同的阵营之间所具有的游戏角色和发展路线也不一样，否则游戏的可玩性将会大大降低。而如何让不同阵营在游戏的进行中保持平衡就需要好好考虑的问题了，一款失衡的策略游戏将也会失去大批的玩家。

6.8　养成类游戏

养成类游戏是目前手机游戏中的新贵，养成类游戏来自于曾经风靡的电子宠物，但后来随着手机软、硬件性能的不断提升，加上手机这种随身携带的特性，养成类游戏慢慢成为了手机平台下不

可忽视的一类游戏。

6.8.1 游戏玩法

- 玩家个数

养成类游戏一般强调主人（即玩家自己）同被养者之间的关系亲密程度，所以一般这类游戏都是单人模式。不过有些养成类游戏为了给玩家一个展示成果的机会，增加了联网的模块来让不同的玩家带着自己的宠物进行各个方面的 PK。

- 游戏过程

养成类游戏一般不要求玩家费过多的脑筋，而且其游戏节奏也很慢，因为一个优秀的宠物不是一朝一夕就培养出来的。玩家玩这类游戏主要的乐趣来源于宠物在自己的照料下一点一点地成长，接受训练并且能够和玩家进行一些简单的情感交流等。

- 取胜条件

除了网络版的养成类游戏时不时地会把宠物搬出来进行 PK，一般情况下养成类游戏没有获胜的概念，这点是和其他类型游戏的最大不同。不过还是会有失败的情况，比如在培养的过程中一时疏忽使宠物死掉了。

6.8.2 视觉效果

- 宠物造型

养成游戏中玩家基本上不会出现在游戏中，所以主要显示的内容就是宠物，对于宠物的造型就需要多花些功夫雕琢了。往往在养成游戏中的宠物都不是现实生活中能直接找得到宠物，宠物或者是一些经过改造过的地球生物，或者干脆是科幻的产物，有些养成类游戏的被养对象还可能是人。

- 游戏画面

通常情况下养成类游戏的显示元素比较单一，主要为被养的对象，所以为了提高玩家的视觉体验，应该把游戏的画面做好规划。同时在游戏中还需要设计管理面板等界面，如宠物属性面板的布局等。图 6-14 显示的是养成类手机游戏“口袋妖怪”的宠物显示面板。

▲图 6-14 “口袋妖怪”的宠物面板

6.8.3 游戏内容设计

- 宠物设计

宠物的设计是养成类游戏的设计重点，因为其相当于是整个游戏的招牌。设计宠物时除了要设计其外在形象，还需要设计其各种属性，如宠物所具有的技能和宠物的成长路线等。

- 培养方式设计

养成类游戏的培养方式设计也很重要，否则玩家就会发现其宠物过于迟钝或是聪明，无法从中体会到成就感。一般来说游戏通过以下几种方式来留住玩家。

（1）把事件变复杂，这里并不是把事件变得困难，只是将其分解得更细致一些。如想要提高宠物的快乐值，游戏并不会提供一个菜单选项叫做“提高快乐值”那样简单，相反，玩家可能需要先喂饱宠物，给宠物洗个澡再和宠物聊天玩耍……这么做不仅会让玩家有事可做，同时也会丰富游戏的内容。

（2）用时间控制游戏过程，如要让宠物学会一项技能，宠物不可能马上就学会，而是需要经过一定的时间后才能学会，这样用时间也可以留住玩家。

（3）多让玩家介入游戏，有了时间来控制游戏过程，游戏的过程中也不应该把玩家晾在一边太长时间，应该尽量让玩家参与更多的决策，可以设计宠物不同的成长路线来让玩家来选择，这样让玩家深度介入，会更容易留住玩家。

- AI 设计

游戏中的人工智能（AI）也是不得不考虑的设计内容，宠物如何在玩家不干涉的情况下自动衰减自身的诸如饥饿、心情等属性，如何响应主人（玩家）的诸如问候之类的交互等，这些都需要开发相应的人工智能算法。

- 其他游戏元素的设计

看了上面的介绍，读者朋友可能会觉得对于玩家来讲，养成游戏或许显得太简单了。的确是这样，但是目前，市面上的一些养成游戏会在游戏中加入其他的游戏元素，如带领宠物和怪物进行战斗、与宠物一起玩益智游戏等。这样一来养成类游戏的可玩性就会大大提高。

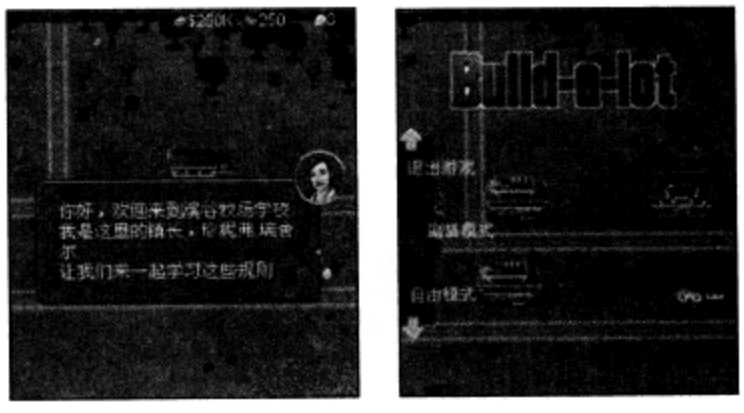
6.9 经营类游戏

经营类游戏是另外一种需要玩家长时间关注的游戏，经营类游戏一般模拟现实世界中的某种行业如餐馆或公司等，如手机游戏“地产大亨”（如图 6-15 所示）就是让玩家通过对房屋土地的买卖或出租成为富翁。玩家的游戏目标在于超越游戏中的其他对手，获得自己的发展，并从经营和管理自己现实中很少接触的事物中获得乐趣。

6.9.1 游戏玩法

- 玩家个数

手机平台下的经营类游戏一般都为单机模式，在有些经营类游戏中玩家主要关心的是如何管理好自己，而在一些游戏中往往会为玩家提供一些竞争对手来增加挑战。



▲图 6-15　“地产大亨”游戏画面

- 获胜条件

对于一些不太具有侵略性的经营类游戏（如经营餐馆），往往没有一个确定的取胜条件，玩家的乐趣在于经营中的收获。有些经营类游戏则需要向策略类游戏那样不断发展扩张自己的势力，最后打败其他的竞争对手赢得胜利。

6.9.2 视觉效果

- 游戏视角

经营类游戏中，玩家要关注的地方很多，所以在手机平台下一般都采用 45°（或 90°）的 2D 俯视视角来呈现游戏的画面。

- 游戏界面

不管是侵略性的经营游戏还是简单管理类的经营游戏，都需要为玩家提供一个可以浏览其所经营项目的运作情况的界面，一般来说这些界面就是一些游戏属性的展示。同时还要为玩家提供一个管理面板来对其所掌握的资源进行分配等操作。

6.9.3 游戏内容设计

- 游戏流程的发展

不同于策略游戏，在经营类游戏中玩家对于整个游戏进程的影响不是特别大（除非是到了玩家已经很强大的时候），所以游戏中大部分时间需要在程序中控制游戏的发展流程，如调动 AI 去和玩家竞争或随机产生突发事件。

- 资源的平衡设计

经营类游戏从本质上讲，就是玩家不断收集资源发展自己的过程，所以每个经营类游戏内部都会有一个资源系统，这种资源可以是最普通的金钱，也可以是人（如“实况足球经理”），还可以是土地（如“地产大亨”）等其他东西。

有些资源会在游戏中被玩家消耗掉，然后产生新的，有点则是在不同人的手中来回交换或交易，还有的则是自动产生自动消耗。所以在设计游戏的时候需要对涉及到的资源进行详细分类，并在游戏进行中进行合理的管理。

6.10 体育类游戏

体育类游戏是面向体育爱好者的一类游戏，虽然拥有的玩家群体不如角色扮演或益智类游戏多，但是体育类游戏还是在众多的手机游戏种类中因独特的内容题材占有一席之地。

6.10.1 游戏玩法

- 玩家人数

由于手机平台下的局限性，一般的体育类游戏都为单机模式，即玩家进行体育竞技的对象是电脑 AI，这时游戏的可玩性很大程度上取决于 AI 的真实程度。

- 取胜方式

体育类游戏主要是模仿现实中体育竞技运动，所以取胜方式就是赢得比赛的胜利，或根据剧情需要赢得一系列比赛的胜利，如“NBA 职业篮球”（如图 6-16 所示）。在这类游戏中玩家主要操控的对象是一个或多个运动员。

有一种体育类游戏融合了经营管理类游戏的元素，使得游戏的乐趣不在于取得竞技上的胜利，而是把一个俱乐部经营管理好，玩家扮演的角色，也不再是运动员，而是教练或经理之类的管理职务，如“实况足球经理”（如图 6-17 所示）这样的游戏。

▲图 6-16　“NBA 职业篮球”游戏画面

▲图 6-17　“实况足球经理”游戏画面

6.10.2　视觉效果

- 游戏视角

体育类游戏的视角应该取决于竞技项目，如果是一对一的比赛，如网球或摔跤等，那么既可以取第一人称视角，也可以取其他视角，因为只要能够看到对手就行。但是对于团队竞技项目（如篮球、足球等），一般不会选择第一人称视角，因为玩家需要实时掌握场上的局面。

而对于一些竞速性质或带有跑道的竞技项目（如滑雪、游泳等），一般采用背后视角来设计游戏，有一些也会采用类似闯关类动作游戏的横向滚屏式。

- 动画设计

对于需要模拟真实竞技场景的体育类游戏，对于人物造型和人物动作的设计也很重要，否则玩家会因为生硬的人物线条和扭曲的人物动作提前放弃游戏。

6.10.3　游戏内容设计

- 操作游戏的设计

在体育类游戏中，由于竞技方式的多样化，往往需要向所控制角色下达很多命令。而手机的操作接口是有限的，只有方向键或其他数字按键等，所以在设计游戏时需要对游戏的操作接口进行详细的研究，确保玩家能够感觉到完整的运动体验。

- 人工智能设计

对于单机模式下的体育类游戏，除了模仿真实运动场景的动画外，人工智能的设计也算是比较重要的环节。对于体育类游戏的 AI，可以采用记录状态的方式来设计 AI，比如在多人球类竞技比赛中，当电脑控制的角色处于不同的状态时（如进攻态和防守态），所采取的对策也不一样。

6.11　小结

本章对目前流行的手机游戏类型逐一进行了介绍。希望读者对这些游戏的主要玩法和不同于其他游戏类型的独到之处有所了解，以便自己在开发相应类型的手机游戏时能够做到心中有数。

虽然本章将各种游戏类型分开介绍，其实随着手机产业的不断发展，一些手机游戏类型之间的分类也越来越模糊，很多手机游戏中融合了不同游戏类型的风格。还有一些新的游戏类型在悄然出现，读者在实际开发中也需要注意这些问题。

第 7 章　游戏背后的数学与物理

本章将会对游戏中的数学和物理知识做一个简单的介绍。在游戏开发中除了要对开发语言、设计模式等知识熟悉外，还必须对数学和物理方面的知识有所了解，因为决定一款游戏视觉效果和玩家体验的，恰恰是这些看似和编程毫不相关的理论知识。

7.1 编程中经常用到的数理知识

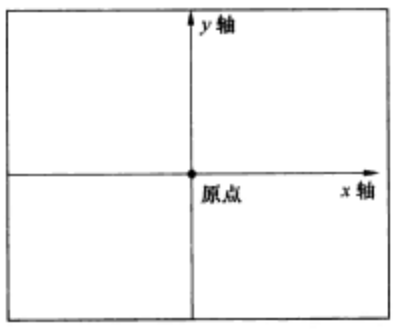
数学和物理知识对于程序开发人员尤其是游戏开发人员来说是不可或缺的，开发人员不需要做到精通，只需要掌握游戏开发中经常会遇到的数学和物理方面的知识即可，本节就来简单介绍一下在编程中经常用到的一些数学和物理方面的知识。

7.1.1　数学方面

造就一款优秀的游戏，仅仅依靠新奇的创意和缜密的业务逻辑还是不够的，数学知识在游戏编程中也起到了不容忽视的作用，本节就来简单列举一些游戏开发中经常用到的数学知识。

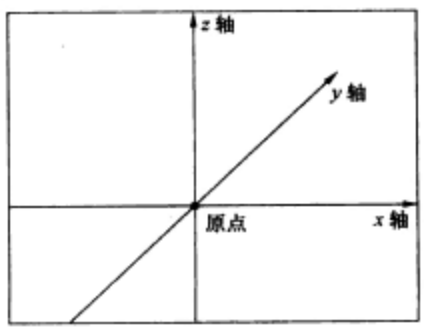
1. 坐标系

如果读者曾经写过简单的小游戏，对于坐标系这个概念肯定不会感到陌生。不论采用什么开发语言，通过写程序在屏幕上绘制图形都必须使用坐标，这些坐标代表所绘制元素在屏幕上的位置，其将会对应到屏幕的坐标系。



▲图 7-1　笛卡尔二维坐标系

这些坐标系有一个共同的名字：笛卡尔坐标系。在游戏编程中，常用到的笛卡尔坐标系统就是二维坐标系（如图 7-1 所示）和三维坐标系（如图 7-2 所示）。二维坐标系是两条交于一点的垂直数轴，而三维坐标系则在二维坐标系中增加了一条垂直于 x、y 轴所确定的平面且过原点的数轴 z。

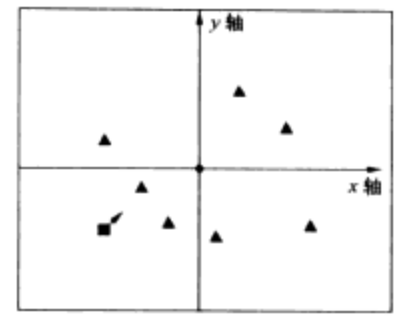


▲图 7-2　笛卡尔三维坐标系

笛卡尔二维坐标系确定了一个平面，如 2D 游戏中的游戏屏幕就是一个平面。当需要表达一个三维的立体空间时就需要采用笛卡尔三维坐标系了。二维坐标系中需要两个坐标值(x,y)来惟一确定平面上的一点，三维坐标系则需要 3 个

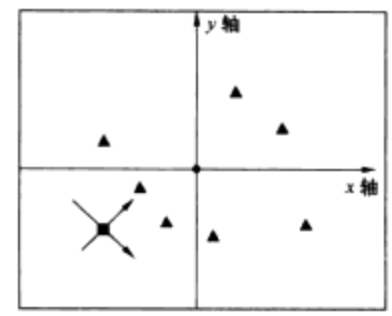
坐标值(x,y,z)。

在一个游戏场景中，往往有一个绝对的坐标系系统，所有的元素都在其中有惟一的位置。这看似已经满足开发需要了，但是很多情况下绝对坐标系统使用起来并不方便。例如，有图 7-3 所示的一个游戏场景，方块代表玩家控制的角色，方块上的箭头指向代表人物朝向，三角形代表敌人。玩家需要检测在其前方有无敌人以及敌人的方位。



▲图 7-3　玩家需要侦测附近敌人的方位

如果在检测前方敌人的算法中使用绝对坐标系，那么计算起来将会非常繁琐，这时候如果建立一个局部的坐标系，以玩家中心为原点，分别以玩家的朝向和过原点且垂直于朝向的向量为 y 轴和 x 轴，如图 7-4 所示。使用这个局部坐标系进行计算将会简化很多工作，而且这个局部坐标系可以和绝对坐标系进行变换。



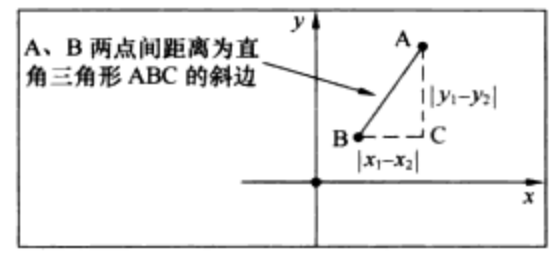
▲图 7-4　以玩家为原点建立局部坐标系

使用局部坐标系可以在很多场合下简化计算，其与绝对坐标系之间的转换也比较简单，可以利用向量的知识来完成。

2. 距离的计算

在游戏编程中肯定少不了一种计算就是两个物体之间的距离，距离计算应用于很多诸如碰撞检测和搜索路径等场合。在距离计算中，比较常用的就是两点间坐标公式，即 $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$，该公式由勾股定理推导而来，如图 7-5 所示。

但是一般情况下，为了节省开销，通常编程计算到最后一步时不进行开方操作而是直接对平方值进行比较得出结果，这样简化了计算，在运算频度高的情况下对性能有一定的优化作用。有些时候距离计算的目的只是用于比较，这时候就可以使用更省力的办法，那就是门特卡罗距离。



▲图 7-5　两点间距离公式计算图示

门特卡罗距离不需要平方，如图 7-5 中所示计算 A、B 两点间的门特卡罗距离时只需要对直角三角形 ABC 的两个直角边求和即可。这种算法虽然并不是很准确，但是在大多数情况下这种算法还是可以很好地满足游戏编程要求的。

3. 向量

向量在进行人工智能的设计时经常会用到，向量的概念想必读者朋友们都很熟悉，这里不再赘述。在游戏开发过程中，用到向量的地方主要有以下几种情况。

- 计算投影和夹角

很多时候游戏中需要的向量操作就是将向量投影到某个特定的平面上，或者对两个向量进行计算求得两个向量的之间的夹角。

- 判断方向

向量也可以用来判断方向，如在游戏中也经常将两个向量相乘，这两个向量分别代表两个游戏

角色的朝向。对这两个向量进行点乘，如果结果为正，则说明二者之间的夹角小于 90°，大致位于同一个方向，如果结果为负，说明两个游戏角色之间面朝不同的方向。

- 参与复杂计算

向量的另外一个重要作用就是参与复杂的数学计算，在有些游戏编程中，需要进行大规模的矩阵运算，而向量在矩阵运算中所扮演的角色也是很重要的。

4. 设计游戏中的各种公式

往往一个具有一定规模的游戏都会需要设计游戏公式。例如，一款以古代战争为题材的策略游戏就需要设计很多的公式，这些公式包括军队战斗力计算公式、军队防御力计算公式、战斗胜负计算公式、玩家经验等属性增长公式等。

要想设计好这些游戏公式，除了对游戏的业务逻辑了解透彻之外，数学的功底如果不够扎实，那么设计出来的公式肯定会闹出笑话。

5. 其他数学公式

数学中（尤其是几何类数学）有着数不清的公式和定理，有些重要的公式是通过集中学习掌握的，也有一些有用的公式或定理主要靠自己平时的积累。如判断一个点是否在多边形内部可以采用射线法，即从待测点向任意方向发出一条射线，如果射线与多边形的交点为奇数，则该点在多边形内部，否则点在多边形的外部。

数学天才高斯曾说过："数学是科学的女王"，从本节介绍的内容也可以看出，数学知识的确在开发过程中必不可少。本节只是将一些常见的数学知识列举出来，希望各位读者多多注意积累和实践。

7.1.2 物理方面

在介绍了游戏中经常用到的数学知识后，本小节就来讲讲物理知识在游戏中的表现形式，之所以先介绍数学知识，就是因为数学是物理的基础，物理公式都是依赖数学实现的。

物理研究的是客观世界的各种规律，而游戏中不管题材如何，对于客观世界的模拟肯定不会少，而模拟客观世界的画面，就必须用到物理方面的知识。具体说来，游戏开发中主要运用的是物理学中与运动有关的内容。

- 速度、加速度、位移

速度是表征物体运动快慢的物理量，其是有方向的，该方向代表物体的运动方向；而加速度则是表征速度变化快慢的物理量，其也是有方向的，代表物体速度是增加还是减少。速度与时间的乘积形成位移。这些知识想必读者朋友们都非常熟悉。

真正在游戏中运用的，还是那些物理公式，如计算位移的公式 $2aS = v_t^2 - v_o^2$ 和 $S = v_0 t + 1/2at^2$，计算加速度的公式 $a = F/m$ 和 $a = \Delta v/\Delta t$ 等。

- 游戏中的物理运动

游戏中经常会出现的物理运动除了简单的匀速直线运动外，还有抛物运动，例如平抛、上抛等。在现实世界会采用重力加速度 g 来计算，在编程中由于各个物理量的单位不同等原因，重力加速度

在数值上并不一定和 g 相等。

对于非直线类的运动的处理方式也很简单，采用笛卡尔坐标系，将运动分解到两个方向轴上，在计算位移时分别进行，最后求得的坐标自然是合成后的运动轨迹。

- “非物理”现象

虽然在游戏中应用物理知识会收到很好的效果，但是物理引擎过于模仿现实也并不总是好事，应该适当允许“非物理”现象的出现，“非物理”现象指的是那些游戏中明显违背常理的运动方式。

例如，让玩家控制的角色在下落的时候可以在空中改变方向水平移动，或者是允许玩家在跳跃的时候在最高点二次起跳等。这些现象显然已经违反了物理学的定理，但是加入到游戏中却会收到更好的效果，这点读者需要注意。

7.2 物理小球在游戏中的应用

前面的小节介绍了游戏编程中经常使用的数学和物理知识，本小节就来通过一个物理小球的例子来说明在编程中合理地使用物理知识所带来的好处。本例主要模拟了一组大小不一的球以一定的水平初速度从高处落下的运动轨迹，整个应用程序的开发将在随后的几个小节中详细地展示给读者。

7.2.1　开发运动体 Movable 类的代码

本例中的小球为一个可移动物体 Movable 对象，该类中除了包含小球图片对象之外，还包括了如位置坐标、水平速度、竖直速度等一系列用于模拟小球运动的成员变量和一些方法。Movable 类的开发可以分为如下两个步骤来进行。

（1）新建一个项目 Sample_7_1，在 src/wyf/wpf 目录下创建文件 Movable.java，并在其中声明成员变量。为了进行物理计算，Movable 类中声明了一些跟物理量相关的成员变量，这些成员变量的声明如下。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_1/src/wyf/wpf 目录下的 Movable .java。

```
1    package wyf.wpf;                                  //声明包语句
2    import android.graphics.Bitmap;                   //引入相关类
3    import android.graphics.Canvas;                   //引入相关类
4                                                      //代表可移动物体的 Movable 类
5    public class Movable{
6          int startX=0;                               //初始 X 坐标
7          int startY=0;                               //初始 Y 坐标
8          int x;                                      //实时 X 坐标
9          int y;                                      //实时 Y 坐标
10         float startVX=0f;                           //初始竖直方向的速度
11         float startVY=0f;                           //初始水平方向的速度
12         float v_x=0f;                               //实时水平方向速度
13         float v_y=0f;                               //实时竖直方向速度
14         int r;                                      //可移动物体半径
15         double timeX;                               //X 方向上的运动时间
16         double timeY;                               //Y 方向上的运动时间
17         Bitmap bitmap=null;                         //可移动物体图片
18         BallThread bt=null;                         //负责小球移动时
19         boolean bFall=false;                        //小球是否已经从木板上下落
20         float impactFactor = 0.25f;                 //小球撞地后速度的损失系数
```

```
21          ……//此处省略构造器代码，将在随后的步骤补全
22          ……//此处省略成员方法 drawSelf 的代码，将在随后的步骤补全
23    }
```

- 代码第 6～7 行声明的 startX 和 startY 变量记录的每一个运动阶段（如从最高点下落到最低点）开始时小球的初始 x、y 坐标，在随后的物理计算中，小球的实时 x、y 坐标将会是初始坐标加上这段时间内的位移。
- 代码第 10～11 行声明的 startVX 和 startVY 是小球每一个运动阶段初始时刻在水平方向 x 和竖直方向 y 方向的速度，二者将用于计算小球的实时速度。代码第 12～13 行声明了小球的实时水平和竖直方向上的速度 v_x 和 v_y。
- 代码第 15～16 行声明的 timeX 和 timeY 分别代表小球在水平和竖直方向上运动的持续时间，当小球的从一个阶段运行到下一个阶段时（如从下落阶段弹起后转入上抛阶段），timeX 和 timeY 将会被重新赋值。
- 代码第 18 行声明了一个 BallThread 对象，该对象为一个线程，其起到了物理引擎的作用，其将根据物理公式对小球的位置坐标等属性进行修改，从而改变小球的运动轨迹。
- 代码第 19 行声明的 bFall 变量用于标识小球是否已经从木板上落下，在程序运行时屏幕的左上部分会有一个木板，所有的小球从木板开始向右进行平抛运动。bFall 为 false 代表小球仍然在木板上移动，还未落下。
- 代码第 20 行声明了一个 float 型的变量 impactFactor，在小球撞到地面上后会根据该变量的值对小球水平和垂直方向上的速度进行衰减。

（2）编写 Movable 类的构造器和成员方法，在 Movable 的构造器中，将会对其部分成员变量进行初始化，并启动物理引擎 BallThread。Movable 类中包含一个成员方法 drawSelf，负责绘制小球到屏幕上。步骤（1）中代码第 21 行和第 22 行省略的部分如下。

```
1                                                   //构造器：负责初始化主要的成员变量
2     public Movable(int x,int y,int r,Bitmap bitmap){
3           this.startX = x;                        //初始化 startX
4           this.x = x;                             //初始化 x
5           this.startY = y;                        //初始化 startY
6           this.y = y;                             //初始化 y
7           this.r = r;                             //初始化小球的半径
8           this.bitmap = bitmap;                   //初始化小球图片
9           timeX=System.nanoTime();                //获取系统时间初始化 timeX
10          this.v_x = BallView.V_MIN + (int)((BallView.V_MAX-BallView.V_MIN)*Math.random());
11          bt = new BallThread(this);              //创建并启动 BallThread
12          bt.start();
13    }
14                                                  //方法：绘制自己到屏幕上
15    public void drawSelf(Canvas canvas){
16          canvas.drawBitmap(this.bitmap,x, y, null);
17    }
```

- 代码第 10 行为初始化小球水平方向速度的操作。这里用到了后面要介绍到的 BallView 类的两个静态常量 V_MIN 和 V_MAX，其分别代表小球水平方向速度的最小值和最大值。本行代码执行后小球将取处于 V_MIN 和 V_MAX 之间的某个数值作为水平速度。
- 代码第 15～17 行定义了成员方法 drawSelf，其接收一个 Canvas 画布对象，并根据成员变

量 x 和 y 的值将自己画出。

7.2.2　开发物理引擎 BallThread 类的代码

在前面的小节完成了可移动物体 Movable 类的代码的开发，下面来进行物理引擎 BallThread 类的开发，该类继承自 Thread 类。在程序中，每个 Movable 对象都将会有一个独立的 BallThread 对象。其主要的功能是改变小球的运动轨迹。

在对 BallThread 类进行开发之前，先解释一下本例中物理引擎的工作机制，了解其是如何改变小球的运动轨迹的。这样有助于理解后面的代码。

- 坐标系统，本例中采用的坐标系统和屏幕绘制的坐标系统一致，y 轴方向向下为正，x 轴方向向右为正。小球各方向速度的正负也表示其方向向下或向上。
- 运动阶段，在本例中将小球的运动按照竖直方向的速度分为若干个阶段，每个阶段中小球在竖直方向上的速度的大小或者是一直增大（下落），或者是一直减小（上升）。即每当小球在竖直方向上的速度发生改变时（如撞地或达到空中最高点），小球就结束该阶段的运动进入一个新的阶段。
- 数值计算，在每个阶段开始时，都会记录下开始的时间，同时还会记录在这个阶段小球的初始 x、y 坐标，初始 x、y 方向上的速度等物理量。之后在这个阶段的运动中，小球的各项实时数据都根据这些记录的物理量以及当前时间计算得出。
- 为零判断，在小球上升的运动中和小球撞击地面后，都需要判断小球的速度是否为零。但是不同于真实的世界，在程序中小球的各项物理量都是离散的（即每隔固定的时间计算出这些物理量的值），小球实际的运动轨迹为一个个离散点。这种情况下如果还采用判断是否为零的方式就有可能出现错误（如在前一次的计算中小球速度为正，下一次的计算速度为负，跳过了速度为零这个转折点，小球将永远不可能出现为零的时刻）。所以在本程序中使用阈值的方式，小球的速度一旦小于某个阈值，就将其认定为零。

介绍完 BallThread 这个物理引擎的工作机制后，就可以进行 BallThread 类的开发了，可以按照如下的步骤来进行。

（1）在 src/wyf/wpf 目录下创建文件 BallThread.java，并在其中编写 BallThread 类的代码。首先声明 BallThread 类的成员变量，编写构造器，代码如下。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_1/src/wyf/wpf 目录下的 BallThread.java。

```
1    package wyf.wpf;                          //声明包语句
2                                              //继承自 Thread 的线程类，负责修改球的位置坐标
3    public class BallThread extends Thread{
4         Movable father;                      //Movable 对象引用
5         boolean flag = false;                //线程执行标志位
6         int sleepSpan = 40;                  //休眠时间
7         float g = 200;                       //球下落的加速度
8         double current;                      //记录当前时间
9                                              //构造器：初始化 Movable 对象引用及线程、执行标志位
10        public BallThread(Movable father){
11             this.father = father;
12             this.flag = true;               //设置线程执行的标志位为 true
13        }
14                                             ……//此处省略 run 方法，将在随后的步骤中补全
15   }
```

- 代码第 7 行声明了代表程序中重力加速度的变量 g 并初始化为 200，这是经过测试确定的一个比较合理的数值。
- 代码第 10～13 行为 BallThread 类的构造器，在构造器中初始化了 Movable 对象并将线程的执行标志位设为 true。

（2）编写线程的 run 方法中小球运动的代码，BallThread 继承自 Thread 类，需要重写 run 方法。BallThread 的 run 方法为物理引擎最核心的部分，其主要进行两项工作，按照物理公式移动改变小球的位置和检测并处理小球达到最高点以及撞击地面的事件。

首先来看物理引擎是如何计算并改变小球位置的，其代码如下。

```
1    //方法：负责根据物理公式修改小球位置
2    public void run(){
3         while(flag){
4              current = System.nanoTime();            //获取当前时间，单位为纳秒
5                                                      //处理水平方向上的运动
6              double timeSpanX = (double)((current-father.timeX)/1000/1000/1000);
                                                       //获取水平方向走过的时间
7              father.x = (int)(father.startX + father.v_x * timeSpanX);
8                                                      //处理竖直方向上的运动
9              if(father.bFall){                       //判断球是否已经移出挡板
10                  double timeSpanY = (double)((current - father.timeY)/1000/1000/1000);
11                  father.y = (int)(father.startY + father.startVY * timeSpanY +
                    timeSpanY*timeSpanY* g/2);
12                  father.v_y = (float)(father.startVY + g*timeSpanY);
13                                        ……//此处省略检测和处理特殊事件的代码，将在随后
                                            //的步骤补全
14        }
15        else if(father.x + father.r/2 >= BallView.WOOD_EDGE){
                                                //判断球是否移出了挡板
16             father.timeY = System.nanoTime();
                                                //记录球竖直方向上的开始运动时间
17             father.bFall = true;             //设置表示是否开始下落标志位
18        }
19        try{
20             Thread.sleep(sleepSpan);         //休眠一段时间
21        }
22        catch(Exception e){                   //捕获异常
23             e.printStackTrace();             //打印异常信息
24        }}}
```

- 代码第 4 行首先获取系统当前时间，单位为纳秒。该值将会在计算水平方向坐标 x 和竖直方向坐标 y 的时候用到。
- 代码第 6～7 行进行的工作是处理小球在水平方向上的运动。其中第 6 行先根据当前时间获得本阶段中从开始到现在小球在水平方向上运动的时间。代码第 7 行首先计算出在该运动阶段中到目前为止小球走过的位移，然后加上小球在该阶段中的初始位移，最终求出小球此时此刻在 x 方向上的准确坐标。
- 代码第 7～18 行进行的工作是处理小球在竖直方向上的运动。首先在代码第 9 行检查小球的 bFall 是否为 true，如果为 true 表明小球已经在下落，可以进行物理计算。如果 bFall 为 false 说明小球还在木板上移动，这时检查需不需要将其设置为 true，即在代码第 15 行检查小球的 x 坐标是否已满足从木板上落下的条件。如果满足条件则将小球 bFall 设置为 true，并获取当前时间作为小

球第一个下落阶段的初始时间。

- 在代码第 15 行使用了 BallView 类的静态常量 WOOD_EDGE，该值记录了屏幕中木板图片的右边沿在屏幕中的位置，小球的水平位置超过该值就需要下落了。
- 代码第 10～12 行为小球正常下落时竖直方向上的坐标变化。其中第 10 行先获取小球到目前为止在该阶段运动的时间，然后在第 11 行通过公式 $S = v_0t + 1/2at^2$ 计算出小球 y 轴方向的位置坐标。代码第 12 行通过公式 $v_t = v_0 + at$ 计算小球竖直方向上的实时速度。

（3）小球每次进行完竖直方向上的运动，就会检测是否满足撞击地面或达到空中最高点的条件，如果满足条件，就需要进行相应的处理，检测和处理事件的代码在步骤（2）代码的第 13 行被省略，在本步骤中将会对其进行详细解释，实现代码如下。

```
1     //判断小球是否到达最高点
2     if(father.startVY < 0 && Math.abs(father.v_y) <= BallView.UP_ZERO){
3          father.timeY = System.nanoTime();        //设置新的运动阶段竖直方向上的开始时间
4          father.v_y = 0;                          //设置新的运动阶段竖直方向上的实时速度
5          father.startVY = 0;                      //设置新的运动阶段竖直方向上的初始速度
6          father.startY = father.y;                //设置新的运动阶段竖直方向上的初始位置
7     }
8     //判断小球是否撞地
9     if(father.y + father.r*2 >= BallView.GROUND_LING && father.v_y >0){
                                                    //判断撞地条件
10         //改变水平方向的速度
11         father.v_x = father.v_x * (1-father.impactFactor);   //衰减水平方向上的速度
12         //改变竖直方向的速度
13         father.v_y = 0 - father.v_y * (1-father.impactFactor);
                                                    //衰减竖直方向上的速度并改变方向
14         if(Math.abs(father.v_y) < BallView.DOWN_ZERO){
                                                    //判断撞地衰减后的速度，太小就停止
15             this.flag = false;
16         }
17         else{                                    //撞地后的速度还可以弹起继续下一阶段的运动
18             //撞地之后水平方向的变化
19             father.startX = father.x;            //设置新的运动阶段的水平方向的起始位置
20             father.timeX = System.nanoTime(); //设置新的运动阶段的水平方向的开始时间
21             //撞地之后竖直方向的变化
22             father.startY = father.y;            //设置新的运动阶段竖直方向上的起始位置
23             father.timeY = System.nanoTime(); //设置新的运动阶段竖直方向开始运动的时间
24             father.startVY = father.v_y;         //设置新的运动阶段竖直方向上的初速度
25         }
26    }
```

- 代码第 2～7 行检测小球是否达到了最高点并进行相应处理。第 2 行代码对小球是否到达最高点进行判断，判断条件为小球竖直方向上速度为负（即向上），且其值小于某一个阈值。这里用到了 BallView 类的静态常量 UP_ZERO。
- 代码第 3～6 行为当小球达到最高点时进入下一个运动阶段前进行的相关物理量的初始化工作。第 3 行将竖直方向运动的开始时间设为当前时间，第 4 行将小球竖直方向上的速度设为 0，第 5 行将小球竖直方向上的初始速度设为 0，第 6 行将竖直方向上的起始位置设为当前位置。
- 代码第 9～25 行检测小球是否撞地并进行相应处理。第 9 行代码对小球是否撞地进行了判断，判断条件为小球竖直方向上的速度是否为正（向下），且小球的下边沿在屏幕上的坐标是否大

于了地平面的坐标，这里用到了 BallView 类的静态常量 GROUND_LING，该值代表地平面在屏幕上的 y 轴坐标。

- 代码第 11 行和第 13 行为小球撞地后水平方向和竖直方向上速度的改变。第 10 行为小球水平方向上速度的衰减；第 11 行为小球竖直方向上速度的衰减，并改变其方向。
- 代码第 14 行对小球衰减后的竖直方向上的速度进行判断，检测其大小是否已经小于阈值 DOWN_ZERO，如果满足条件，则通过第 15 行的代码停止物理引擎线程的执行，之后小球将不再移动。
- 当小球衰减后的竖直方向上的速度还足够大时，就需要进入下一个运动阶段了，代码第 17～25 行为对下一个运动阶段开始前的初始化操作。其中 19～20 行为水平方向上进入新的运动阶段的初始化工作，第 22～24 行为竖直方向上进入新的运动阶段的初始化工作。

说明 通过本小节对物理引擎的设计以及实现的介绍，读者大概也体会到了物理引擎设计的复杂性，因此更加需要对物理学的理论知识有足够的了解和一定的驾驭能力。

7.2.3 视图类——开发 BallView 类的代码

前面两个小节介绍了可移动物体 Movable 类和物理引擎 BallThread 类的开发，完成这两部核心环节的工作后，下面的工作将稍微轻松一些了。本小节就来介绍负责画面渲染的视图类 BallView 代码的开发，其步骤如下。

（1）在 src/wyf/wpf 目录下新建文件 BallView.java，并在其中开发 BallView 类的代码。在 BallView 类中声明了一些物理计算时要使用的静态常量，同时还声明了程序中要绘制的图片资源以及要绘制的小球对象列表，其主框架代码如下。

提示 BallView 类继承自 android.view 包下 SurfaceView 类，SurfaceView 不同于普通的 View，其具有不同的绘制机理，适合用于开发游戏程序，本书后面的游戏案例中均采用 SurfaceView。使用 SurfaceView 需要实现 SurfaceHolder.Callback 接口，该接口可以对 SurfaceView 进行编辑等操作以及监控 SurfaceView 的变化。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_1/src/wyf/wpf 目录下的 BallView.java。

```
1    package wyf.wpf;                                  //声明包语句
2    import java.util.ArrayList;                       //引入相关类
3    import java.util.Random;                          //引入相关类
4    import android.content.Context;                   //引入相关类
5    import android.content.res.Resources;             //引入相关类
6    import android.graphics.Bitmap;                   //引入相关类
7    import android.graphics.BitmapFactory;            //引入相关类
8    import android.graphics.Canvas;                   //引入相关类
9    import android.graphics.Color;                    //引入相关类
10   import android.graphics.Paint;                    //引入相关类
11   import android.view.SurfaceHolder;                //引入相关类
12   import android.view.SurfaceView;                  //引入相关类
13   //继承自 SurfaceView 的子类
14   public class BallView extends SurfaceView implements SurfaceHolder.Callback{
15        public static final int V_MAX=35;            //小球水平速度的最大值
```

```
16        public static final int V_MIN=15;        //小球竖直速度的最大值
17        public static final int WOOD_EDGE = 60;//木板的右边沿的 x 坐标
18        public static final int GROUND_LING = 450;
                                        //游戏中代表地面 y 坐标，小球下落到此会弹起
19        public static final int UP_ZERO = 30;
                                        //小球在上升过程中，如果速度大小小于该值就算为 0
20        public static final int DOWN_ZERO = 60;
                                        //小球在撞击地面后，如果速度大小小于该值就算为 0
21        Bitmap [] bitmapArray = new Bitmap[6]; //各种颜色形状的小球图片引用
22        Bitmap bmpBack;                         //背景图片对象
23        Bitmap bmpWood;                         //木板图片对象
24        String fps="FPS:N/A";                 //用于显示帧速率的字符串，初始时为："FPS:N/A"
25        int ballNumber =8;                      //小球数目
26        ArrayList<Movable> alMovable = new ArrayList<Movable>();//小球对象数组
27        DrawThread dt;                          //后台屏幕绘制线程
28        //构造器
29        public BallView(Context activity){
30             super(activity);                   //调用父类构造器
31             getHolder().addCallback(this);
32             initBitmaps(getResources());       //初始化图片
33             initMovables();                    //初始化小球
34             dt = new DrawThread(this,getHolder());              //初始化重绘线程
35        }
36        ……//此处省略 initBitmaps 方法的代码，将在随后的步骤中补全
37        ……//此处省略 initMovables 方法的代码，将在随后的步骤中补全
38        ……//此处省略 doDraw 方法的代码，将在随后的步骤中补全
39        @Override
40        public void surfaceChanged(SurfaceHolder holder, int format, int width,
41                  int height) {                 //重写 surfaceChanged 方法
42        }
43        @Override
44        public void surfaceCreated(SurfaceHolder holder) {//从写 surfaceCreated 方法
45             if(!dt.isAlive()){                 //若 DrawThread 没有启动，就启动这个线程
46                  dt.start();
47             }
48        }
49        @Override
50        public void surfaceDestroyed(SurfaceHolder holder) {
                                                  //重写 surfaceDestroyed 方法
51             dt.flag = false;                   //停止线程的执行
52             dt = null;                         //将 dt 指向的对象声明为垃圾
53        }
54   }
```

- 代码第 15～20 行声明了一系列的静态常量，这些静态常量的含义已在注释和之前的小节中说明。将这些程序运行中不会改变的数值声明为静态常量要比直接用字面常量的代码可维护性高，当需要修改时也很方便。
- 代码第 21～24 行为程序中需要绘制到屏幕上的图片等资源的对象引用。其中第 21～23 行分别为各种小球的图片对象、背景砖墙的图片对象和木板图片对象。代码第 24 行声明的是用于显示帧速率的字符串对象，帧速率在程序调试中是非常重要的调试数据。
- 代码第 26 行为可移动物体 Movable 对象的集合。
- 代码第 27 行为后台用于刷新屏幕的线程对象，其内容将会在后面的步骤中详细说明。
- 代码第 29～35 行为 BallView 类构造器的代码，其中第 32 和 33 行分别调用了成员方法 initBitmaps 和 initMovables 用于初始化图片资源和 Movable 对象。
- 代码第 31 行为 SurfaceView 的 SurfaceHolder 对象添加了一个回调（Callback）接口的实现，

该接口有 surfaceCreated、surfaceChanged 和 surfaceDestroyed 3 个方法需要实现。代码第 39～53 行为这 3 个方法的实现。

说明 surfaceCreated 方法当 SurfaceView 被创建时调用，所以诸如启动后台刷屏线程的代码需要写在这里。surfaceDestroyed 方法当 SurfaceView 被销毁时调用，一些清理工作可以放在这里进行。surfaceChanged 方法当 SurfaceView 发生尺寸改变时被调用。

（2）开发 BallView 类的主要成员方法，步骤（1）中代码第 36～38 行省略了 initBitmaps、initMovables 和 doDraw 方法的代码，在本步骤中将会开发这些成员方法，代码如下。

```
1    //方法：初始化图片
2    public void initBitmaps(Resources r){
3         bitmapArray[0] = BitmapFactory.decodeResource(r, R.drawable.ball_red_small);
                                                                    //红色较小球
4        bitmapArray[1] = BitmapFactory.decodeResource(r, R.drawable.ball_purple_small);
                                                                    //紫色较小球
5        bitmapArray[2] = BitmapFactory.decodeResource(r, R.drawable.ball_green_small);
                                                                    //绿色较小球
6         bitmapArray[3] = BitmapFactory.decodeResource(r, R.drawable.ball_red);
                                                                    //红色较大球
7         bitmapArray[4] = BitmapFactory.decodeResource(r, R.drawable.ball_purple);
                                                                    //紫色较大球
8         bitmapArray[5] = BitmapFactory.decodeResource(r, R.drawable.ball_green);
                                                                    //绿色较大球
9         bmpBack = BitmapFactory.decodeResource(r, R.drawable.back);
                                                                    //背景砖墙
10        bmpWood = BitmapFactory.decodeResource(r, R.drawable.wood);
                                                                    //木板
11   }
12   //方法：初始化小球
13   public void initMovables(){
14        Random r = new Random();                                  //创建一个 Random 对象
15        for(int i=0;i<ballNumber;i++){
16             int index = r.nextInt(32);                           //产生随机数
17             Bitmap tempBitmap=null;                              //声明一个 Bitmap 图片引用
18             if(i<ballNumber/2){
19                  tempBitmap = bitmapArray[3+index%3];            //如果是初始化前一半球，就从
                                                                    //大球中随机找一个
20             }
21             else{
22                  tempBitmap = bitmapArray[index%3];              //如果是初始化后一半球，就从
                                                                    //小球中随机找一个
23             }
24             //创建 Movable 对象
25             Movable m = new Movable(0,70-tempBitmap.getHeight(),tempBitmap.getWidth()/2,
               tempBitmap);
26             alMovable.add(m);                                    //将新建的 Movable 对象添加
                                                                    //到 ArrayList 列表中
27        }
28   }
29   //方法：绘制程序中所需要的图片等信息
30   public void doDraw(Canvas canvas) {
31        canvas.drawBitmap(bmpBack, 0, 0, null);                   //绘制背景图片
32        canvas.drawBitmap(bmpWood, 0, 60, null);                  //绘制木板图片
33        for(Movable m:alMovable){                   //遍历 Movable 列表，绘制每个 Movable 对象
34             m.drawSelf(canvas);
35        }
```

```
36          Paint p = new Paint();                    //创建画笔对象
37          p.setColor(Color.BLUE);                   //为画笔设置颜色
38          p.setTextSize(18);                        //为画笔设置字体大小
39          p.setAntiAlias(true);                     //设置抗锯齿
40          canvas.drawText(fps, 30, 30, p);          //画出帧速率字符串
41    }
```

- 代码第 1～11 行为 initBitmaps 方法的代码。该方法将程序中会使用到的图片对象创建好并将其引用存放到成员变量中。bitmapArray 数组中存放了大尺寸和小尺寸的红、紫、绿 3 种小球。创建 Movable 对象时将会从数组中任选一个作为 Movable 自己的图片。
- 代码第 12～28 行为 initMovables 方法的代码，该方法主要进行的工作是根据成员变量 ballNumber 的值创建相应个数的 Movable 对象。
- 代码第 13 行创建了一个 Random 对象，因为创建 Movable 对象时小球选择哪张图片是随机产生的。代码第 16 行调用 Random 对象的 nextInt 方法产生了一个随机数，该随机数经过处理将会作为成员变量 bitmapArray 数组的下标。
- 代码 17～23 行是根据产生的随机数找到将来作为 Movable 对象显示图片的那个图片对象。
- 因为 bitmapArray 数组中的图片分为大尺寸和小尺寸，为了在绘制这些图片时，小尺寸图片不会被大尺寸的图片遮挡住，在产生随机图片时强行规定，持有大尺寸图片的 Movable 对象在 ArrayList 中的位置是在持有小尺寸图片的 Movable 的前面。代码第 18 行进行判断的目的就在于此，让 ArrayList 前半部分的 Movable 的图片均为大尺寸。
- 代码第 25 行创建了一个 Movable 对象，该对象的初始 x 坐标为 0，y 坐标为木板的 y 坐标 60 减去小球的直径。
- 代码第 29～41 行为 doDraw 方法的代码，该方法将会在 DrawThread 中被调用。代码第 31～35 行是绘制图片，代码 36～40 行为绘制帧速率字符串。

7.2.4　绘制线程——开发 DrawThread 类的代码

在 7.2.3 小节 BallView 类的代码中提到了 DrawThread 对象，DrawThread 类功能比较简单，主要负责定时重绘屏幕和计算帧速率，本小节介绍 DrawThread 对象的开发，其步骤如下。

（1）在项目 Sample_7_1 的 src/wyf/wpf 目录下创建文件 DrawThread.java，并在其中声明 DrawThread 类成员变量，开发构造器，其代码如下所列。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_1/src/wyf/wpf 目录下的 DrawThread.java。

```
1    package wyf.wpf;                                    //声明包语句
2    import android.graphics.Canvas;                     //引入相关类
3    import android.view.SurfaceHolder;                  //引入相关类
4    public class DrawThread extends Thread{
5         BallView bv;                                   //BallView 对象引用
6         SurfaceHolder surfaceHolder;                   //SurfaceHolder 对象引用
7         boolean flag=false;                            //线程执行标志位
8         int sleepSpan = 30;                            //休眠时间
9         long start = System.nanoTime();                //记录起始时间，该变量用于计算帧速率
10        int count=0;                                   //记录帧数，该变量用于计算帧速率
11                                                       //构造器
12        public DrawThread(BallView bv,SurfaceHolder surfaceHolder){
13             this.bv = bv;                             //为 BallView 对象应用赋值
14             this.surfaceHolder = surfaceHolder;       //为 SurfaceHolder 对象应用赋值
```

```
15                this.flag = true;                    //设置标志位
16            }
17            ......                                   //此处省略 run 方法的代码，将在随后的步骤补全
18        }
```

● 代码第 9 行和第 10 行声明了用于计算帧速率的两个成员变量，start 用于记录每次计算的起始时间，count 用于记录帧数，当屏幕画出 20 帧时计算一次帧速率。

● 代码第 12～16 行为 DrawThread 类的构造器，在构造器中主要是为成员变量 bv 和 surfaceHolder 赋值，同时将线程执行标志位设为 true。

（2）开发 run 方法的代码，run 方法为线程的执行方法，DrawThread 类的主要功能也是由 run 方法来实现的，其代码如下。

```
1     //方法：线程的执行方法，用于绘制屏幕和计算帧速率
2     public void run(){
3         Canvas canvas = null;                          //声明一个 Canvas 对象
4         while(flag){
5             try{
6                 canvas = surfaceHolder.lockCanvas(null);  //获取 BallView 的画布
7                 synchronized(surfaceHolder){
8                     bv.doDraw(canvas);             //调用BallView的doDraw方法进行绘制
9                 }
10            }
11            catch(Exception e){
12                e.printStackTrace();               //捕获并打印异常
13            }
14            finally{
15                if(canvas != null){                //如果 canvas 不为空
16                    surfaceHolder.unlockCanvasAndPost(canvas);
                                                     //surfaceHolder 解锁，并将画布对象传回
17                }
18            }
19            this.count++;
20            if(count == 20){                       //如果计满 20 帧
21                count = 0;                         //清空计数器
22                long tempStamp = System.nanoTime();  //获取当前时间
23                long span = tempStamp - start;     //获取时间间隔
24                start = tempStamp;                 //为 start 重新赋值
25                double fps = Math.round(100000000000.0/span*20)/100.0;
                                                     //计算帧速率
26                bv.fps = "FPS:"+fps;               //将计算出的帧速率设置到 BallView
                                                     //的相应字符串对象中
27            }
28            try{
29                Thread.sleep(sleepSpan);           //线程休眠一段时间
30            }
31            catch(Exception e){
32                e.printStackTrace();               //捕获并打印异常
33            }}}
```

● 代码第 3～18 行为进行 BallView 屏幕重绘的代码。其实现机制是先将 BallView 的画布加锁，然后调用 BallView 的 doDraw 方法对 BallView 的画布进行重新绘制。最后解锁 BallView 的画布并将其传回。

● 代码第 19～34 行为计算帧速率的代码。本例中计算帧速率的方法是，首先求出程序绘制 20 帧所消耗的时间 span，然后计算 100s 内能够包含几个 span。100s 内包含的 span 的个数乘以 20，

就能得出 100s 内能够绘制几帧，再除以 100 即可求得 1s 内绘制的帧数，即帧速率 FPS。

说明　帧速率是调试游戏等程序时一个比较重要的参数，建议读者在以后的程序调试中采用打印帧速率的方式来查找和解决问题。

7.2.5　开发 Activity 部分的代码

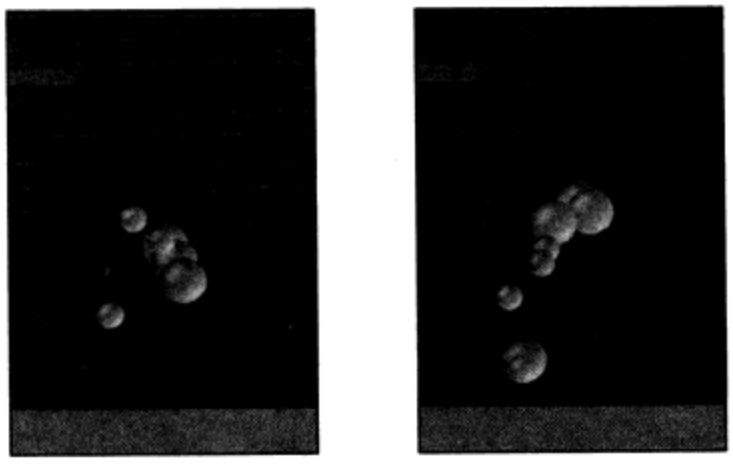
前面的小节将程序中主要的几个功能类代码的开发做了介绍，最后来看看 Activity 代码的开发。在本程序中，Activity 的代码比较简单，打开项目 Sample_7_1 的 src/wyf/wpf 目录下的 Sample_7_1.java，将其代码替换为如下代码。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_1/src/wyf/wpf 目录下的 Sample_7_1.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    import android.os.Bundle;                                 //引入相关类
4    import android.view.Window;                               //引入相关类
5    import android.view.WindowManager;                        //引入相关类
6    //继承自 Activity 的引用
7    public class Sample_7_1 extends Activity {
8      BallView bv;                                            //BallView 对象引用
9        @Override
10       public void onCreate(Bundle savedInstanceState) {
11           super.onCreate(savedInstanceState);
12           requestWindowFeature(Window.FEATURE_NO_TITLE);    //设置不显示标题
13           getWindow().setFlags(                             //设置为全屏模式
14               WindowManager.LayoutParams.FLAG_FULLSCREEN,
15               WindowManager.LayoutParams.FLAG_FULLSCREEN
16           );
17           bv = new BallView(this);                          //创建 BallView 对象
18           setContentView(bv);                               //设置 BallView 对象为屏幕
19       }
20   }
```

- 代码第 8 行声明了一个 BallView 对象，并在第 16 和第 17 行将其初始化并设置为 Activity 的当前屏幕。
- 代码第 12～15 行为设置全屏的代码，其中第 12 行去掉了标题栏，第 13～15 行去掉了手机屏幕最上方显示手机信号和电量等信息的信息栏。

到此整个程序的代码已经开发完毕，下面运行本程序，其效果如图 7-6 所示。



▲图 7-6　物理小球程序运行效果图

7.3 粒子系统

在 7.2 小节通过一个物理小球的例子向读者讲解如何在游戏编程中应用物理学知识为游戏画面添彩。本小节将会以焰火和瀑布为例，对游戏开发中另外一个常用到的数学物理应用——粒子系统的开发做一个简单地介绍。

粒子系统与物理小球有类似的地方，其都是通过数学方法和物理公式模拟客观世界中物体的运动轨迹。不同的是物理小球更强调个体运动，而焰火粒子等粒子系统更注重整体感觉，这点区别从二者的代码中也有所体现。本书在随后的几个小节中，首先将会详细介绍焰火粒子系统的开发，并在之后介绍如何在焰火粒子系统的基础上开发瀑布粒子系统。

7.3.1 粒子对象类——开发粒子对象 Particle 类和粒子集合 ParticleSet 类

在本例中，每个火焰粒子为一个 Particle 类的对象，程序中产生的所有 Particle 对象都由一个 ParticleSet 对象来管理。本小节将按照如下的步骤完成这两个类代码的开发。

（1）在 Eclipse 中新建一个项目 Sample_7_2，在 src/wyf/wpf 目录下新建一个 Particle.java 文件，在其中输入如下代码。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_2/src/wyf/wpf 目录下的 Particle.java。

```
1    package wyf.wpf;                                          //声明包语句
2    //每个 Particle 对象代表一个粒子对象
3    public class Particle{
4         int color;                                           //粒子颜色
5         int r;                                               //粒子半径
6         double vertical_v;                                   //垂直速度
7         double horizontal_v;                                 //水平速度
8         int startX;                                          //初始 x 坐标
9         int startY;                                          //初始 y 坐标
10        int x;                                               //实时 x 坐标
11        int y;                                               //实时 y 坐标
12        double startTime;                                    //起始时间
13        //构造器，初始化成员变量
14        public Particle(int color, int r, double vertical_v,
15                  double horizontal_v, int x, int y, double startTime){
16             this.color = color;                             //初始化粒子颜色
17             this.r = r;                                     //初始化粒子半径
18             this.vertical_v = vertical_v;                   //初始化竖直方向速度
19             this.horizontal_v = horizontal_v;               //初始化水平方向上速度
20             this.startX = x;                                //初始化开始位置的 x 坐标
21             this.startY = y;                                //初始化开始位置的 y 坐标
22             this.x = x;                                     //初始化实时 x 坐标
23             this.y = y;                                     //初始化实时 y 坐标
24             this.startTime = startTime;                     //初始化开始运动的时间
25        }
26   }
```

说明　Particle 类的开发比较简单，其中主要声明了一些用于计算粒子位置的物理量，并编写了能够对这些变量进行初始化操作的有参构造器。

（2）在 src/wyf/wpf 目录下新建一个文件 ParticleSet.java，并在其中编写 ParticleSet 类的代码。ParticleSet 类的主要功能是存放所有的粒子对象，并提供成员方法用于添加粒子，其代码如下。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_2/src/wyf/wpf 目录下的 ParticleSet.java。

```
1    package wyf.wpf;                                    //声明包语句
2    import java.util.ArrayList;                         //引入相关类
3    import android.graphics.Color;                      //引入相关类
4    //ParticleSet 类负责管理和添加粒子对象
5    public class ParticleSet{
6         ArrayList<Particle> particleSet;               //用于存放 Particle 对象的集合
7         //构造器，初始化粒子集合
8         public ParticleSet(){
9              particleSet = new ArrayList<Particle>();
10        }
11        //方法：向粒子集合中添加指定个数的粒子对象
12        public void add(int count, double startTime){
13             for(int i=0; i<count; i++){               //创建 count 个数的 Particle 对象
14                  int tempColor = this.getColor(i);//获得粒子颜色
15                  int tempR = 1;                       //粒子半径
16                  double tempv_v = -30 + 10*(Math.random());
                                                         //随机产生粒子竖直方向上的速度
17                  double tempv_h = 10 - 20*(Math.random());
                                                         //随机产生粒子水平方向上的速度
18                  int tempX = 160;                     //粒子的 X 坐标是固定的
19                  int tempY = (int)(100 - 10*(Math.random()));
                                                         //随机产生粒子 Y 坐标，90～100 之间
20                                                       //创建 Particle 对象
21                  Particle particle = new Particle(tempColor, tempR,
22                       tempv_v, tempv_h, tempX, tempY, startTime);
23                  particleSet.add(particle);           //将创建好的 Particle 对象添加到列表中
24             }
25        }
26        //方法：获取指定索引的颜色
27        public int getColor(int i){
28             int color = Color.RED;
29             switch(i%4){                              //对 i 进行分支判断
30             case 0:
31                  color = Color.RED;                   //将颜色设为红色
32                  break;
33             case 1:
34                  color = Color.GREEN;                 //将颜色设为绿色
35                  break;
36             case 2:
37                  color = Color.YELLOW;                //将颜色设为黄色
38                  break;
39             case 3:
40                  color = Color.GRAY;                  //将颜色设为灰色
41                  break;
42             }
43             return color;                             //返回得到的颜色
44        }
45   }
```

- 代码第 6 行声明了一个 ArrayList 对象，该对象用来存储程序中的粒子对象。代码第 8～10 行为 ParticleSet 类的构造器，其主要的工作是初始化成员变量 particleSet。
- 代码第 11～25 行定义了成员方法 add，该方法根据传入的参数 count 和 startTime，创建指定数个以指定时间为起始时间的 Particle 对象。

- 代码第 14～19 行为创建 Particle 对象前的准备工作。其中第 14 行调用方法 getColor 获得要创建的 Particle 对象的颜色，第 15 行为定义其半径的代码，第 16 和 17 行在指定范围内随机产生粒子竖直方向和水平方向上的初速度。代码 18～19 行确定了粒子的 x、y 坐标。
- 代码第 26～44 行定义了 getColor 方法，该方法接收参数 i 并根据其值返回红、绿、黄、灰 4 种颜色中的一种。

说明 在粒子系统中，采用的坐标系仍然是以屏幕左上角为原点，以向右为 x 轴正向，以向下为 y 轴正向。所以代码第 16 行产生的粒子竖直方向上的初速度为−30～−20，方向向上。第 17 行产生的粒子水平方向上的初速度为−10～10，方向向左或向右。

7.3.2 开发焰火粒子系统的物理引擎 ParticleThread 类的代码

前面的小节完成了粒子类 Particle 和粒子集合类 ParticleSet 的开发，本小节开发火焰粒子系统的物理引擎和负责显示粒子的视图类 ParticleView，其步骤如下。

在 src/wyf/wpf 目录下新建一个文件 ParticleThread.java，在其中编写粒子系统物理引擎的代码，ParticleThread 类继承自 Thread 类，主要对粒子集合 ParticleSet 类对象进行操作，其代码如下。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_2/src/wyf/wpf 目录下的 ParticleThread. java。

```
1    package wyf.wpf;                                          //声明包语句
2    import java.util.ArrayList;                               //引入相关类
3    //继承自 Thread 的类，主要功能是添加粒子并修改粒子的位置
4    public class ParticleThread extends Thread{
5         boolean flag;                                        //线程执行标志位
6         ParticleView father;                                 //ParticleView 对象引用
7         int sleepSpan = 80;                                  //线程休眠时间
8         double time = 0;                                     //物理引擎的时间轴
9         double span = 0.15;                                  //每次计算粒子的位移时采用的时间间隔
10        //ParticleThread 类的构造器
11        public ParticleThread(ParticleView father){
12             this.father = father;
13             this.flag = true;                               //设置线程执行标志位为 true
14        }
15        //线程的执行方法
16        public void run(){
17             while(flag){
18                  father.ps.add(5, time);                    //每次添加 5 个粒子
19                  ArrayList<Particle> tempSet = father.ps.particleSet;
                                                               //获取粒子集合
20                  int count = tempSet.size();                //记录粒子集合的大小
21                  for(int i=0; i<count; i++){                //遍历粒子集合，修改其轨迹
22                      Particle particle = tempSet.get(i);
23                      double timeSpan = time - particle.startTime;
                                                               //计算从程序开始到现在经过的时间
24                      int tempx = (int)(particle.startX+particle.horizontal_v*timeSpan);
                                                               //计算出粒子的 X 坐标
25                      int tempy = (int)(particle.startY + 4.9*timeSpan*timeSpan
26                                      + particle.vertical_v*timeSpan);
                                                               //计算出粒子的 Y 坐标
27                      if(tempy>300){                         //如果粒子超过屏幕下边沿
28                            tempSet.remove(particle);
                                                               //从粒子集合中移除该 Particle 对象
29                            count = tempSet.size();          //重新设置粒子个数
```

```
30                      }
31                      particle.x = tempx;                    //修改粒子的 X 坐标
32                      particle.y = tempy;                    //修改粒子的 Y 坐标
33                  }
34                  time += span;                              //将时间延长
35                  try{
36                      Thread.sleep(sleepSpan);               //休眠一段时间
37                  }
38                  catch(Exception e){
39                      e.printStackTrace();                   //捕获并打印异常
40                  }}}}
```

- 代码第 5～9 行声明了 ParticleThread 类的成员变量，主要的成员变量在第 11～14 行的构造器中被初始化。
- 代码第 16～42 行定义了线程的 run 方法，该方法主要的功能是每隔固定的时间向粒子集合 ParticleSet 对象中添加 5 个粒子，并修改粒子集合中现有粒子的位置坐标。当粒子集合中的粒子对象的 y 坐标超过一定数值之后，将会从粒子集合中删除。
- 代码第 21～33 行进行的工作是遍历粒子集合，修改每个粒子的位置。首先在代码第 23 行获取粒子从诞生到目前经过的时间 timeSpan，然后在第 24 行根据粒子的初始 x 坐标和初始水平速度以及 timeSpan 计算出粒子的实时 x 坐标。类似地，在代码第 25 行根据粒子的初始 y 坐标、初始竖直速度、重力加速度（在此取 9.8）以及 timeSpan 计算出粒子的实时 y 坐标。
- 代码第 27～30 行对新计算出的粒子 y 坐标进行判断，如果其值超过某个值，就将其从粒子集合中移除并及时更改 count 的值。在这里用到了 ParticleView 类的静态常量 DIE_OUT_LINE，粒子将在其 y 坐标达到或超过该值时被移除。
- 代码第 34 行将记录物理引擎时间的变量 time 进行增加操作，表示时间的流逝。

说明　在这里需要注意一下，本粒子系统在进行物理计算时，并没有采用获取系统时间的方式，而是自己定义了一个时间轴（成员变量 time）。这样可以自己确定时间轴行进的快慢程度（通过改变成员变量 span 的值），而不必依赖于系统的时间。

7.3.3　视图类——开发视图类 ParticleView 及其相关类

本小节讲解程序中的视图类 ParticleView 及其相关类的开发过程，在粒子系统中，ParticleView 及其相关类主要的功能是，在屏幕上绘制出粒子的运动，开发步骤如下。

（1）在 src/wyf/wpf 目录下新建文件 ParticleView.java，并在其中编写 ParticleView 类的代码，该类继承自 SurfaceView 并实现了 SurfaceHolder.Callback 接口。首先进行声明成员变量、开发构造器及接口方法的工作，其代码如下。

代码位置：见随书光盘中源代码/第 7 章/Sample_7_2/src/wyf/wpf 目录下的 ParticleView. java。

```
1    package wyf.wpf;                          //声明包语句
2    import java.util.ArrayList;               //引入相关类
3    import android.content.Context;           //引入相关类
4    import android.graphics.Canvas;           //引入相关类
5    import android.graphics.Color;            //引入相关类
6    import android.graphics.Paint;            //引入相关类
7    import android.graphics.RectF;            //引入相关类
```

```
8    import android.view.SurfaceHolder;                    //引入相关类
9    import android.view.SurfaceView;                      //引入相关类
10   //继承自 SurfaceView 并实现 SurfaceHolder.Callback 接口的类
11   public class ParticleView extends SurfaceView implements SurfaceHolder.Callback{
12        public static final int DIE_OUT_LINE = 300;
                                                           //粒子的 Y 坐标超过该值会从粒子集合中移除
13        DrawThread dt;                                   //后台刷新屏幕线程
14        ParticleSet ps;                                  //ParticleSet 对象引用
15        ParticleThread pt;                               //ParticleThread 对象引用
16        String fps = "FPS:N/A";                          //声明帧速率字符串，初始时为："FPS:N/A"
17        //构造器，初始化主要成员变量
18        public ParticleView(Context context) {
19             super(context);                                  //调用父类构造器
20             this.getHolder().addCallback(this);              //添加 Callback 接口
21             dt = new DrawThread(this, getHolder());          //创建 DrawThread 对象
22             ps = new ParticleSet();                          //创建 ParticleSet 对象
23             pt = new ParticleThread(this);                   //创建 ParticleThread 对象
24        }
25        ……//在此省略绘制方法 doDraw 的代码，将在随后的步骤补全
26        @Override
27        public void surfaceChanged(SurfaceHolder holder, int format, int width,
28                  int height) {                              //重写 surfaceChanged 方法
29        }
30        @Override
31        public void surfaceCreated(SurfaceHolder holder) {//从写 surfaceCreated 方法
32             if(!dt.isAlive()){          //如果 DrawThread 没有启动，就启动这个线程
33                  dt.start();
34             }
35             if(!pt.isAlive()){          //如果 ParticleThread 没有启动，就启动这个线程
36                  pt.start();
37             }
38        }
39        @Override
40        public void surfaceDestroyed(SurfaceHolder holder) {
                                                           //重写 surfaceDestroyed 方法
41             dt.flag = false;                                 //停止线程的执行
42             dt = null;                                       //将 dt 指向的对象声明为垃圾
43             pt.flag = false;                                 //停止线程的执行
44             pt = null;                                       //将 pt 指向的对象声明为垃圾
45        }
46   }
```

- 代码第 12～16 行进行的工作是声明 ParticleView 类的成员变量。代码第 18～24 行为 ParticleView 类的构造器定义代码，其主要的工作是初始化主要的成员变量。
- 代码第 25～42 行分别对 SurfaceHolder.Callback 接口中的 surfaceChanged、surfaceCreated 和 surfaceDestroyed 方法进行了重写。在 surfaceCreated 方法中启动了后台刷屏线程 DrawThread 和物理引擎线程 ParticleThread，在 surfaceDestroyed 方法中则停止了上述线程的执行。

（2）开发绘制方法 doDraw 的代码。doDraw 方法的主要功能是将粒子集合中的粒子绘制到屏幕上，同时输出帧速率。步骤（1）代码第 25 行省略掉的代码如下。

```
1    //方法：绘制屏幕
2    public void doDraw(Canvas canvas){
3         canvas.drawColor(Color.BLACK);                      //清除屏幕
4         ArrayList<Particle> particleSet = ps.particleSet;   //获得 ParticleSet 对象中的粒子集合
                                                              //对象
5         Paint paint = new Paint();                          //创建画笔对象
6         for(int i=0;i<particleSet.size();i++){              //遍历粒子集合，绘制每个粒子
```

```
7                Particle p = particleSet.get(i);
8                paint.setColor(p.color);                    //设置画笔颜色为粒子颜色
9                int tempX = p.x;                            //获得粒子 X 坐标
10               int tempY = p.y;                            //获得粒子 Y 坐标
11               int tempRadius = p.r;                       //获得粒子半径
12               RectF oval = new RectF(tempX, tempY, tempX+2*tempRadius, tempY+2*tempRadius);
13               canvas.drawOval(oval, paint);               //绘制(椭)圆粒子
14           }
15           paint.setColor(Color.WHITE);                    //设置画笔颜色
16           paint.setTextSize(18);                          //设置文字大小
17           paint.setAntiAlias(true);                       //设置抗锯齿
18           canvas.drawText(fps, 15, 15, paint);            //画出帧速率字符串
19       }
```

- 代码第 3 行进行的工作是每次绘制之前进行清除屏幕操作。
- 代码第 5 行创建了一个 Paint 对象，其将用于绘制椭圆粒子和画出帧速率字符串。
- 代码第 6～14 行进行的主要工作是遍历粒子对象的集合，根据粒子的 x、y 坐标和其半径，在代码第 12 行创建一个 RectF 对象，RectF 对象为包含了 4 个点坐标的矩形对象。
- 代码第 13 行根据创建的 RectF 对象画出一个（椭）圆代表粒子，该（椭）圆是 RectF 对象所代表的矩形的内接（椭）圆。
- 代码第 15～18 行为绘制帧速率字符串的代码，这些代码和物理小球系统中帧速率字符串的绘制方式相同，在此不再解释，读者可以查阅前面的讲解。

（3）在 src/wyf/wpf 目录下新建文件 DrawThread.java，并在其中编写 DrawThread 类的代码。DrawThread 类继承自 Thread 类，其主要的功能是每隔固定的时间刷新 ParticleView 和计算帧速率。

刷新 ParticleView 的方式是调用 ParticleView 的 SurfaceHolder 对象，并将 ParticleView 的画布加锁，然后调用 ParticleView 的 doDraw 方法重新绘制 ParticleView 的画布，最后将 ParticleView 的画布解锁并将其传回。

DrawThread 类的代码和前面讲解过的物理小球系统中的 DrawThread 类代码基本相同，由于篇幅有限，本书在此不予列出，读者可以自行查阅随书光盘，代码位置为随书光盘中源代码/第 7 章/Sample_7_2/src/wyf/wpf 目录下的 DrawThread.java。

7.3.4　开发程序 Activity 部分的代码

前面的小节讲解了焰火粒子系统主要功能类的代码的开发，最后来开发程序 Activity 部分的代码。Activity 要进行的工作主要是创建 ParticleView 对象并将其设置为当前显示的屏幕。打开项目 Sample_7_2 的 src/wyf/wpf 目录下的 Sample_7_2.java，并将其原有代码替换为如下代码。

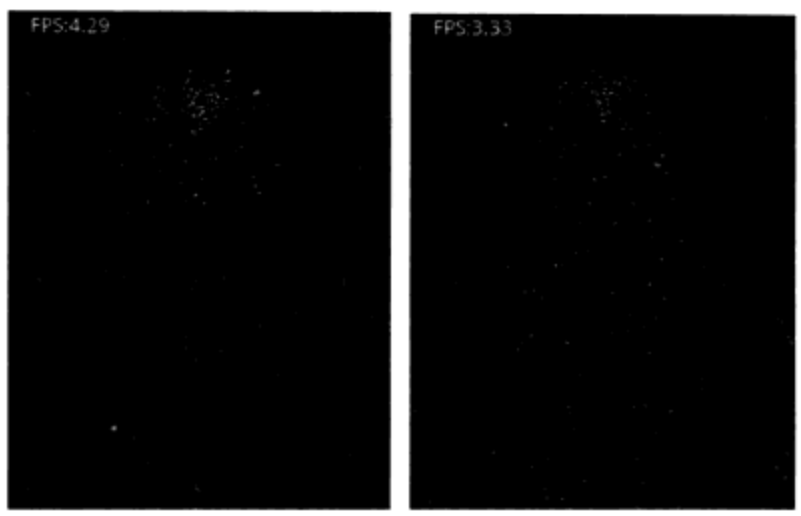
代码位置：见随书光盘中源代码/第 7 章/Sample_7_2/src/wyf/wpf 目录下的 Sample_7_2. java。

```
package wyf.wpf;                                            //声明包语句
import android.app.Activity;                                //引入相关类
import android.os.Bundle;                                   //引入相关类
import android.view.Window;                                 //引入相关类
import android.view.WindowManager;                          //引入相关类
                                                            //继承自 Activity 的子类
public class Sample_7_2 extends Activity {
  @Override
    public void onCreate(Bundle savedInstanceState) { //重写 onCreate 方法
```

```
10          super.onCreate(savedInstanceState);
11          requestWindowFeature(Window.FEATURE_NO_TITLE); //设置不显示标题
12          getWindow().setFlags(                          //设置为全屏模式
13              WindowManager.LayoutParams.FLAG_FULLSCREEN,
14              WindowManager.LayoutParams.FLAG_FULLSCREEN);
15          ParticleView lz = new ParticleView(this);      //创建一个 LiZiView 对象
16          setContentView(lz);                            //将屏幕设置为 ParticleView 对象
17      }
18  }
```

说明 Activity 部分的开发主要是 onCreate 方法的重写，在 onCreate 方法中首先设置屏幕显示为全屏显示，然后创建一个 ParticleView 对象，将其作为屏幕显示的内容。

到此有关焰火粒子系统的全部代码已经开发完成，运行本程序，效果如图 7-7 所示。



▲图 7-7 焰火粒子系统运行效果图

7.3.5 将焰火粒子系统改为瀑布粒子系统

前面的小节分步骤讲解了焰火粒子系统的开发，本小节介绍如何开发瀑布粒子系统。瀑布粒子和焰火粒子十分类似，二者的运动都是带有初速度的下落运动。所不同的是焰火粒子水平方向和竖直方向上的速度均不为零，而瀑布粒子只有水平方向上初速度，竖直方向上的初速度为零。

明白了焰火粒子和瀑布粒子的异同，就可以在焰火粒子系统的基础上进行修改来完成瀑布粒子系统的开发。由于瀑布粒子和焰火粒子只是在生成的时候被赋予的初速度不同，那么只需要修改粒子生成部分的 ParticleSet 类即可，瀑布粒子系统的开发步骤如下。

（1）在 Eclipse 新建一个项目 Sample_7_3，将项目 Sample_7_2 中 src/wyf/wpf 目录下的 Particle.java、ParticleSet.java、ParticleThread.java、ParticleView.java 和 DrawThread.java 复制到项目 Sample_7_3 的 src/wyf/wpf 目录下。

（2）打开项目 Sample_7_3 中 src/wyf/wpf 目录下的 Sample_7_3.java，将项目 Sample_7_2 中 src/wyf/wpf 目录下的 Sample_7_2.java 中的代码全部复制到 Sample_7_3.java 中。

（3）打开项目 Sample_7_3 中 src/wyf/wpf 目录下的 ParticleSet.java，找到用于向粒子集合中添加粒子对象的成员方法 add，将其代码修改为如下代码。

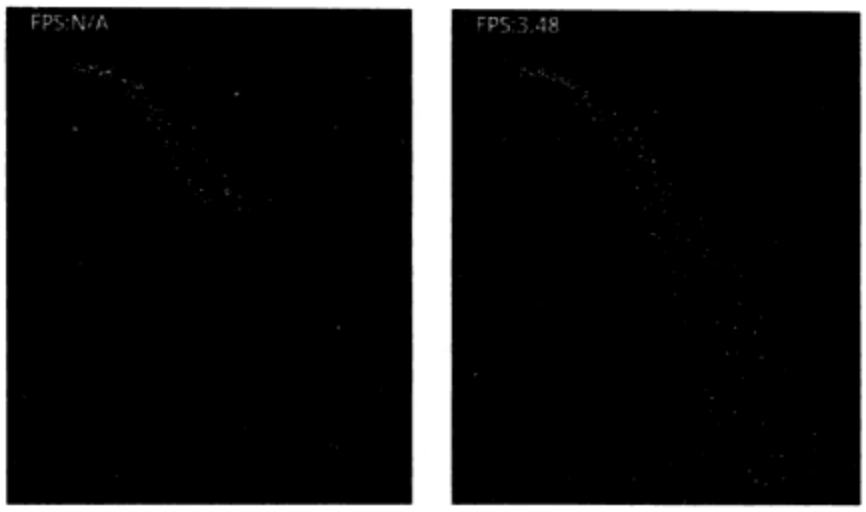
```
1    //方法：向粒子集合中添加指定个数的粒子对象
2    public void add(int count, double startTime){
3         for(int i=0; i<count; i++){                    //创建 count 个数的 Particle 对象
4              int tempColor = this.getColor(i);         //获得粒子颜色
5              int tempR = 1;                            //粒子半径
6              double tempv_v = 0;                       //粒子竖直方向上速度设置为零
7              double tempv_h = 20 + 10*(Math.random());
                                                         //随机产生粒子水平方向上速度
8              int tempX = 50;                           //粒子的 X 坐标是固定的
9              int tempY = (int)(50 - 10*(Math.random()));    //随机产生粒子 Y 坐标
10             //创建 Particle 对象
11             Particle particle = new Particle(tempColor, tempR, tempv_v, tempv_h, tempX,
               tempY, startTime);
```

```
12                  particleSet.add(particle);          //将创建好的 Particle 对象添加到列表中
13              }
14      }
```

上述代码与焰火粒子系统中 ParticleSet 类的 add 方法的不同之处在于如下几处。

- 代码第 6～7 行设置了瀑布粒子的初始水平方向和竖直方向上的速度，其中竖直方向上速度设置为零，水平方向上初速度为 20～30 之间的随机整数。
- 代码第 8～9 行设置了瀑布粒子的初始 x、y 坐标，瀑布粒子的 x 坐标是固定的 50，y 坐标为 40～50 之间的随机整数。

经过以上 3 个步骤的操作，到此瀑布粒子的开发也就完成了，运行 Sample_7_3，效果如图 7-8 所示。



▲图 7-8　瀑布粒子系统运行效果图

7.4 碰撞检测技术

碰撞检测是游戏中必不可缺少的一个功能，无论是角色扮演游戏还是其他类型的游戏，二维还是三维游戏，都会涉及碰撞检测模块的开发。玩家打中怪物、足球撞墙反弹、炮弹击中飞机、玩家捡到宝箱……这些都用到了碰撞检测技术。

碰撞检测技术的实现也是基于数学和物理方面的知识来进行计算和判断的。在游戏开发中，不同的场合有不同的碰撞检测方式，本节将主要讲解碰撞检测在游戏中的使用方式以及使用碰撞检测时的基本原则，最后会具体地介绍几种常用的碰撞检测技术。

7.4.1　碰撞检测技术基础

一般情况下，碰撞检测只会发生在游戏中的实体对象的位置发生了变化之后，如怪物走动、炮弹沿轨道移动、玩家跳起等。不同的碰撞检测其目的也不尽相同，如怪物走动后检测是否遇到玩家，炮弹移动后检测是否打中目标等。

对于移动之后进行碰撞检测的场合，程序中通常按照以下的流程来应用碰撞检测：

- 更新实体对象的位置；
- 进行碰撞检测；
- 如果碰到了，进行相应处理。

说明　上述流程是针对单个实体对象来说的，即每个实体在自己位置更新了之后就进行碰撞检测。还有一种方法是在一类实体的位置全部更新完毕之后，再逐个进行碰撞检测。如将屏幕中所有炮弹全部移动位置，然后判断每个炮弹是否碰到目标，这种方法并不能适用于所有场合。

上述流程中的第三步是对碰撞检测进行处理，最简单的一种处理就是让实体对象后退回到原来的位置，除去这种简单的情况，通常碰撞检测环节和处理碰撞环节是结合在一起的。实现碰撞检测所涉及的内容主要有如下 3 个方面。

- 确定检测对象

游戏在运行中会产生很多实体对象，在进行碰撞检测时并不需要对所有的实体对象都检测一遍，如玩家没必要检测是否和自己发出的子弹碰撞，静止的宝箱也没必要检测是否和另外的宝箱碰撞。所以在开始碰撞检测之前，首先要确定检测的对象是什么。

- 检测是否碰撞

这是碰撞检测的核心环节，在这个环节需要综合考虑游戏本身需求和运行平台的性能问题，合理地选择碰撞检测的算法。

- 处理碰撞

当检测到有碰撞发生时，就需要根据碰撞的类型进行相应的处理，如炮弹在碰到目标后会爆炸并给目标造成伤害。

考虑到游戏的最佳用户体验问题，通常在开发游戏的碰撞检测模块时需要遵循以下几个原则。

- 尽量避免碰撞检测，如果无法避免就尽量减少检测的次数。由于进行碰撞检测基本上都会进行一些数学计算，所以应该尽量少用，减少检测的机会。
- 采用不影响游戏性能的算法尽快检测。对于碰撞检测来说，精确和速度就像是算法设计中的空间复杂度和时间复杂度，二者很难达到两全。所以在游戏开发时要多做衡量，选择一种对于精确程度和执行速度来说都比较适中的算法。
- 碰撞处理要柔和，不要让玩家感到不适应。碰撞检测的处理不能过于生硬，如物体从地面抛向远处，在飞行的时候如果检测到与地面发生了碰撞，不应该立刻让其停止运动，而是就着惯性向前滑动一段距离。

提示

对于以上所介绍的碰撞检测时需要遵循的原则，在随后的小节中将会在需要的位置具体地讲解。

7.4.2 游戏中实体对象之间的碰撞检测

具体来讲，碰撞检测主要分为游戏实体对象（如玩家控制的英雄、怪物、发射的炮弹等）之间的碰撞检测以及游戏实体对象与环境（如游戏场景中的墙、台阶、树等）之间的碰撞检测。本小节将介绍有关游戏中实体对象间碰撞检测的技术。

游戏中实体对象与环境之间的碰撞检测无法偷工减料（否则将会出现怪物穿墙而过的奇怪现象），但是实体间的碰撞检测可以稍加优化。所以在研究实体间碰撞的算法前，需要考虑如何减少待检测实体的个数，一般有如下几种可考虑的方案。

- 静止的实体不负责碰撞检测

如游戏中静止的宝箱不应该定时检测玩家控制的英雄有没有与自己发生碰撞，这项工作应该交给二者中进行移动的一方即玩家自己来负责。

- 只进行单向碰撞检测

如射击游戏中，不应该出现这样的检测算法：敌人射出的子弹会在移动中检测是否遇到了玩家控制的英雄，而玩家控制的英雄在移动中也会检测是否有子弹打中了自己。这种算法首先是多此一举，降低游戏的性能；其次还会出现玩家被一颗子弹打中，受到双倍伤害（进行了两次碰撞处理）的奇怪现象。

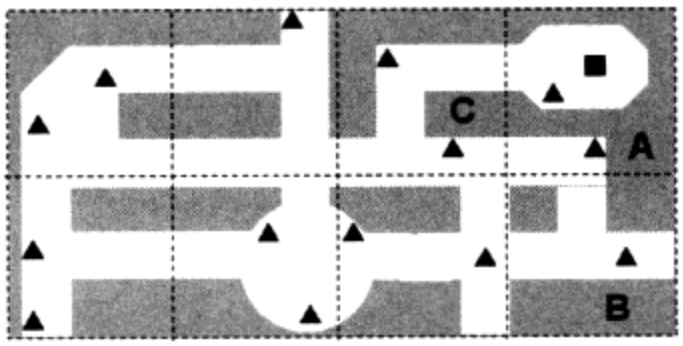
一般来说，碰撞检测应该由两个实体对象中主动的一方来进行。如对于子弹和英雄，是子弹击中英雄而不是英雄迎接子弹，所以应该由子弹负责二者的碰撞检测，而不是“被碰撞”的英雄。

- 距离远的实体对象不进行碰撞检测

在游戏中，如果某两个实体之间的距离太远，在检测碰撞时会将较远的实体忽略，这样会对游戏的执行速度提高不少，实现这种策略主要有如下两种方式。

（1）划分网格法。

将游戏地图划分为若干个格子，每个实体隶属于一个格子单元（如果有横跨多个格子的实体，其隶属格子也只能有一个，如中心点所在的格子）。这样进行碰撞检测时，实体对象只需要检测隶属格子及相邻格子中的实体对象是否与自己发生碰撞即可，如图 7-9 所示。



▲图 7-9　采用划分网格法对游戏地图进行划分

在图 7-9 中所示的游戏地图中，方块代表需要进行碰撞检测的实体，三角形代表被检测的实体，采用网格划分法对游戏地图进行划分后。方块所在的格子为 A，与其相邻的格子为 B 和 C，那么方块在进行碰撞检测的时候只需要与格子 A、B 和 C 中的三角进行碰撞检测即可。

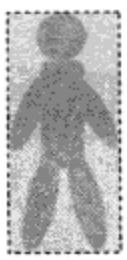
（2）实体排序法。

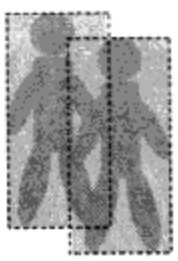
这种方法的实现途径首先是保持被检测实体是有序的，如按照实体某个方向轴的坐标大小排列，这样在对实体序列中某一个实体进行碰撞检测时，只需要检测与该实体在实体序列中相邻的实体即可。不过这种方式需要消耗一定的性能来维护实体的有序性。

在尽可能减少了碰撞检测的次数后，下面将会讲解必须要进行碰撞检测时通常采用的检测算法，主要有矩（圆柱）形检测和圆（球）形检测两种。

1. 矩（圆柱）形检测

这种检测算法是给实体外层套上矩形（二维游戏中采用）或圆柱形（三维游戏中采用），下面以二维游戏为例说明矩形检测的用法。首先为实体套上一个外接矩形框，如图 7-10 所示，在进行实体间碰撞检测时，只需要检测两个实体的外接矩形是否发生了碰撞，如图 7-11 所示。


▲图 7-10　为实体套上矩形框


▲图 7-11　对实体进行矩形检测

具体检测的算法可描述为如下。

- 取两个实体的左上角坐标$(x1,y1)$和$(x2,y2)$以及实体宽度 w 和高度 h。
- 声明 4 个变量 maxX、minX、maxY 和 minY，并将其分别赋值为两实体中 x 坐标的较大值、

x 坐标较小值、y 坐标较大值和 y 坐标较小值。

- 判断是否 $\max X<\min X+w$，且 $\max Y<\min Y+h$。如果满足这两个条件，则说明两个实体发生了碰撞，进行碰撞处理。

上述算法实现比较简单，但是前提条件是进行检测的两个实体的宽度和高度必须相同，有一些游戏中并不满足这个前提条件。那么就应该在上述第二个步骤中多记录几个变量 maxWidth、minWidth、maxHeight 和 minHeight。或者应该采用下面的第二种算法。

- 取两个实体的左上角坐标(x_1,y_1)和(x_2,y_2)以及二者的宽度 w_1、w_2 和高度 h_1、h_2。
- 判断是否 $x_1<x_2+w_2$，且 $x_2<x_1+w_1$，且 $y_1<y_2+h_2$，且 $y_2<y_1+h_1$。如果满足这 4 个条件，则说明两个实体发生了碰撞，进行碰撞处理。

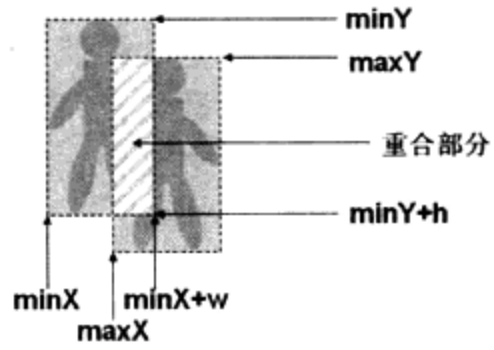
对于二维游戏中的矩形检测，又可以称为边界检测，这种检测在大多数情况能够很好地满足游戏的需要。但是有些情况下两个实体的边界发生碰撞后，实体并没有发生碰撞，如图 7-12 所示。

解决图 7-12 出现的误差问题的一个办法就是，在发生碰撞后计算重合面积，即当检测到实体的矩形框碰撞时，计算两个矩形框的重合面积，只有重合比例（即重合部分面积占整个矩形框面积的百分比）达到一定数值时才认定两个实体发生了碰撞。以上面介绍的第一种矩形框碰撞检测算法为例，检测到矩形框碰撞后重合面积的计算可以按照如下的步骤来进行。

- 计算重合部分的宽度 w，其值为 $\min X+w-\max X$。
- 计算重合部分的高度 h，其值为 $\min Y+h-\max Y$。
- 计算重合部分的面积 $S=w*h$。
- 计算重合比例，如果超过某个值，则说明两个实体之间发生了碰撞，进行碰撞处理。

读者可以参考图 7-13 的说明来理解上述的算法步骤中公式的含义。在矩形框碰撞检测的基础上加入重合面积的判断，这样一来碰撞检测就会比较真实可靠，本书后面要讲解的游戏案例中有一部分正是采用这种算法。

▲图 7-12　矩形碰撞产生的误差



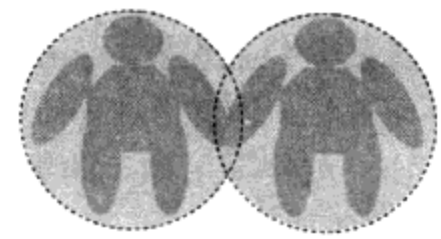
▲图 7-13　计算重合面积示意图

2. 圆（球）形检测

另外一种碰撞检测的算法是给实体对象套上圆形（二维游戏中采用）或球形（三维游戏中采用）的边框，下面以二维游戏为例说明圆形检测的用法。圆形检测就是在需要进行检测的实体上套上一个外接圆，如图 7-14 所示。

进行碰撞检测时，只需要对相关实体的圆形框进行检测，如果两个圆相交，则这两个实体就发生了碰撞，如图 7-15 所示。圆形检测的碰撞检测算法可描述为如下。

▲图 7-14　为实体外层套上圆框

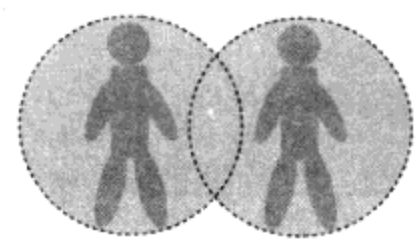


▲图 7-15　对实体间进行圆形检测

- 取两个实体的中心点坐标(x1,y1)和(x2,y2)。
- 取两个实体的半径，求其和为 R。
- 计算两个实体中心点间的距离 D，使用 $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$ 公式。
- 将 D 与 R 对比，如果 D 小于 R，则二者发生了碰撞，进行碰撞处理。

在实际开发中，为了提高性能，往往比较 D 和 R 的平方，这样免去了开方的计算。圆形检测实现起来比较简单，速度也够快，但是不够准确。例如，有一类如图 7-16 所示形状特殊的实体，当对这些实体进行碰撞检测时会出现如图 7-17 的误差。

▲图 7-16　为特殊实体外层套上圆框



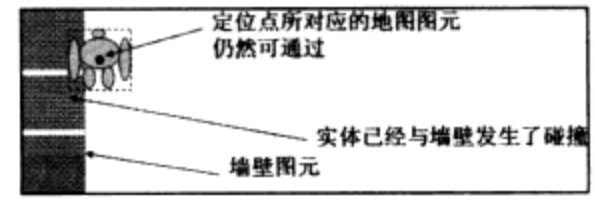
▲图 7-17　使用圆形检测出现的误差

在图 7-14 中，两个实体的圆框已经碰上，但是实体之间并没有发生碰撞，这样就产生了误差。这种误差可能会带给玩家比较糟糕的体验，如游戏中玩家还没有碰到怪物可能就已经受到伤害了。解决这种误差带来的问题，可以在使用圆形检测到碰撞后进行更深入、全面的碰撞检测。

7.4.3　游戏实体对象与环境之间的碰撞检测

前面的小节介绍了游戏中常见的实体对象之间的碰撞检测技术，游戏中的实体对象不但要检测是否与其他实体对象发生碰撞，还需要检测是否与游戏环境发生碰撞。本小节简单介绍一些常用的游戏实体与环境的碰撞检测技术。

很多时候游戏的地图都是由图元（Tile）组成，而游戏中的实体对象与环境的检测即是与所遇到的地图中的图元进行判断。一般的做法是为实体对象设定一个定位点（如定位点在实体中心位置），在与地图进行碰撞检测时，先计算并获得定位点所在的位置对应地图中的那个格子，然后判断该格子是否属于不可通过的图元对象，如果是，则发生碰撞，进行碰撞处理。



▲图 7-18　采用定位点的方式产生的误差

这种算法有一个缺点，就是碰撞检测的可靠程度完全取决于定位点的选取，例如，将定位点选在实体的中心位置，那么可能会出现如图 7-18 所示的情况。实体明明都已经陷入到墙里面了，但是由于定位点隶属的地图图元仍然还是可通过的，碰撞检测的结果将会是没有发生

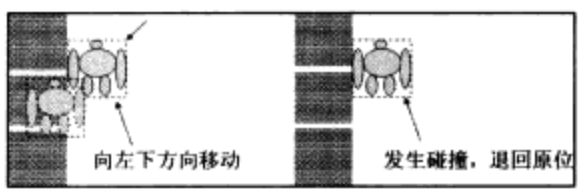
碰撞。

如果将实体对象的定位点设置为左上角，则其在判断左上方向上的墙壁时会比较灵敏，而在其他方向上就变得不可靠了。因此，这种碰撞检测方式不适合所有的场合，如果希望实体与环境之间的碰撞检测足够真实，就得采用如下的解决方案了。

- 移动实体对象的位置。
- 求出实体对象的左上、左下、右下、右上 4 个角的坐标。
- 分别检测这 4 个点所对应的地图图元是否可以通过。只要有一个点检测到了不可通过的地图图元，则发生碰撞，进行碰撞处理。

说明 上述的碰撞检测算法不仅适合于地图由图元构成的场合，对于其他场合也是适用的。有些情况下是不必对 4 个角上的点都进行检测的（如向左上运动），所以该算法还可以进行优化。

一般情况下，实体对象与环境发生碰撞之后，实体对象的处理方式通常是退回到移动前的位置，这种算法并不十分完美。如实体对象当前的位置靠在墙边，运动方向为向左下方运动，那么其在移动单位距离时必然会与墙壁发生碰撞，如图 7-19 所示。



▲图 7-19　实体处理碰撞时产生的误差

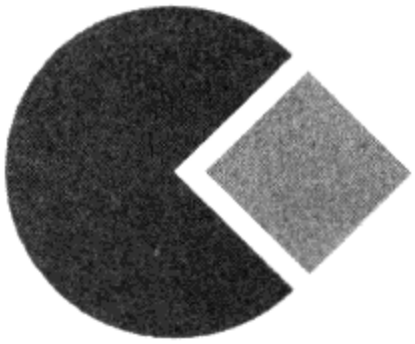
这时如果让实体退回到移动前的位置就显得太不合逻辑了，正确的处理方式是让实体沿着墙壁向下移动一段距离。要想实现这种处理方式，在碰撞检测时就必须将 x、y 方向的判断单独进行。其检测步骤如下所示。

- 根据指令先在 x 方向上移动指定的距离。
- 进行碰撞检测，如果发生碰撞，退回原位，x 方向上的移动无效。
- 根据指令在 y 方向上移动指定的距离。
- 进行碰撞检测，如果发生碰撞，退回原位，y 方向上的移动无效。

采用上述算法来处理图 7-19 中的情况，实体在检测到 x 方向上无法向左移动后，并不影响其在 y 方向上的移动，实体会沿着墙壁向下滑动，而不是留在原位。对于实体不紧靠墙壁但是仍然会在一个单位移动距离内与墙壁发生碰撞的情况，其处理方式与实体紧靠墙壁时比较类似，只是需要对 x 方向上的位置进行修正使其在本次移动过程结束后 x 方向上紧靠墙壁。

7.5 小结

本章主要介绍了在游戏开发中需要了解并掌握的数学和物理方面的知识，希望读者在以后的学习和工作中注意积累。同时不仅要明白这些数学、物理知识，还要懂得如何在开发过程中正确地使用这些知识使游戏效果更加逼真。



第 8 章　游戏中的人工智能

人工智能（Artificial Intelligence，AI），是让计算机有一定的智能，能够与人进行交互的技术。游戏中应用人工智能可增加游戏的可玩性，如果游戏中没有人工智能，就不会有象棋的人机对战，也不会有逃脱密室的怪物追捕英雄，更不会有网络游戏中玩家与怪兽的激烈战斗。

本章通过一些比较成熟的例子来介绍人工智能在 Android 平台中的应用。人工智能的开发过程中对数学和物理知识的运用是必不可少的。

提示　如果读者不能完全理解本章算法的原理，可以先运用。不过还是建议在理解算法原理的前提下再去应用，这样可以根据实际情况随时更改算法，以达到更好的效果。

8.1 让怪物聪明起来——Android 中的路径搜索

开发过游戏的读者都知道，让怪物动起来容易，可是想让怪物聪明地动起来就难了。本节将通过一个完整的演示程序，对游戏中能够让怪物聪明运动的各种算法进行详细介绍。

8.1.1　路径搜索示例基本框架的搭建

在正式介绍搜索算法之前，需要先将示例的框架搭建出来，这样在介绍各个搜索算法时才能够看到算法的运行效果。搭建基本界面的具体步骤如下。

（1）新建一个名为 Sample_8_1 的 Android 项目。

（2）将程序中用到的两张图片 source.png 与 target.png 放入到 res/ drawable-mdpi 目录下。

（3）在 src/wyf/ytl 下创建文件 GameView.java，然后开发其框架代码，具体如下所示。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 GameView.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.content.Context;                           //引入 Context 类
3    import android.graphics.Canvas;                           //引入 Canvas 类
4    import android.graphics.Color;                            //引入 Color 类
5    import android.util.AttributeSet;                         //引入 AttributeSet 类
6    import android.view.View;                                 //引入 View 类
7    public class GameView extends View{
8         private static final int VIEW_WIDTH = 320;           //此 View 的宽度
9         private static final int VIEW_HEIGHT = 340;          //此 View 的高度
10        Game game;
11        Spinner mySpinner;                                   //Spinner 的引用
```

```
12        TextView CDTextView;
13        int span = 13;
14        Bitmap source = BitmapFactory.decodeResource(getResources(), R.drawable.source);
15        Bitmap target = BitmapFactory.decodeResource(getResources(), R.drawable.target);
16        Paint paint = new Paint();
17        private Handler myHandler = new Handler(){            //用来更新 UI 线程的控件
18                public void handleMessage(Message msg) {
19                if(msg.what == 1) {                           //改变长度的 TextView 的值
20                  CDTextView.setText("路径长度: " + (Integer)msg.obj);
21              }
22          }
23        };
24        public GameView(Context context, AttributeSet attrs) {   //构造器
25              super(context, attrs);
26        }
27        protected void onDraw(Canvas canvas) {                   //重写绘制方法
28              try{
29                    onMyDraw(canvas);                            //调用自己的绘制方法
30              }catch(Exception e){}
31        }
32        protected void onMyDraw(Canvas canvas){                  //自己的绘制方法
33              super.onDraw(canvas);
34              canvas.drawColor(Color.GRAY);
35              ……//省略了一部分绘制代码，将在以后完善
36        }
37        protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec){
38                                                    //重写的方法，返回的是此 View 的大小
39            setMeasuredDimension(VIEW_WIDTH,VIEW_HEIGHT);
40        }
41    }
```

- 第 8～9 行定义了该 View 的宽度和高度，而第 24～26 行为 GameView 类的构造器，因为需要将该类配置到 XML 文件中，所以必须要重写该构造器。
- 第 27～31 行为重写的绘制方法，在 onDraw 方法中调用自己实现的绘制方法使结构更清晰。
- 第 32～36 行为自己实现的绘制方法。
- 第 37～40 行重写 View 的 onMeasure 方法，该方法返回期望的 View 大小。

（4）打开 res/layout 下的 main.xml 文件，将其代码替换成如下代码。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/res/layout 目录下的 main.xml。

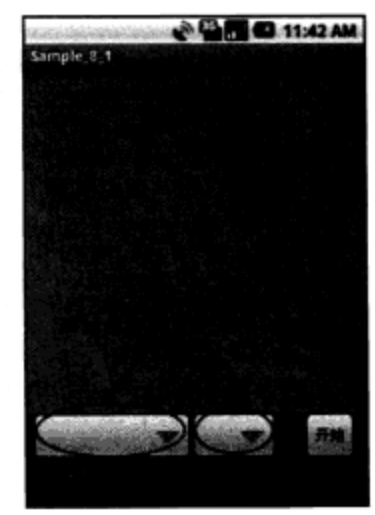
```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       android:orientation="vertical"
4       android:layout_width="fill_parent"
5       android:layout_height="fill_parent">         <!-- 定义一个线性布局 -->
6      <wyf.ytl.GameView
7         android:id="@+id/gameView"
8         android:layout_width="fill_parent"
9         android:layout_height="wrap_content"/>     <!-- 添加实现的 View -->
10     <AbsoluteLayout
11       xmlns:android="http://schemas.android.com/apk/res/android"
12       android:layout_width="fill_parent"
13       android:layout_height="fill_parent">         <!-- 添加一个坐标布局 -->
```

```
14          <Spinner
15          android:id="@+id/mySpinner"
16              android:layout_width="150px"
17              android:layout_height="wrap_content"
18              android:layout_x="5px"
19              android:layout_y="0px"/>          <!--在指定位置添加一个下拉列表 Spinner-->
20           <Spinner
21           android:id="@+id/target"
22              android:layout_width="wrap_content"
23              android:layout_height="wrap_content"
24              android:layout_x="155px"
25              android:layout_y="0px"/>          <!-- 在指定位置添加一个下拉列表 Spinner-->
26         <Button
27              android:id="@+id/go"
28              android:layout_width="wrap_content"
29              android:layout_height="wrap_content"
30              android:text="开始"
31              android:layout_x="260px"
32              android:layout_y="0px"/>          <!--在指定位置添加一个开始按钮-->
33         <TextView
34              android:id="@+id/bushu"
35              android:layout_width="wrap_content"
36              android:layout_height="wrap_content"
37              android:text="使用步数："
38              android:layout_x="20px"
39              android:layout_y="50px"/>         <!--在指定位置添加一个文本控件-->
40         <TextView
41              android:id="@+id/changdu"
42              android:layout_width="wrap_content"
43              android:layout_height="wrap_content"
44              android:text="路径长度："
45              android:layout_x="150px"
46              android:layout_y="50px"/>         <!--在指定位置添加一个文本控件-->
47       </AbsoluteLayout>
48    </LinearLayout>
```

- 第 2～5 行定义了一个垂直的线性布局。
- 第 6～9 行向线性布局中添加之前实现的 GameView，需要使用全称类名。
- 第 10～13 行向线性布局中添加一个坐标布局。
- 第 14～25 行在坐标布局的指定位置添加两个下拉列表框 Spinner。

第 26～32 行为再在坐标布局的指定位置添加一个开始按钮。同时在代码的第 33～46 行再向坐标布局添加两个文本控件。

运行该项目，得到的效果如图 8-1 所示。



▲图 8-1　程序框架效果图

8.1.2　路径搜索示例的控制面板实现

前面已经将该例子的基本框架搭建完成，但是还有一部分控件没有内容，接下来将继续对框架进行完善，具体步骤如下。

（1）开发地图类，该类中包含所有的地图信息。在 src/wyf/ytl 下新建 MapList.java 文件，开发其代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 MapList.java。

```
1    package wyf.ytl;                                    //声明包语句
```

```
2    public class MapList {                                          //该类为地图类
3      static int[] source={2,2};                                    //出发点坐标
4      static int[][] target={                                       //目标点坐标数组
5              {13,2},{4,22},{0,11},{9,10},{21,22}
6      };
7      static int[][][] map = new int[][][]                          //地图数组
8      {
9              {
10                   {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
11                   {1,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0},
12                   ……                                            //该处省略了数组中的大部分代码
13                   ……                                            //读者可自行查看光盘中此处代码
14                   {1,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0},
15                   {1,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0}
16             }
17      };
18   }
```

- 第 3 行定义了出发点的位置，位于整个地图的第 2 行第 2 列。
- 第 4～6 行定义了目标点的位置数组，这样可以添加多个目标点，运行时通过选择不同的目标点来测试同一种搜索算法在不同情况下的运行效果。
- 第 7～17 行为该例子的地图数组，是用三维数值表示的，以后如需添加新地图，只需在三维数组中再添加一个二维地图矩阵即可。

（2）创建算法类的框架，在 src/wyf/ytl 下创建 Game.java 文件，实现其框架代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 Game.java。

```
1    package wyf.ytl;                                                //声明包语句
2    import java.util.ArrayList;                                     //引入相关类
3    import java.util.HashMap;                                       //引入相关类
4    import java.util.LinkedList;                                    //引入相关类
5    import java.util.PriorityQueue;                                 //引入相关类
6    import java.util.Stack;                                         //引入相关类
7    import android.os.Handler;                                      //引入相关类
8    import android.os.Message;                                      //引入相关类
9    import android.widget.Button;                                   //引入相关类
10   import android.widget.TextView;                                 //引入相关类
11   public class Game {                                             //算法类
12        int algorithmId=0;                                         //算法代号
13        int mapId = 0;                                             //地图编号
14        int[][] map = MapList.map[mapId];
15        int[] source = MapList.source;                             //出发点
16        int[] target = MapList.target[0];                          //目标点
17        GameView gameView;                                         //gameView 的引用
18        Button goButton;                                           //goButton 的引用
19        TextView BSTextView;                                       //BSTextView 的引用
20        ArrayList<int[][]> searchProcess=new ArrayList<int[][]>();   //搜索过程
21        Stack<int[][]> stack=new Stack<int[][]>();                   //深度优先所用栈
22        HashMap<String,int[][]> hm=new HashMap<String,int[][]>();    //结果路径记录
23        LinkedList<int[][]> queue=new LinkedList<int[][]>();       //广度优先所用队列
24                                                                   //A*用优先级队列
25        PriorityQueue<int[][]> astarQueue=new PriorityQueue<int[][]>(100,new
          AstarComparator (this));
26                                                //记录到每个点的最短路径 for Dijkstra
27        HashMap<String,ArrayList<int[][]>> hmPath=new HashMap<String,
          ArrayList<int[][]>>();
28                                                //记录路径长度 for Dijkstra
29        int[][] length=new int[MapList.map[mapId].length][MapList.map[mapId][0].
```

```
        length];
30      int[][] visited=new int[MapList.map[0].length][MapList.map[0][0].length];
                                                    //0 未去过，1 去过
31      int[][] sequence={
32            {0,1},{0,-1},
33            {-1,0},{1,0},
34            {-1,1},{-1,-1},
35            {1,-1},{1,1}
36      };
37      boolean pathFlag=false;                          //true 找到了路径
38      int timeSpan=10;//时间间隔
39      private Handler myHandler = new Handler(){       //用来更新 UI 线程的控件
40            public void handleMessage(Message msg){
41            if(msg.what == 1){                         //改变按钮状态
42                  goButton.setEnabled(true);
43            }
44            else if(msg.what == 2){                    //改变步数的 TextView 的值
45                  BSTextView.setText("使用步数：" + (Integer)msg.obj);
46            }
47            }
48      };
49      public void clearState(){                        //清空所有状态与列表
50            pathFlag=false;
51            searchProcess.clear();                     //清空搜索过程
52            stack.clear();                             //清空深度优先所用的栈
53            queue.clear();                             //清空广度优先所用的队列
54            astarQueue.clear();                        //清空 A*用到的队列
55            hm.clear();                                //清空结果路径
56            visited=new int[MapList.map[mapId].length][MapList.map[mapId][0].length];
57            hmPath.clear();
58            for(int i=0;i<length.length;i++){
59                  for(int j=0;j<length[0].length;j++){
60                        length[i][j]=9999;       //初始路径长度为最大距离都不可能的那么大
61                  }
62            }
63      }
64      public void runAlgorithm(){                      //运行算法
65            clearState();
66            switch(algorithmId){
67                  //……在此省略 runAlgorithm 方法的部分代码，将在之后补全
68            }}}
```

- 第 14～16 行得到地图类中的地图、出发点以及目标点。
- 第 20～30 行初始化各个搜索算法所用到的栈和队列等。
- 第 31～36 行表示相对于当前点的 8 个方向的点。
- 第 39～48 行创建一个 Handler 类，根据发来的信息来更新不同的 UI 线程的控件。
- 第 49～63 行的方法是用来清空所有状态的，而第 64～68 行的方法是用来根据 algorithmId 值来调用不同算法的。

（3）开发 Activity 的代码，打开 src/wyf/ytl 下的 Sample_8_1.java 文件，将其代码替换成下列代码。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 Sample_8_1.java。

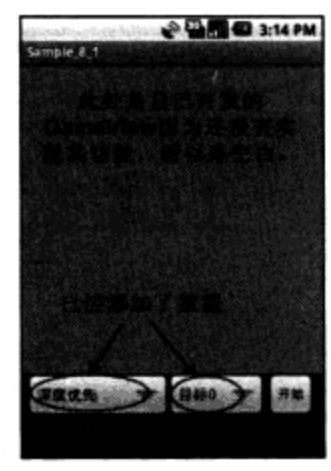
```
1     package wyf.ytl;                                    //声明包语句
2     import android.app.Activity;                        //引入相关类
3     import android.os.Bundle;                           //引入相关类
4     import android.view.View;                           //引入相关类
```

```
5    import android.widget.AdapterView;                                  //引入相关类
6    import android.widget.ArrayAdapter;                                 //引入相关类
7    import android.widget.Button;                                       //引入相关类
8    import android.widget.Spinner;                                      //引入相关类
9    import android.widget.TextView;                                     //引入相关类
10   public class Sample_8_1 extends Activity {
11        private static final String[] mySpinner_str = {                //搜索下拉列表的内容
12             "深度优先","广度优先","Dijkstra","广度优先 A*","Dijkstra A*"
13        };
14        Spinner mySpinner;                                             //搜索下拉列表框
15        Spinner targetSpinner;                                         //目标下拉列表框
16        Button goButton;                                               //开始按钮
17        GameView gameView;                                             //自己实现的地图 View
18        TextView BSTextView;                                           //使用步数的文本
19        TextView CDTextView;                                           //路径长度的文本
20        Game game;
21        private ArrayAdapter<String> adapter;                          //搜索下拉列表的模型
22        private ArrayAdapter<String> adapter2;                         //目标下拉列表的模型
23          public void onCreate(Bundle savedInstanceState) {  //重写 onCreate 回调方法
24             super.onCreate(savedInstanceState);
25             setContentView(R.layout.main);                      //设置当前显示的用户界面
26             mySpinner = (Spinner)findViewById(R.id.mySpinner);
                                                                   //得到搜索下拉列表的引用
27             targetSpinner = (Spinner)findViewById(R.id.target);
                                                                   //得到目标下拉列表的引用
28             gameView = (GameView) findViewById(R.id.gameView);
                                                                   //得到 GameView 的引用
29             BSTextView = (TextView)findViewById(R.id.bushu);
                                                                   //得到使用步数的文本控件的引用
30             CDTextView = (TextView)findViewById(R.id.changdu);
                                                                   //得到路径长度的文本控件的引用
31             goButton = (Button) findViewById(R.id.go); //得到开始按钮的引用
32             game = new Game();                                  //初始化算法类
33                                                                 //创建搜索下拉列表的模型
34             adapter = new ArrayAdapter<String>(this,android.R.layout.simple_spinner_
               item, mySpinner_str);
35             String[] target_str = new String[MapList.target.length];
                                                                   //根据目标点的个数创建一个数组
36             for(int i=0; i<MapList.target.length; i++){
37             target_str[i] = "目标"+i;
38             }
39                                                                 //创建目标下拉列表的模型
40           adapter2 = new ArrayAdapter<String>(this,android.R.layout.simple_spinner_
             item, target_str);
41             mySpinner.setAdapter(adapter);                      //设置模型
42             targetSpinner.setAdapter(adapter2);                 //设置模型
43             goButton.setOnClickListener(                        //按钮监听器
44             new Button.OnClickListener(){
45                       public void onClick(View v) {
46                            game.runAlgorithm();                 //调用运算方法
47                            goButton.setEnabled(false);
48                       }
49               }
50             );
51             targetSpinner.setOnItemSelectedListener(       //目标选择的下拉列表监听
52             new Spinner.OnItemSelectedListener(){
53                       public void onItemSelected(AdapterView<?> a, View v,int arg2,
                         long arg3){
54                            game.target = MapList.target[arg2];
55                            game.clearState();             //将 game 的状态清空
56                            gameView.postInvalidate();     //重写绘制 gameView
57                       }
```

```
58                          public void onNothingSelected(AdapterView<?> arg0){
59                          }
60              }
61          );
62          mySpinner.setOnItemSelectedListener(                    //算法选择的下拉列表监听
63              new Spinner.OnItemSelectedListener(){
64                          public void onItemSelected(AdapterView<?> ada, View v, int arg2,
                            long arg3){
65                                  game.clearState();              //将 game 的状态清空
66                                  game.algorithmId =  (int) ada.getSelectedItemId();
                                                                    //得到选择的算法 ID
67                                  gameView.postInvalidate();      //重写绘制 gameView
68                          }
69                          public void onNothingSelected(AdapterView<?> arg0) {
70                          }
71              }
72           );
73          this.initIoc();                                         //调用依赖注入方法
74      }
75      public void initIoc(){                                      //依赖注入
76              gameView.game = this.game;
77              gameView.mySpinner = this.mySpinner;   //为 gameView 中 mySpinner 变量赋值
78              gameView.CDTextView = this.CDTextView;
79              game.gameView = this.gameView;
80              game.goButton = this.goButton;         //为 game 中 goButton 变量赋值
81              game.BSTextView = this.BSTextView;
82      }}
```

- 第 11～13 行定义了搜索下拉列表的内容，第 14～19 行声明了各个控件的引用，第 23 行重写 onCreate 回调方法。
- 第 26～31 行得到各个控件的引用，第 35～38 行根据目标点的个数创建一个数组并为其赋值。
- 第 40 行创建下拉列表模型，两个参数分别为显示的风格 View 及数据模型。
- 第 43～50 行为开始按钮添加监听，当单击按钮时，根据所选择的算法调用算法运算，将按钮设置成不可用状态。
- 第 51～61 行为目标选择下拉列表添加监听，当选择了目标点时，将 game 的状态清空并重绘 gameView。
- 第 62～72 行为搜索下拉列表添加监听，当选择了算法时，清空 game 的状态，然后重绘 gameView。
- 第 75～82 行方法的目的是将各个类需要的引用传进去。

此时的运行效果如图 8-2 和图 8-3 所示。



▲图 8-2　下拉列表添加数据

▲图 8-3　单击下拉列表效果

8.1.3　路径搜索示例 GameView 的实现

前面已经将控制面板的界面以及基本功能开发完毕，接下来将对 GameView 类进行开发，使整个应用更加完善，该类的开发步骤如下。

（1）先打开 src/wyf/ytl 下的 GameView.java 文件，开发 onMyDraw 方法，将下列代码添加到 8.1.1 小节 GameView.java 代码的第 35 行。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 GameView.java。

```
1    protected void onMyDraw(Canvas canvas){                    //开发的绘制方法
2         super.onDraw(canvas);
3         canvas.drawColor(Color.GRAY);                         //背景色
4         paint.setColor(Color.BLACK);                          //设置颜色
5         paint.setStyle(Style.STROKE);                         //设置风格
6         canvas.drawRect(5, 5, 313, 327, paint);               //绘制矩形
7         int[][] map = game.map;
8         int row = map.length;
9         int col = map[0].length;
10        for(int i=0; i<row; i++){                             //绘制地图块
11             for(int j=0; j<col; j++){
12                  if(map[i][j] == 0){                          //白色
13                       paint.setColor(Color.WHITE);            //设置画笔颜色
14                       paint.setStyle(Style.FILL);             //设置画笔风格为填满
15        canvas.drawRect(6+j*(span+1), 6+i*(span+1), 6+j*(span+1)+span, 6+i*(span+1)+
          span, paint);
16                  }
17                  else if(map[i][j] == 1){                     //黑色
18                       paint.setColor(Color.BLACK);
19                       paint.setStyle(Style.FILL);             //设置代码风格
20        canvas.drawRect(6+j*(span+1), 6+i*(span+1), 6+j*(span+1)+span, 6+i*(span+1)+
          span, paint);
21                  }
22             }
23        }
24        ArrayList<int[][]> searchProcess=game.searchProcess;
25        for(int k=0;k<searchProcess.size();k++){              //绘制寻找过程
26             int[][] edge=searchProcess.get(k);
27             paint.setColor(Color.BLACK);                      //设置颜色
28             paint.setStrokeWidth(1);
29             canvas.drawLine(                                  //绘制直线
30                  edge[0][0]*(span+1)+span/2+6,edge[0][1]*(span+1)+span/2+6,
31                  edge[1][0]*(span+1)+span/2+6,edge[1][1]*(span+1)+span/2+6,
32                  paint
33             );
34        }
35        //绘制结果路径
36        if(
37             mySpinner.getSelectedItemId()==0||
38             mySpinner.getSelectedItemId()==1||
39             mySpinner.getSelectedItemId()==3
40        ){                                        //"深度优先","广度优先","广度优先 A*" 绘制
41             if(game.pathFlag){
42                  HashMap<String,int[][]> hm=game.hm;
43                  int[] temp=game.target;
44                  int count=0;                                 //路径长度计数器
45                  while(true){
46                       int[][] tempA=hm.get(temp[0]+":"+temp[1]);
47                       paint.setColor(Color.BLACK);            //设置颜色
48                       paint.setStyle(Style.STROKE);           //加粗
49                       paint.setStrokeWidth(2);                //设置画笔粗度为 2px
50                       canvas.drawLine(                        //画线
51                            tempA[0][0]*(span+1)+span/2+6,tempA[0][1]*(span+1)+
                              span/2+6,
52                            tempA[1][0]*(span+1)+span/2+6,tempA[1][1]*(span+1)+
                              span/2+6,
53                            paint
54                       );
55                       count++;                                //count 自加
56                       if(tempA[1][0]==game.source[0]&&tempA[1][1]==game.source[1]){
```

```
                                                        //有否到出发点
57                          break;
58                      }
59                      temp=tempA[1];
60                  }
61                  Message msg1 = myHandler.obtainMessage(1, count);
                                                        //改变 TextView 文字
62                  myHandler.sendMessage(msg1);        //发送 Handler 消息
63              }
64          }
65          else if(
66              mySpinner.getSelectedItemId()==3||      //选择的是 3 或 4
67              mySpinner.getSelectedItemId()==4
68          ){                                          //"Dijkstra"绘制
69              if(game.pathFlag){
70                  HashMap<String,ArrayList<int[][]>> hmPath=game.hmPath;
71                      ArrayList<int[][]> alPath=hmPath.get(game.target[0]+":"+game.
                        target[1]);
72                      for(int[][] tempA:alPath){
73                          paint.setColor(Color.BLACK);
74                          paint.setStyle(Style.STROKE);   //加粗
75                          paint.setStrokeWidth(2);        //设置画笔粗度为 2px
76                          canvas.drawLine(                //画线
77                              tempA[0][0]*(span+1)+span/2+6,tempA[0][1]*(span+1)+
                                span/2+6,
78                              tempA[1][0]*(span+1)+span/2+6,tempA[1][1]*(span+1)+
                                span/2+6,
79                              paint
80                          );
81                      }
82                      Message msg1 = myHandler.obtainMessage(1, alPath.size());
                                                        //改变 TextView 文字
83                      myHandler.sendMessage(msg1);    //发送 Handler 消息
84                  }
85          }
86          //绘制出发点
87          canvas.drawBitmap(source, 6+game.source[0]*(span+1), 6+game.source[1]*(span+1),
            paint);
88          //绘制目标点
89          canvas.drawBitmap(target, 6+game.target[0]*(span+1), 6+game.target[1]*(span+1),
            paint);
90      }
```

- 第 3 行绘制背景色。第 6 行绘制一个大矩形作为外框。
- 第 11～23 行对地图进行循环检测，当为 0 时绘制白色矩形，当为 1 时绘制黑色障碍物矩形。
- 第 24～34 行绘制搜索的过程。
- 第 36～64 行绘制的是深度优先、广度优先、广度优先 A*算法的搜索结果。第 65～85 行绘制的是 Dijkstra 算法的搜索过程。
- 第 86～89 行在指定位置绘制出发点及目标点。

（2）在 src/wyf/ytl 下创建 AStarComparator.java 类文件，该类为 A*算法所使用的比较器，其代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 AStarComparator. java。

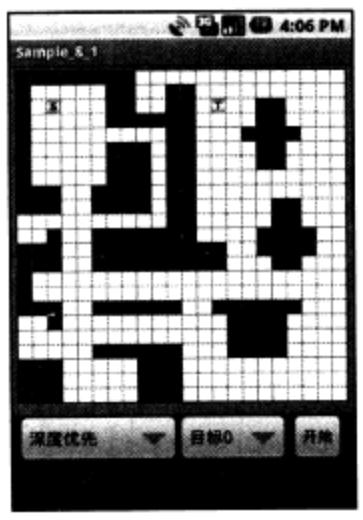
```
1   package wyf.ytl;
2   import java.util.*;                                 //引入相关包
3   public class AStarComparator implements Comparator<int[][]>{
```

```
                                                                    //实现了 Comparator 接口
4           Game game;
5           public AStarComparator(Game game){                      //构造器
6                this.game=game;
7           }
8           public int compare(int[][] o1,int[][] o2){              //比较方法
9                int[] t1=o1[1];
10               int[] t2=o2[1];
11               int[] target=game.target;                          //得到目标点
12                                                                  //直线物理距离
13               int a=(t1[0]-target[0])*(t1[0]-target[0])+(t1[1]-target[1])*(t1[1]-
                 target[1]);
14               int b=(t2[0]-target[0])*(t2[0]-target[0])+(t2[1]-target[1])*(t2[1]-
                      target[1]);
15                                                                  //门特卡罗距离
16               //int a=game.visited[o2[0][1]][o2[0][0]]+Math.abs(t1[0]-target[0])+
                 Math.abs(t1[1]- target[1]);
17               //int b=game.visited[o2[0][1]][o2[0][0]]+Math.abs(t2[0]-target[0])+
                 Math.abs(t2[1]- target[1]);
18               return a-b;                                        //返回差值
19          }
20          public boolean equals(Object obj){                      //重写的 equals 方法
21               return false;
22          }
23     }
```

- 第 3 行 AStarComparator 实现了 Comparator 接口。
- 第 12～14 行使用直线物理距离计算距离的值。
- 第 16～17 行使用门特卡罗距离计算距离的值。

提示 两点间距离的计算方法有很多种，此处给出了常用的两种，直线物理距离与门特卡罗距离，在程序中使用的是直线物理距离，所以将门特卡罗距离的相关代码注释掉。

（3）运行该例子得到的效果如图 8-4 所示。



▲图 8-4　界面完成效果

8.1.4　深度优先路径搜索 DFS

深度优先搜索 DFS 在搜索过程中不考虑各个边的开销，只考虑路径的选择。其思路是站在一个连通图的节点上，然后尽可能地沿着一条边深入，当遇到死胡同时进行回溯，然后继续搜索，直到找到目标为止。

下面将通过图 8-5 来介绍深度优先的搜索过程，首先出发点是节点 1，搜索过程如下。

（1）从节点 1 出发，按人为规定的顺序选择一条路径到达节点 2，如图 8-5 中 A 所示。

（2）在节点 2 处按之前的规定选择一条路径到达节点 3，如图 8-5 中 B 所示。

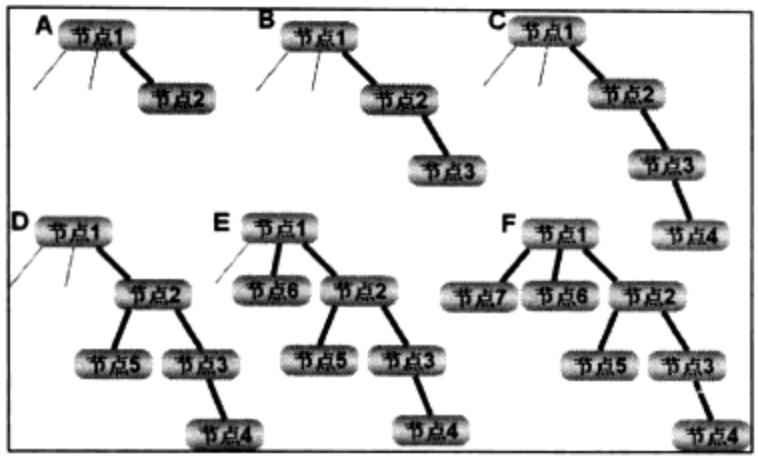
（3）在节点 3 处仍然继续深入搜索到达节点 4，如图 8-5 中 C 所示。

（4）当到达节点 4 时发现没有路可以走了。回溯到节点 3 查看，当发现节点 3 仍没有路可走时

继续回溯到节点 2，在节点 2 发现通往节点 5 的边且没有访问过，所以访问节点 5，如图 8-5 中 D 所示。

（5）而在节点 5 处无路可走回溯到节点 2，经过判断后再回溯到节点 1，此时发现节点 6 没有访问过，则访问节点 6，如图 8-5 中 E 所示。

（6）接着因节点 6 没有子节点，所以从节点 6 回溯到节点 1，之后发现节点 7 没有访问过，则访问节点 7，此时图中所有节点全部访问到了，搜索结束，如图 8-5 中 F 所示。



▲图 8-5　深度优先示例图

深度优先算法在程序中一般用栈来存储待访问的边，接下来运用之前开发的搜索示例框架实现 Android 平台中的深度搜索算法，实现步骤如下。

（1）打开 src/wyf/ytl 下的 Game.java 文件，为该类添加一个表示深度优先算法的方法，其代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 Game.java 的 DFS 方法。

```
1    public void DFS(){                                          //深度优先算法
2        new Thread(){                                           //创建一个线程
3            public void run(){
4                boolean flag=true;
5                int[][] start={                                 //开始状态
6                    {source[0],source[1]},
7                    {source[0],source[1]}
8                };
9                stack.push(start);
10               int count=0;                                    //步数计数器
11               while(flag){
12                   int[][] currentEdge=stack.pop();            //从栈顶取出边
13                   int[] tempTarget=currentEdge[1];            //取出此边的目的点
14                   //判断目的点是否去过，若去过则直接进入下次循环
15                   if(visited[tempTarget[1]][tempTarget[0]]==1){
16                       continue;
17                   }
18                   count++;
19                   visited[tempTarget[1]][tempTarget[0]]=1;//标识目的点为访问过
20                                                           //将临时目的点加入搜索过程中
21                   searchProcess.add(currentEdge);
22                                                           //记录此临时目的点的父节点
23               hm.put(tempTarget[0]+":"+tempTarget[1],new int[][]{currentEdge[1],
                 currentEdge [0]});
24                   gameView.postInvalidate();
25                   try{Thread.sleep(timeSpan);}catch(Exception  e){e.printStack
                     Trace();}
26                                                           //判断是否找到目的点
27                   if(tempTarget[0]==target[0]&&tempTarget[1]==target[1]){
28                       break;
29                   }
30                                                           //将所有可能的边入栈
31                   int currCol=tempTarget[0];
32                   int currRow=tempTarget[1];
33                   for(int[] rc:sequence){                 //对 sequence 进行循环
34                       int i=rc[1];
35                       int j=rc[0];
```

```
36                                  if(i==0&&j==0){continue;}  //都为 0 时跳出本次循环
37                                  if(currRow+i>=0&&currRow+i<MapList.map[mapId].length
                                    &&currCol+j> =0
38                                                  &&currCol+j<MapList.map[mapId][0].
                                                    length&&
39                                  map[currRow+i][currCol+j]!=1){      //当等于 1 时
40                                        int[][] tempEdge={
41                                              {tempTarget[0],tempTarget[1]},
42                                              {currCol+j,currRow+i}
43                                        };
44                                        stack.push(tempEdge);
45                                  }
46                            }
47                      }
48                      pathFlag=true;
49                      gameView.postInvalidate();              //重新绘制 gameView
50                      //设置按钮的可用性
51                      Message msg1 = myHandler.obtainMessage(1);
52                      myHandler.sendMessage(msg1);            //通过 Handler 设置按钮的可用性
53                      //改变 TextView 文字
54                      Message msg2 = myHandler.obtainMessage(2, count);
55                      myHandler.sendMessage(msg2);            //发送消息
56                }
57          }.start();
58    }
```

- 第 5～9 行在算法开始之前先将起始点到起始点作为第一条边入栈。
- 第 11～47 行为深度优先算法的主要代码，思路与之前介绍过的相同，通过对栈进行操作，逐渐将遇到的边入栈，然后从栈顶取出边查看是否去过，去过则跳出本次循环继续下一次循环，没去过的则访问该边，如此循环直到找到目标点为止。

提示　因为此算法运行可能需要很长时间，为了使用户界面能够正常响应，故将耗时的算法代码放到单独的线程中执行。

（2）在 src/wyf/ytl 下的 Game.java 文件的第 64～70 行的 runAlgorithm 方法中的 switch 中添加如下几行代码（见 8.2.1 小节 Game.java 代码的第 67 行）来调用深度优先算法。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 下的 Game.java 的 runAlgorithm 方法。

```
1          case 0:                                          //深度优先算法
2                DFS();                                     //调用深度优先算法计算
3          break;
```

说明　上述代码功能为调用刚开发的深度优先算法，计算从出发点到目标点的路径。

（3）运行该案例，选择目标 0，然后单击“开始”按钮，得到的运行效果如图 8-6 所示，粗线为搜索结果，细线为搜索过程。之后再选择目标 1，运行结果如图 8-7 所示。

8.1.5　广度优先路径搜索 BFS

广度优先搜索是在游戏中使用较多的一种搜索算法，其基本思路是站在一个节点上，先将所有连接到该节点的节点访问到，然后再继续访问下一层，直到找到目标为止。下面同样通过几个简单

的示意图来介绍广度优先搜索的过程，出发点为节点 1，步骤如下。

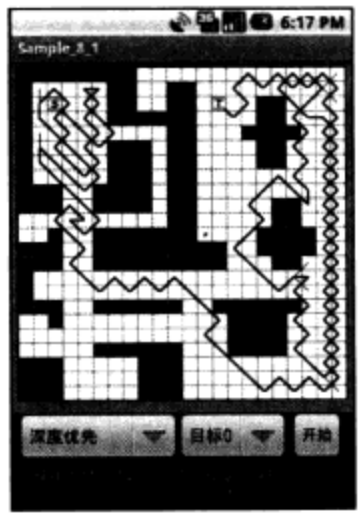
（1）站在节点 1 处，检测所有与节点 1 相连的边，按一定规则访问到节点 2，检测并记录所有与节点 2 相连的边，如图 8-8 中 A 所示。

（2）之后再访问节点 3，同样检测并记录所有与节点 3 相连的节点，如图 8-8 中 B 所示。

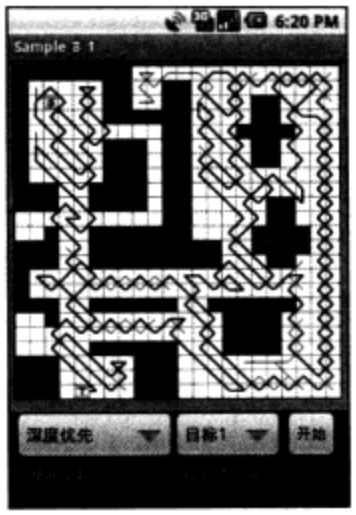
（3）再访问节点 4，如图 8-8 中 C 所示。

（4）然后访问节点 5，并记录与节点 5 相连的边，如图 8-8 中 D 所示。

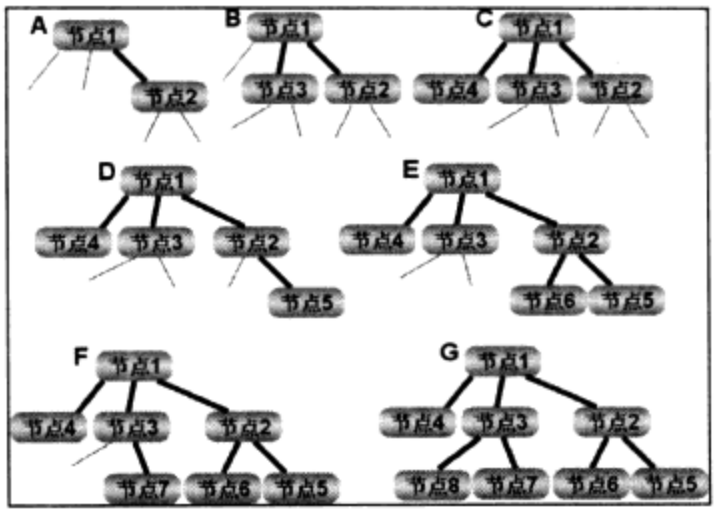
（5）同样原理依次再访问节点 6、节点 7、节点 8。如图 8-8 中 E、图 8-8 中 F 和图 8-8 中 G 所示。



▲图 8-6　搜索目标 0 的深度优先



▲图 8-7　搜索目标 1 的深度优先



▲图 8-8　广度优先算法搜索过程

广度优先在编程方面与深度优先基本相同，只是深度优先使用的是栈存储待访问的边，而广度优先使用的是队列。接下来继续对之前的例子进行完善，加上广度优先算法的实现，步骤如下。

（1）打开 src/wyf/ytl 下的 Game.java 类文件，在该类中添加一个名为 BFS 的方法，该方法将实现广度优先搜索算法，其代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 Game.java 的 BFS 方法。

```
1    public void BFS(){                                          //广度优先算法
2            new Thread(){
3                public void run(){
4                    int count=0;                                //步数计数器
5                    boolean flag=true;
6                    int[][] start={                             //开始状态
7                        {source[0],source[1]},
8                        {source[0],source[1]}
9                    };
10                   queue.offer(start);
11                   while(flag){
12                       int[][] currentEdge=queue.poll();       //从队首取出边
13                       int[] tempTarget=currentEdge[1];        //取出此边的目的点
14                       //判断目的点是否去过，若去过则直接进入下次循环
15                       if(visited[tempTarget[1]][tempTarget[0]]==1){
16                           continue;
17                       }
18                       count++;
19                       visited[tempTarget[1]][tempTarget[0]]=1;
                                                        //标识目的点为访问过
20                       searchProcess.add(currentEdge);
```

```
                                                    //将临时目的点加入搜索过程中
21                                                  //记录此临时目的点的父节点
22                              hm.put(tempTarget[0]+":"+tempTarget[1],
23                                        new int[][]{currentEdge[1],currentEdge[0]});
24                              gameView.postInvalidate();
25                              try{Thread.sleep(timeSpan);}catch(Exception e){e.print
                                StackTrace();}
26                                                  //判断是否找到目的点
27                              if(tempTarget[0]==target[0]&&tempTarget[1]==
                                target[1]){
28                                   break;
29                              }
30                                                  //将所有可能的边入队列
31                              int currCol=tempTarget[0];
32                              int currRow=tempTarget[1];
33                              for(int[] rc:sequence){     //对 sequence 进行循环
34                                   int i=rc[1];
35                                   int j=rc[0];
36                                   if(i==0&&j==0){continue;} //都为 0 时跳出本次循环
37                                   if(currRow+i>=0&&currRow+i<MapList.map[mapId].
                                     length
38                                             &&currCol+j>=0&&currCol+j<MapList.map
                                               [mapId][0]. length&&
39                                   map[currRow+i][currCol+j]!=1){ //都为 1 时
40                                        int[][] tempEdge={
41                                             {tempTarget[0],tempTarget[1]},
42                                             {currCol+j,currRow+i}
43                                        };
44                                        queue.offer(tempEdge);
45                                   }
46                              }
47                         }
48                         pathFlag=true;
49                         gameView.postInvalidate();                //重新绘制
50                         Message msg1 = myHandler.obtainMessage(1);
51                         myHandler.sendMessage(msg1);              //设置按钮的可用性
52                         Message msg2 = myHandler.obtainMessage(2, count);
53                         myHandler.sendMessage(msg2);              //改变 TextView 文字
54                    }
55               }.start();
56          }
```

- 第 5～9 行算法开始之前先将起始点到起始点当作第一条边入栈。
- 第 11～47 行为广度优先算法的主要代码，算法思路已经介绍过了，而操作与深度优先基本相同，只是此处不再使用栈来存储待访问的边，使用的是队列。

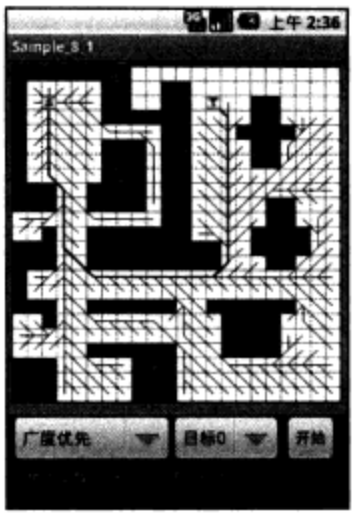
提示　同深度优先算法相同，此处也需要将耗时的算法代码放在单独的线程中执行。

（2）在 src/wyf/ytl 下的 Game.java 文件的第 64～70 行的 runAlgorithm 方法中的 switch 中添加如下几行代码（见 8.2.1 小节 Game.java 代码的第 67 行）。

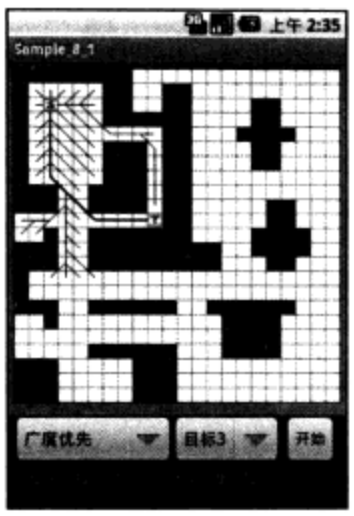
```
1    case 1:                                                    //广度优先算法
```

```
2            BFS();                                          //调用广度优先算法计算
3            break;
```

（3）再运行该示例，选择广度优先算法，单击“开始”按钮，得到的运行效果如图 8-9、图 8-10 所示，粗线为搜索结果，细线为搜索过程。读者可以再选择其他目标点比较搜索的过程。



▲图 8-9　广度优先搜索过程 1

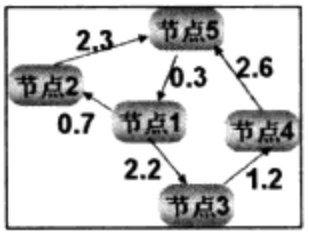


▲图 8-10　广度优先搜索过程 2

8.1.6　路径搜索算法——Dijkstra

Dijkstra 算法是典型的最短路径算法，一般用于求出从一点出发到达另一点的最优路径，但是，因为其遍历运算的节点较多，所以运行效率较低。

接下来将详细介绍 Dijkstra 路径搜索算法的执行思路，出发点（源节点）为节点 1，如图 8-11 所示。

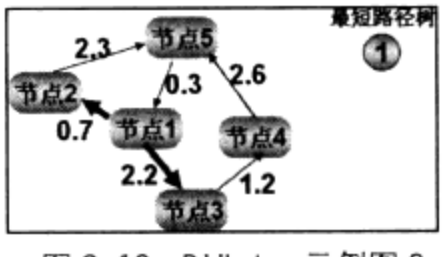


▲图 8-11　Dijkstra 示例图 1

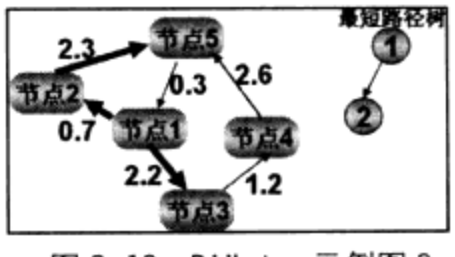
（1）首先将节点 1 加入到最短路径树中，并且将从它出发的所有边加到搜索边界中，如图 8-12 所示。

（2）然后查找所有在计算搜索边界中的各个边所指向的节点中到源节点距离最近的点（是节点 2），将节点 2 加入到最短路径树，然后将由节点 2 出发且到达点不在最短路径树上的边加入到搜索边界，如图 8-13 所示。

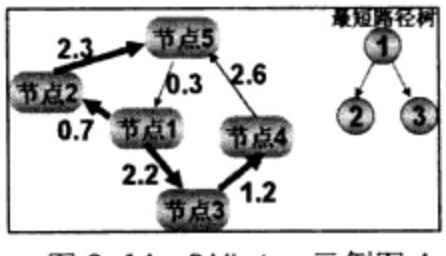
（3）再检查搜索边界上的边所指向的点，计算各点到源节点的距离，节点 5 距离是 3，节点 3 距离是 2.2，因此，将节点 3 加入到最短路径树，同时将节点 3 出发的边加入到搜索边界中，如图 8-14 所示。



▲图 8-12　Dijkstra 示例图 2



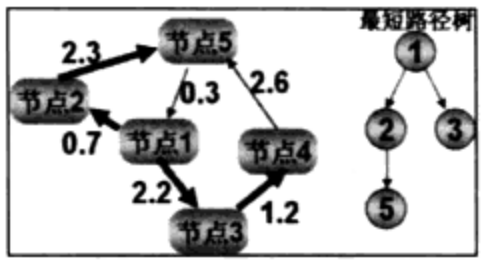
▲图 8-13　Dijkstra 示例图 3



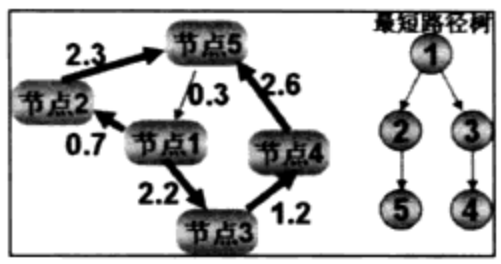
▲图 8-14　Dijkstra 示例图 4

（4）再检查搜索边界上的边所指向的点，节点 5 到源节点的距离是 3，节点 4 到源节点的距离是 3.4，因此，将节点 5 加入到最短路径树，而从节点 5 出发的边只有一条，但是，这条边指向的源节点为节点 1，所以不需要加入到搜索边界上，如图 8-15 所示。

（5）此时再检测搜索边界中的边所指向的点，发现只有节点 4 没有访问，且节点 4 到源节点的距离是 3.4，所以，将节点 4 加入到最短路径树，而从节点 4 出发的边只有一条，指向节点 5，而距源节点的距离为 6，比最短路径树中到节点 5 的距离大，所以不用继续考虑，如图 8-16 所示。



▲图 8-15　Dijkstra 示例图 5



▲图 8-16　Dijkstra 示例图 6

（6）此时已经访问到所有节点，搜索完成。

接下来，根据上面的原理，继续对之前的案例进行完善，添加一个表示 Dijkstra 算法的方法，操作步骤如下。

（1）打开 src/wyf/ytl 下的 Game.java 文件，为该类添加一个新方法，该方法即为 Dijkstra 搜索算法，其代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_1/src/wyf/ytl 目录下的 Game.java 的 Dijkstra 方法。

```
1    public void Dijkstra(){                                    //Dijkstra 算法
2            new Thread(){                                      //创建线程
3                public void run(){                             //重写 run 方法
4                    int count=0;                               //步数计数器
5                    boolean flag=true;                         //搜索循环控制
6                    int[] start={source[0],source[1]};         //开始点 col,row
7                    visited[source[1]][source[0]]=1;
8                    for(int[] rowcol:sequence){ //计算此点所有可以到达点路径及长度
9                        int trow=start[1]+rowcol[1];
10                       int tcol=start[0]+rowcol[0];    //行列
11                       if(trow<0||trow>18||tcol<0||tcol>18)continue;
12                       if(map[trow][tcol]!=0)continue;//不为 0 时跳出本次循环
13                                                       //记录路径长度
14                       length[trow][tcol]=1;
15                                                       //计算路径
16                       String key=tcol+":"+trow;
17                       ArrayList<int[][]> al=new ArrayList<int[][]>();
18                       al.add(new int[][]{{start[0],start[1]},{tcol,trow}});
19                       hmPath.put(key,al);
20                                                       //将去过的点记录
21                       searchProcess.add(new int[][]{{start[0],start[1]},
                         {tcol,trow}});
22                       count++;
23                   }
24                   gameView.postInvalidate();          //重新绘制
25                   outer:while(flag){
26                   //找到当前扩展点 K，要求扩展点 K 为从开始点到此点目前路径最短，且此点未考
                     //察过
27                       int[] k=new int[2];
28                       int minLen=9999;
29                       for(int i=0;i<visited.length;i++){  //循环计算距离
30                           for(int j=0;j<visited[0].length;j++){
31                               if(visited[i][j]==0){
32                                   if(minLen>length[i][j]){
33                                       minLen=length[i][j];
```

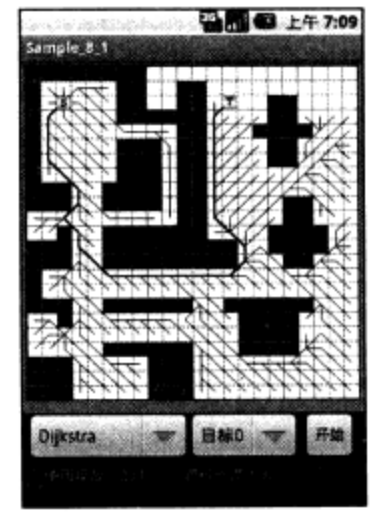
```
34                                                  k[0]=j;        //col
35                                                  k[1]=i;        //row
36                                              }
37                                          }
38                                      }
39                                  }
40                                  visited[k[1]][k[0]]=1;          //设置去过的点
41                                  gameView.postInvalidate();      //重绘
42                                  int dk=length[k[1]][k[0]];      //取出开始点到 K 的路径长度
43                                  ArrayList<int[][]> al=hmPath.get(k[0]+":"+k[1]);
                                                                    //取出开始点到 K 的路径
44                                  //循环计算所有 K 点能直接到的点到开始点的路径长度
45                                  for(int[] rowcol:sequence){
46                                      int trow=k[1]+rowcol[1];  //计算出新的要计算的点的坐标
47                                      int tcol=k[0]+rowcol[0];
48                                      //若要计算的点超出地图边界或地图上此位置为障碍物，则舍弃考
                                        //察此点
49  if(trow<0||trow>MapList.map[mapId].length-1||tcol<0||tcol>MapList.map[mapId][0].
    length-1) continue;
50                                      if(map[trow][tcol]!=0)continue;
51                                      int dj=length[trow][tcol];
                                                                //取出开始点到此点的路径长度
52                                      int dkPluskj=dk+1;     //计算经 K 点到此点的路径长度
53                                      //若经 K 点到此点的路径长度比原来的小，则修改到此点的路径
54                                      if(dj>dkPluskj){
55                                          String key=tcol+":"+trow;
56                                                                //克隆开始点到 K 的路径
57                                          ArrayList<int[][]> tempal=(ArrayList<int[]
                                            [][]>)al.clone();
58                                                          //将路径中加上一步从 K 到此点
59                                          tempal.add(new int[][]{{k[0],k[1]},{tcol,
                                            trow}});
60                                                          //将路径设置为从开始点到此点的路径
61                                          hmPath.put(key,tempal);
62                                                          //修改从开始点到此点的路径长度
63                                          length[trow][tcol]=dkPluskj;
64                                                          //若此点从未计算过路径长度则将此
                                                            //点加入考察过程记录
65                                          if(dj==9999){          //将去过的点记录
66                                              searchProcess.add(new int[][]{{k[0],
                                                k[1]}, {tcol, trow}});
67                                              count++;
68                                          }
69                                      }
70                                                                //看是否找到目的点
71                                      if(tcol==target[0]&&trow==target[1]){
72                                          pathFlag=true;
73                                          Message msg1 = myHandler.obtainMessage(1);
74                                          myHandler.sendMessage(msg1);//设置按钮的可用性
75                                          Message msg2 = myHandler.obtainMessage(2,
                                            count);
76                                          myHandler.sendMessage(msg2);
                                                                //改变 TextView 文字
77                                          break outer;
78                                      }
79                                  }
80                                  try{Thread.sleep(timeSpan);}catch(Exception e){e.print
                                    StackTrace();}
81                              }
82                          }
83                      }.start();                                //启动线程
84              }
```

- 第 8～23 行计算源节点到所有可以到达点路径及长度。
- 第 25 行开始检测找到当前扩展点，要求扩展点为从开始点到此点的路径是最短，且此点尚未考察过。
- 第 29～39 行计算点与点之间的距离。
- 第 44～69 行循环计算所有 K 点能直接到的点到开始点的路径的长度。
- 第 71～77 行查看到达是否是目的点，如果是将开始按钮设为可用，并将使用步数和路径显示在 TextView 上。

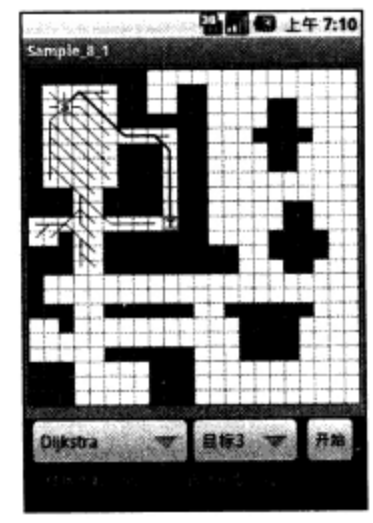
（2）在 src/wyf/ytl 下 Game.java 文件的第 64～70 行的 runAlgorithm 方法中的 switch 中添加如下几行代码（见 8.2.1 小节 Game.java 代码的第 67 行）。

```
1    case 3:                                          //Dijkstra 算法
2         Dijkstra();                                 //调用 Dijkstra 算法
3         break;
```

（3）再运行该示例，选择 Dijkstra 算法，然后单击“开始”按钮，可得到如图 8-17 和图 8-18 所示的运行效果，粗线为搜索结果，细线为搜索过程。读者可以再选择其他目标点比较搜索的过程。



▲图 8-17　Dijkstra 搜索过程 1



▲图 8-18　Dijkstra 搜索过程 2

8.1.7　用 A*算法优化搜索

A*算法是一种启发式搜索，所谓启发式搜索，就是利用一个启发因子评估每次寻找的路线的优劣，再决定往哪个节点走。实际上，A*只是一种思想，可以运用这种思想对之前实现的搜索算法进行优化。之前已经开发出启发因子 AStarComparator 类，为优先级队列设置比较器 AStarComparator 即可。

1. 用 A*算法优化广度优先算法

运用 A*算法对广度优先算法优化很简单，只需将广度优先中使用的广度优先队列改为 A*优先级队列即可，步骤如下。

（1）之前已经将 A*优先级队列创建，下面对其进行简单的介绍，其代码如下。

```
PriorityQueue<int[][]>  astarQueue=new  PriorityQueue<int[][]>
(100,new AStarComparator (this));
```

该句为 Game 类的成员变量，创建优先级队列并将创建的比较器 AStarComparator 传入。

（2）打开 src/wyf/ytl 下的 Game.java 文件，再为该类添加一个名为 BFSAStar 的方法。

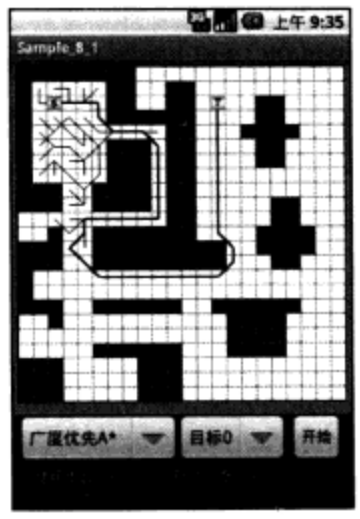
（3）将之前开发的 BFS 方法中的代码复制到 BFSAStar 方法内，然后将其中使用到的队列全部替换成 astarQueue 即可，读者可以查看光盘中的该方法程序。

（4）还需在 src/wyf/ytl 下的 Game.java 文件的第 64～70 行的 runAlgorithm 方法中的 switch 中

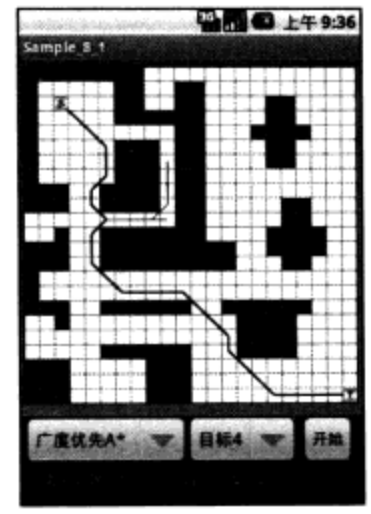
添加如下几行代码（见 8.2.1 小节 Game.java 代码的第 67 行）。

```
1    case 2:                                          //广度优先 A*算法
2         BFSAStar();                                 //调用广度优先 A*算法
3         break;
```

（5）再运行该程序选择广度优先 A*算法，得到的运行效果如图 8-19 和图 8-20 所示。



▲图 8-19　广度优先 A*效果 1



▲图 8-20　广度优先 A*效果 2

提示　用 A*优化的广度优先搜索算法得到的路径基本上是最优路径，但是并不能保证一定是最优路径，但执行效率较高，所以在游戏中应用较多。

2. 用 A*算法优化 Dijkstra 算法

前面已经对广度优先算法进行了 A*优化，接下来将继续对 Dijkstra 算法进行优化，具体步骤如下。

（1）打开 src/wyf/ytl 下的 Game.java 文件，再为该类添加一个名为 DijkstraAStar 的方法。

（2）将 8.1.6 小节开发的 Dijkstra 方法中的内容复制到 DijkstraAStar 中。

（3）将 8.1.6 小节代码中的第 32～36 行代码替换成下列代码。

```
1    if(length[i][j]!=9999){                                  //当值不为 9999 时
2         if(iniFlag){                                        //第一个找到的可能点
3                   minLen=length[i][j]+
4                        (int)Math.sqrt((j-target[0])*(j-target[0])+(i-target[1])*
                         (i-target[1]));
5                   k[0]=j;                                   //col
6                   k[1]=i;                                   //row
7                   iniFlag=!iniFlag;
8         }
9         else{                                               //不是第一个找到的可能点
10             int tempLen=length[i][j]+
11                  (int)Math.sqrt((j-target[0])*(j-target[0])+(i-target[1])*
                    (i-target[1]));
12             if(minLen>tempLen){                             //当 minLen 大于 tempLen 时
13                  minLen=tempLen;
14                  k[0]=j;                                   //col
```

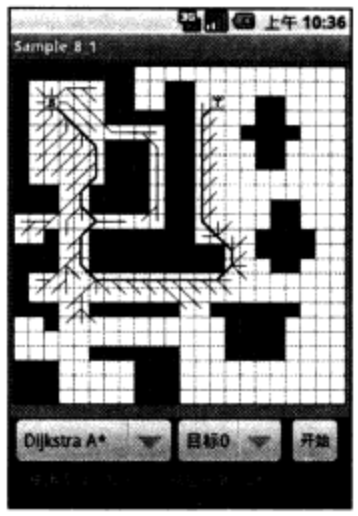
```
15                      k[1]=i;                                    //row
16                  }
17              }
18      }
```

（4）还需在 src/wyf/ytl 下的 Game.java 文件的第 64～70 行的 runAlgorithm 方法中的 switch 下添加如下几行代码（见 8.2.1 小节 Game.java 代码的第 67 行）。

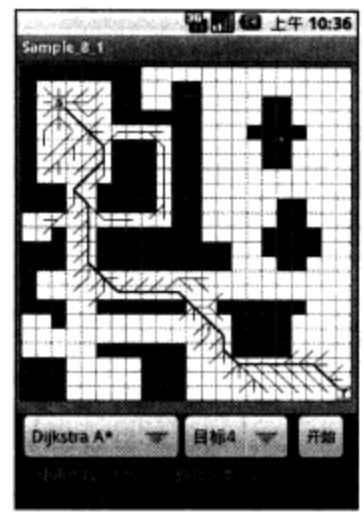
```
1     case 4:                                          //Dijkstra A*算法
2          DijkstraAStar ();                           //调用 DijkstraA*算法
3          break;
```

（5）再运行该程序，选择 Dijkstra A*算法，单击“开始”按钮后得到的运行效果如图 8-21 和图 8-22 所示。

提示　用 A*优化的 Dijkstra 搜索算法得到的路径一定是最优路径，在真正的游戏开发中，应用该算法的次数比较多。



▲图 8-21　Dijkstra A*效果图 1



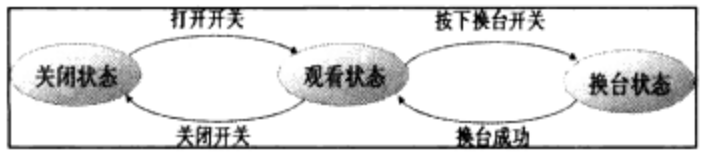
▲图 8-22　Dijkstra A*效果图 2

8.2 有限状态机

前一节已经介绍了人工智能中路径搜索的各种算法，本节将对人工智能的另一方面的状态机进行简单介绍，并通过几个案例介绍有限状态机在 Android 中是如何实现的。

8.2.1　何为有限状态机

有限状态机就是一个具有有限数量状态，并且能够根据相应的操作从一个状态变换到另一个状态，而在同一时刻只能处在一种状态下的智能体。



▲图 8-23　电视机的简单状态迁移

下面通过一个电视机的状态迁移来介绍什么是有限状态机，如图 8-23 所示。

例如，读者在家学习累了想看电视时，会先打开电视开关才能观看，那么，这时电视的状态会因为打开开

关的操作从关闭状态变换成观看状态。

当想看其他节目时又会按下换台开关，电视的状态会从观看状态变换成换台状态（黑屏一瞬间），而当换台成功时，又会从换台状态变换成观看状态。此时读者就会想了，这变来变去的真讨厌索性不看了，那么就需要按下关闭开关，将电视机的状态从观看状态切换到关闭状态。

这不断切换状态的电视就是一个有限状态机，当然，一个游戏中的智能体的行为往往要比电视换台复杂得多，但所体现的有限状态机原理是一样的。

提示　当然，现在电视的实际状态要比这个多，这里只是为了解释说明什么是有限状态机，从而进行了简化。

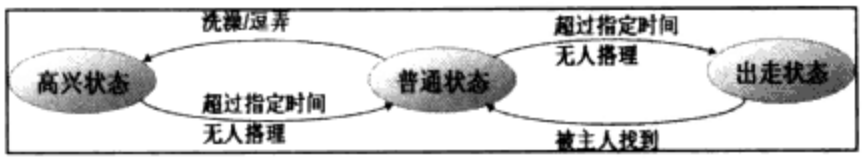
在游戏的开发过程中有很多地方会应用有限状态机，下面是一些应用了有限状态机的游戏例子。

- 魔兽世界中的 NPC 就应用了有限状态机，它会根据不同的情况具有移动到位、巡逻、沿路前进等状态。
- QQ 宠物也是一个典型的有限状态机的例子，很久没有喂食它会饥饿，很久没有理它他会心情不好，而如果饲养不好它还会生病。
- 雷电中的导弹也可以理解成是一个有限状态机，具有移动、碰撞、消失等状态。
- 冒险类游戏中的怪物也应用了有限状态机，当与玩家战斗时，当玩家比较强时怪物会选择逃跑，而当玩家比较弱时，怪物会选择攻击。

8.2.2　有限状态机的简单实现

首先看一个有限状态机简单实现的案例，该案例是一个养成游戏简化版，完成后运行效果如图 8-24 所示。游戏中的宠物有 3 种状态，分别为高兴、普通以及出走，玩家可通过按钮完成相关操作，宠物会根据玩家相关操作与自身的状态综合情况来改变状态，当长时间无人搭理宠物，宠物会自动切换到出走状态，出走状态下只能对其执行寻找操作，该案例的有限状态机原理如图 8-25 所示。

▲图 8-24　宠物饲养程序



▲图 8-25　养成游戏宠物的状态迁移

提示　实际游戏中宠物的状态会比这个多得多，在这里只是为了介绍有限状态机的应用，所以进行了简化。

该例子的开发步骤如下。

（1）新建一个名为 Sample_8_2 的 Android 项目。

（2）将所需要的 3 张表情图片放到 res/drawable-mdpi 目录下。

（3）打开 res/layout 下的 main.xml 文件，将其代码替换成如下代码。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_2/res/layout 下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                           <!-- 添加一个线性布局 -->
7        <ImageView
8                android:id="@+id/myImageView"
9                android:layout_width="wrap_content"
10               android:layout_height="wrap_content"
11               android:src="@drawable/common"
12               android:layout_gravity="center"
13       />                                          <!-- 添加一个 ImageView 控件 -->
14       <TextView
15               android:id="@+id/myTextView"
16               android:layout_width="wrap_content"
17               android:layout_height="wrap_content"
18               android:layout_gravity="center"
19               android:textSize="20px"
20               android:text="状态：普通"
21       />                                          <!-- 添加一个 TextView 文本控件 -->
22       <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
23               android:orientation="horizontal"
24               android:layout_width="fill_parent"
25               android:layout_height="fill_parent"
26               android:layout_gravity="center"
27       >                                           <!--添加一个水平的线性布局 -->
28               <Button
29                    android:id="@+id/bath"
30                    android:layout_width="106px"
31                    android:layout_height="wrap_content"
32                    android:layout_gravity="bottom"
33                    android:textSize="18px"
34                    android:text="洗澡"
35               />                                  <!-- 添加洗澡按钮-->
36               <Button
37                    android:id="@+id/engage"
38                    android:layout_width="106px"
39                    android:layout_height="wrap_content"
40                    android:text="逗弄"
41                    android:textSize="18px"
42                    android:layout_gravity="bottom"
43               />                                  <!-- 添加逗弄按钮-->
44               <Button
45                    android:id="@+id/find"
46                    android:layout_width="106px"
47                    android:layout_height="wrap_content"
48                    android:text="寻找"
49                    android:textSize="18px"
50                    android:layout_gravity="bottom"
51               />                                  <!-- 添加寻找按钮-->
52       </LinearLayout>
```

```
53    </LinearLayout>
```

- 第 2～6 行添加一个垂直的线性布局，第 7～13 行在线性布局中添加一个 ImageView 控件，并设置其对齐方式为居中。
- 第 14～21 行添加一个文本控件，设置对齐方式居中，字体大小为 20px。
- 第 22～27 行再添加一个水平的线性布局，对齐方式为居中。
- 第 28～51 行依次添加 3 个按钮控件，并分别设置了对齐方式及按钮上文字的大小等属性。

（4）定义两个表示状态和表示主人动作的枚举类型，定义方法是在 src/wyf/ytl 下的 Sample_8_2.java 文件中定义两个枚举类，即在 Sample_8_2 的最后加上下列代码。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_2/src/wyf/ytl 下的 Sample_8_2.java。

```
1     enum DogState {                                  //表示状态的枚举类型
2          HAPPY_STATE,                                //高兴状态
3          COMMON_STATE,                               //普通状态
4          AWAY_STATE;                                 //出走状态
5     }
6     enum MasterAction {                              //表示主人动作的枚举
7          BATH_ACTION,                                //洗澡
8          ENGAGE_ACTION,                              //逗弄
9          FIND_ACTION,                                //寻找
10         ALONE_ACTION;                               //无人搭理
11    }
```

提示　在正常的游戏开发中，应该将枚举类型单独放在一个文件中，此处考虑到演示的简单，将其放在了 Activity 的文件中。

（5）在 src/wyf/ytl 下创建 GameDog 类，并实现其代码如　下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_2/src/wyf/ytl 下的 GameDog.java。

```
1     package wyf.ytl;                                 //声明包语句
2     public class GameDog {
3          Sample_8_2 activity = null;                 //activity 的引用
4          private DogState currentState=DogState.COMMON_STATE;
                                                       //设置宠物，当前初始状态为普通状态
5          public GameDog(Sample_8_2 activity){            //构造器
6               this.activity = activity;
7          }
8          public boolean updateState(MasterAction ma){    //接收条件，更新状态的方法
9               boolean result=true;
10              switch(currentState){
11                   case HAPPY_STATE:                     //当前为高兴状态
12                      switch(ma){
13                        case ALONE_ACTION:               //超过指定时间无人搭理
14                          currentState=DogState.COMMON_STATE;    //切换状态
15                          activity.myImageView.setImageResource(R.drawable.common);
                                                           //换图
16                          activity.myTextView.setText("状态：普通");
17                        break;
18                        default:
19                          result=false;                  //返回 false 表示状态切换出错
20                      }
21                   break;
22                   case COMMON_STATE:                    //当前为普通状态
```

```
23                      switch(ma){
24                        case ALONE_ACTION:                    //超过指定时间无人搭理
25                          currentState=DogState.AWAY_STATE; //切换状态
26                          activity.myImageView.setImageResource(R.drawable.away);
                                                              //换图
27                          activity.myTextView.setText("状态：出走");
28                        break;
29                        case BATH_ACTION:                     //洗澡
30                        case ENGAGE_ACTION:                   //逗弄
31                          currentState=DogState.HAPPY_STATE;//切换状态
32                          activity.myImageView.setImageResource(R.drawable.happy);
                                                              //换图
33                          activity.myTextView.setText("状态：高兴");
34                        break;
35                        default:
36                          result=false;                       //返回 false 表示状态切换出错
37                      }
38                  break;
39                  case AWAY_STATE:                            //当前为出走状态
40                      switch(ma){
41                        case FIND_ACTION:                     //寻找
42                          currentState=DogState.COMMON_STATE;    //切换状态
43                          activity.myImageView.setImageResource(R.drawable.common);
                                                              //换图
44                          activity.myTextView.setText("状态：普通");
45                        break;
46                        default:
47                          result=false;                       //返回 false 表示状态切换出错
48                      }
49                  break;
50              }
51              return result;                                  //返回 true 表示状态切换成功
52          }
53      }
```

- 第 4 行设置宠物的当前初始状态为普通状态。
- 第 8 行为接收玩家的动作，更新状态的方法开始运行。
- 第 10～20 行当前状态为高兴状态时，只能接收无人搭理动作，而当接收到无人搭理动作后，切换宠物状态到普通，然后改变图片和文字的内容。

（6）再开发 Activity 的代码，打开 src/wyf/ytl 下的 Sample_8_2.java 文件，将 Activity 类的代码替换成如下的代码，但之前添加的枚举类不需替换。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_2/src/wyf/ytl 下的 Sample_8_2.java。

```
1     package wyf.ytl;                                          //声明包语句
2     import android.app.Activity;                              //引入相关类
3     import android.os.Bundle;                                 //引入相关类
4     import android.os.Handler;                                //引入相关类
5     import android.os.Message;                                //引入相关类
6     import android.view.View;                                 //引入相关类
7     import android.view.View.OnClickListener;                 //引入相关类
8     import android.widget.Button;                             //引入相关类
9     import android.widget.ImageView;                          //引入相关类
10    import android.widget.TextView;                           //引入相关类
11    public class Sample_8_2 extends Activity implements OnClickListener{
12          ImageView myImageView = null;                       //ImageView 控件的引用
13          TextView myTextView = null;                         //TextView 控件的引用
14          Button bath = null;                                 //洗澡按钮
```

```
15        Button engage = null;                                    //逗弄按钮
16        Button find = null;                                      //寻找按钮
17        GameDog gameDog = null;                                  //GameDog 的引用
18        //ActionThread actionThread = null;                      //后台线程的引用
19        Handler myHandler = new Handler(){                       //用来更新 UI 线程中的控件
20             public void handleMessage(Message msg) {
21                  switch(msg.what){
22                  case 1:                                         //为后台线程发来的消息
23                       gameDog.updateState(MasterAction.ALONE_ACTION);
                                                                    //长时间无人搭理
24                       break;
25                  }
26             }
27     };
28      public void onCreate(Bundle savedInstanceState) {          //重写的 onCreate 方法
29          super.onCreate(savedInstanceState);
30          setContentView(R.layout.main);
31          myImageView = (ImageView) this.findViewById(R.id.myImageView);
                                                                    //得到 myImageView 的引用
32          myTextView = (TextView) this.findViewById(R.id.myTextView);
                                                                    //得到 myTextView 的引用
33          bath = (Button) this.findViewById(R.id.bath);           //得到 bath 的引用
34          engage = (Button) this.findViewById(R.id.engage);       //得到 engage 的引用
35          find = (Button) this.findViewById(R.id.find);           //得到 find 的引用
36          bath.setOnClickListener(this);                          //添加监听
37          engage.setOnClickListener(this);                        //添加监听
38          find.setOnClickListener(this);                          //添加监听
39          gameDog = new GameDog(this);
40          //actionThread = new ActionThread(this);
41          //actionThread.start();                                 //启动后台线程
42      }
43        public void onClick(View v) {                             //实现监听器中的方法
44          if(v == bath){                                          //按下的是洗澡按钮
45              gameDog.updateState(MasterAction.BATH_ACTION);
46          }
47          else if(v == engage){                                   //按下的是逗弄按钮
48              gameDog.updateState(MasterAction.ENGAGE_ACTION);
49          }
50          else if(v == find){                                     //按下的是查找按钮
51              gameDog.updateState(MasterAction.FIND_ACTION);
52          }}}
```

- 第 12～18 行声明了各个对象的引用。第 19～27 行定义一个 Handler，用来根据接收到的后台线程发来的消息，来更改前台宠物的状态值。
- 第 28～42 行为重写的 onCreate 方法，设置当前用户界面，然后得到各个控件的引用并为按钮控件添加监听器。
- 第 10～41 行因为后台线程 ActionThread 类此时还没有开发，所以先将其注释掉。
- 第 43～52 行实现了监听接口中的 onClick 方法，根据出发的按钮不同执行不同的动作。

（7）接下来最后一步，就是开发后台线程 ActionThread.java 类，后台线程的作用是每隔一段时间对宠物的状态更新一次，例如，在高兴状态长时间无人搭理时，会变成普通状态，而在普通状态又长时间无人搭理时则会变成出走状态，后台线程类的代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_2/src/wyf/ytl 下的 ActionThread.java。

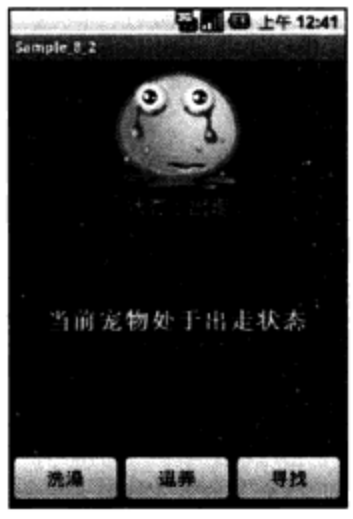
```
1     package wyf.ytl;                                              //声明包语句
2     public class ActionThread extends Thread{
```

```
3          private int sleepSpan = 5000;                          //睡眠的毫秒数
4          boolean flag = true;                                   //循环标记位
5          Sample_8_3 activity;                                   //activity 的引用
6          public ActionThread(Sample_8_3 activity){              //构造器
7               this.activity = activity;
8          }
9          public void run(){                                     //重写的 run 方法
10              while(flag){                                      //循环
11                   try{
12                        Thread.sleep(sleepSpan);                 //睡眠指定毫秒数
13                   }
14                   catch(Exception e ){
15                        e.printStackTrace();                     //打印异常信息
16                   }
17                   activity.myHandler.sendEmptyMessage(1);
                                                            //想 activity 发生 handler 消息
18              }
19         }
20    }
```

说明 该类非常简单，只是定时向 activity 的 myHandler 发送类型为 1 的消息即可。

（8）最后将第 6 步的代码中的第 18、40、41 行注释取消。

到此，整个例子已经开发完成，运行该案例，得到的运行效果如图 8-26 和图 8-27 所示。



▲图 8-26　出走状态下的宠物

▲图 8-27　普通状态下的宠物

说明 这种实现方式的优点是状态转移很简单时开发简单，但缺点也很明显，所有状态转移的管理都在一个方法中，若状态很多转移复杂则维护困难。

8.2.3　有限状态机的 OO 实现

前一节已将对有限状态机的简单实现进行了介绍，读者可能已经发现，宠物的状态越多，开发和维护的难度也随之越大。想象一下，如果宠物的状态有几十种，那么 GameDog 类的 updateState 方法就会有几十种分支，而在真实的游戏中，每次状态的改变还会再伴随着一系列的操作，那么 updateState 方法将会长得难以为继，这就需要使用新的设计模式来进行开发。

这种新的有限状态机是通过 OO（Object Oriented）来实现的，在面向对象的设计模式中，将宠物的状态封装成单个类，状态的切换由状态类来决定，这样结构更清晰，也更好管理。

下面通过一个案例来说明如何采用 OO 思想实现有限状态机，因本书篇幅有限，本小节只给出了与前一节案例的不同之处，如有需要，读者可自行查阅随书光盘中的源代码，案例的开发步骤如下。

（1）首先需要新建一个名为 Sample_8_3 的 Android 项目。

（2）接着将案例中用到的的 3 张表情图片放到 res/drawable-mdpi 目录下。

（3）编写 main.xml 布局文件，其代码与前一节的完全相同。

（4）打开 Sample_8_3.java 文件，在其最后加入动作的枚举类型以及各个状态类，代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_3/src/wyf/ytl 下的 Sample_8_3.java。

```
1    enum MasterAction {                                          //表示主人动作的枚举
2         BATH_ACTION,                                            //洗澡
3         ENGAGE_ACTION,                                          //逗弄
4         FIND_ACTION,                                            //寻找
5         ALONE_ACTION;                                           //无人搭理
6    }
7    abstract class State {                                       //所有状态类型的基类
8         public Sample_8_3 activity;
9         public State(Sample_8_3 activity){                      //构造器
10             this.activity = activity;
11        }
12        public abstract State toNextState(MasterAction ma);
13   }
14   class HappyState extends State{                              //表示高兴状态的类
15        ……  //省略了该类的全部代码。其实现与 CommonState 的实现方式相同，读者可自行开发
16             //或者查看随书光盘中的代码。
17   }
18   class CommonState extends State{                             //表示普通状态的类
19        public CommonState(Sample_8_3 activity) {               //构造器
20             super(activity);
21        }
22        public State toNextState(MasterAction ma){              //进入下一状态
23             State result=this;
24             switch(ma){
25             case ALONE_ACTION:                                  //超过指定时间无人搭理
26                result= new AwayState(activity);
27                activity.myImageView.setImageResource(R.drawable.away);      //换图
28                activity.myTextView.setText("状态：出走");       //设置文本内容
29             break;
30             case BATH_ACTION:                                   //洗澡
31             case ENGAGE_ACTION:                                 //逗弄
32                  result= new HappyState(activity);
33                  activity.myImageView.setImageResource(R.drawable.happy);   //换图
34                  activity.myTextView.setText("状态：高兴");      //设置文本内容
35                  break;
36             }
37             return result;
38        }
39   }
40   class AwayState extends State {                              //表示出走状态的类
41        ……  //省略了该类的全部代码。其实现与 CommonState 的实现方式相同，读者可自行开发
42             //或者查看随书光盘中的代码。
43   }
```

- 第 1～6 行与前一节的相同，定义表示主人动作的枚举类型。
- 第 7～13 行定义所有状态类型的基类。
- 第 18～39 行定义表示宠物普通状态的类，其中只有一个方法 toNextState，根据得到的动作

创建新状态并做换图等相关操作。

（5）在 src/wyf/ytl 下创建 GameDog 类，并实现其代码如下。

代码位置：见随书光盘中源代码/第 8 章/Sample_8_3/src/wyf/ytl 下的 GameDog.java。

```
1    package wyf.ytl;                                          //声明包语句
2    public class GameDog {
3         Sample_8_3 activity = null;                          //activity 的引用
4         private State currentState;                          //宠物的初始当前状态
5         public GameDog(Sample_8_3 activity){                 //构造器
6              this.activity = activity;
7              currentState=new CommonState(activity);         //设置宠物的初始当前状态为普通状态
8         }
9         public boolean updateState(MasterAction ma){         //接收条件，更新状态的方法
10             State beforeState=currentState;
11             currentState=currentState.toNextState(ma);
12             return !(beforeState==currentState);            //返回 true 表示状态切换成功
13        }
14   }
```

- 第 7 行初始化宠物的当前状态为普通状态。
- 第 9～13 行一样是接收动作改变状态，只是改变状态的操作不在此处出来，而是全部交给当前状态对象自身。

（6）之后的 3 步与前一节案例的最后 3 步完全相同，读者可参考前一节案例的步骤来完成开发。

到此，有限状态机的 OO 实现案例也已经开发完成，虽然运行效果与简单实现没有任何区别，但是后台所应用的设计模式是完全不同的。

提示　这种实现方式的好处是，游戏中的智能体不需要管理状态转移，状态的转移由状态对象自己管理，增加新状态也比较简单。

8.3 小结

本章中的第一部分介绍了各种搜索算法在 Android 平台中的实现，第二部分介绍了有限状态机的相关知识。在真实的游戏开发中，人工智能的应用远远不止这些，本书因篇幅有限，只能对 AI 的基本知识进行介绍，有兴趣的读者可以自行查阅人工智能方面的相关书籍进行更深层次的学习。

第 9 章　游戏开发小秘技

本章主要介绍游戏开发的一些技巧，包括地图编辑器、模糊逻辑以及一些游戏的优化技巧等。这些问题对于游戏开发者来说非常重要，但是初学者往往会忽略这些。

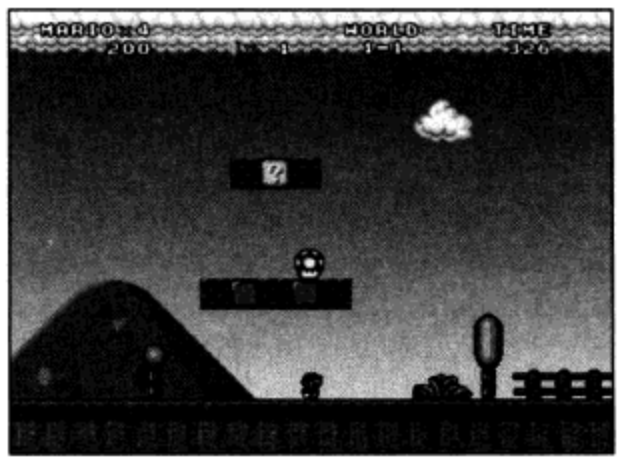
9.1 地图编辑器与关卡设计

本节先对关卡、地图的重要性进行说明，之后再介绍地图编辑器的设计与使用，并通过一个简单的地图编辑器的开发实例向读者讲解地图编辑器的开发要领。

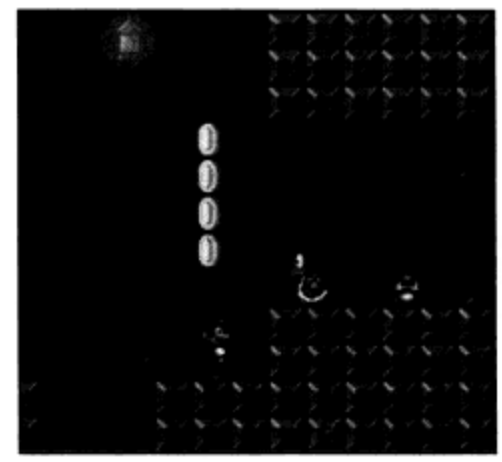
9.1.1　关卡地图的重要性

对于一个高可玩性的游戏来说，关卡地图的好坏是最关键的，下面通过一些成功游戏的例子来说明，关卡地图设计是多么重要的。

- 大家耳熟能详的超级玛丽游戏，该游戏的地图和关卡绝对是非常优秀的，如图 9-1 和图 9-2 所示。



▲图 9-1　超级玛丽游戏画面 1



▲图 9-2　超级玛丽游戏画面 2

- 益智类游戏大富翁的地图也是典型的例子，如图 9-3 和图 9-4 所示。
- 经典 RPG 游戏三国英雄传的地图设计也是比较美观、合理的，如图 9-5 所示。

读者可能很想知道像这些成熟游戏的地图是怎么设计出来的，其实任何一款具有较大地图的游戏，在开发之前一定会开发适合自己的地图编辑器，即地图设计器，有了地图编辑器才能使游戏开发事半功倍。现在网上有很多成熟的游戏地图编辑器，如果是简单游戏的开发可以直接使用，而稍

复杂一些的则必须自己开发或者基于成熟的地图编辑器进行改进。

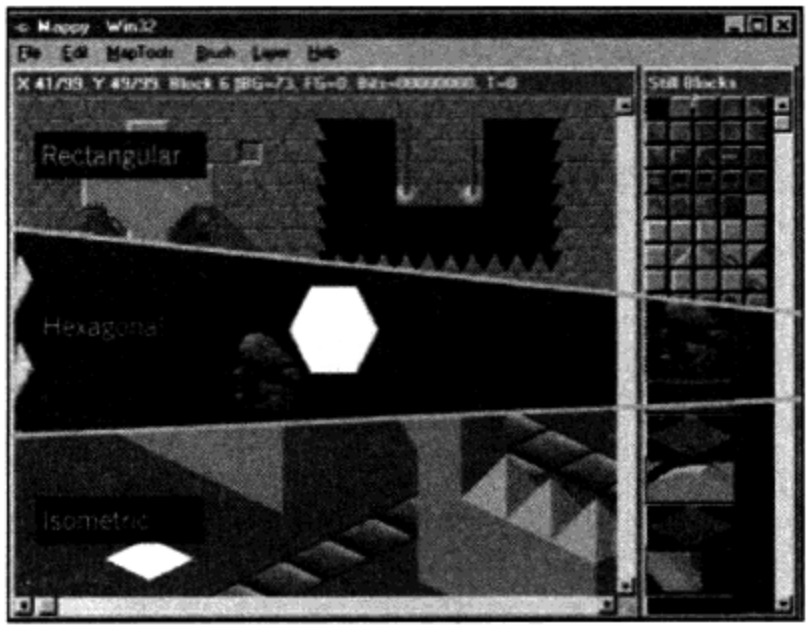
▲图 9-3　手机大富翁游戏截图

▲图 9-4　超级大富翁游戏截图

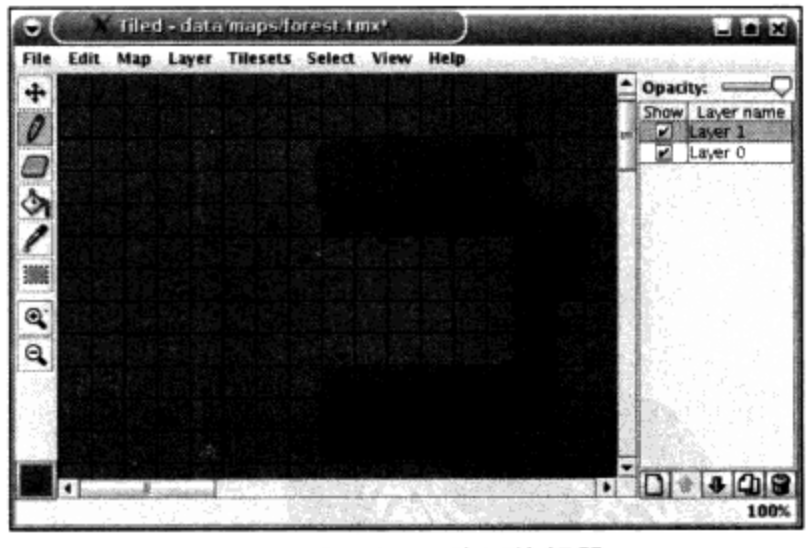
▲图 9-5　三国英雄传

优秀的地图编辑器有很多，下面列出其中几种。

- Mappy 地图编辑器（如图 9-6 所示）是最出名的地图编辑器，其功能强大，可以编辑 2D 和 3D 地图，官方网站是 http://www.tilemap.co.uk/mappy.php。
- Tiled 地图编辑器也非常有名，但其只能编辑 2D 的地图。Tiled 完全由 Java 语言写成，小巧玲珑且功能强大，最重要的是可以免费使用，是本书推荐的地图编辑器，官方网站为 http://mapeditor.org/index.html，其程序运行截图如图 9-7 所示。



▲图 9-6　Mappy 地图编辑器

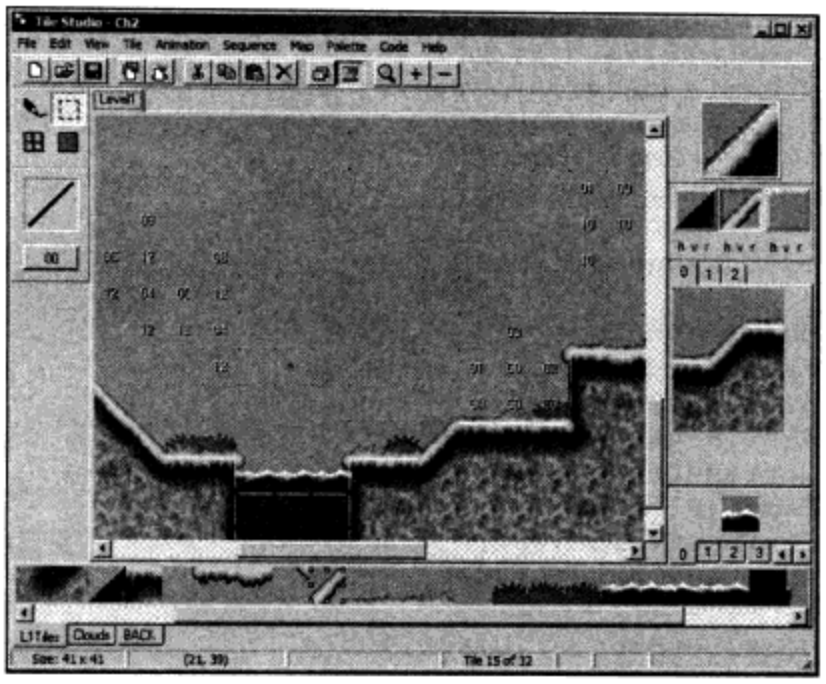


▲图 9-7　Tiled 地图编辑器

- TILE STUDIO 也是一个不错的地图编辑器，它不但具有自定义地图的输出格式，而且还能对图片进行简单的编辑，官方网站是 http://tilestudio.sourceforge.net/，其界面如图 9-8 所示。
- 还有很多其他的地图编辑器，如 Open tUME、Games Factory Pack 3.1 等。

提示

在一个游戏开发时，如果现存的地图编辑器能够满足该游戏的地图设计需求，建议使用已经存在的地图编辑器，而大多数情况下并不能满足设计的需求，这就需要单独开发满足需求的地图编辑器，但开发时可以参考或基于成熟的地图编辑器来简化开发。



▲图 9-8　TILE STUDIO 地图编辑器

9.1.2　图片分割界面的实现

在很多时候，别人的地图编辑器并不适合自己，基本上所有的带有地图的游戏，在开发之前都会开发专用的地图编辑器，接下来将带领读者开发一个简单的地图编辑器，开发步骤如下。

（1）启动 Eclipse，创建新的 Java Project 名为 Sample_9_1。

（2）在 src 目录下创建 MapEditor.java，创建时添加的包为 wyf.ytl。

（3）打开刚刚新建的 MapEditor.java 文件，用下列代码替换已有的代码。

代码位置：见随书光盘中源代码/第 9 章/Sample_9_1/src/wyf/ytl 目录下的 MapEditor.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import java.awt.event.ActionEvent;                        //引入相关类
3    ……//此处省略了各个类的引用代码，读者可自行查看光盘中的代码
4    import javax.swing.event.ChangeListener;                  //引入相关类
5    public class MapEditor extends JFrame implements ActionListener,ChangeListener{
6         private JMenu[] jMenu = {                            //菜单项
7              new JMenu("文件"),
8         };
9         private JMenuItem[] jFileItem = {                    //子菜单项
10             new JMenuItem("打开"),
11        };
12        private JMenuBar jMenuBar = new JMenuBar();          //菜单栏
13        JFileChooser jFileChooser = new JFileChooser();      //文件选择窗口
14        JSlider jSliderX = new JSlider(JSlider.HORIZONTAL,10,70,60);
                                                               //创建两个 JSlider 控件
15        JSlider jSliderY = new JSlider(JSlider.VERTICAL,10,70,60);
16        JLabel jLabel_rows = new JLabel("地图行数:");         //文本
17        JTextField jTextField_rows = new JTextField("10");   //输入框
18        JLabel jLabel_cols = new JLabel("地图列数:");         //文本
19        JTextField jTextField_cols = new JTextField("10");   //输入框
20        JButton jButton = new JButton("确定");                //确定按钮
21        JSpinner jSpinnerX = new JSpinner();                 //创建两个 JSpinner 控件
22        JSpinner jSpinnerY = new JSpinner();
23        JScrollPane jsp;                                     //滚动窗口
24        //SplitPanel sp;
```

```
25          public MapEditor(){                                          //构造器
26              for(JMenuItem item : jFileItem){                         //将子菜单添加到文件菜单下
27                  jMenu[0].add(item);
28                  item.addActionListener(this);
29              }
30              for(JMenu temp: jMenu){                                  //将菜单项添加到菜单栏
31                  jMenuBar.add(temp);
32              }
33              this.setJMenuBar(jMenuBar);
34                                                                       //初始化垂直分割拖拉条
35              jSliderY.setBounds(560,10,40,100);                       //设置位置和大小
36              jSliderY.setMinorTickSpacing(2);
37              jSliderY.setMajorTickSpacing(20);
38              jSliderY.setPaintTicks(true);
39              jSliderY.setPaintLabels(true);
40              this.add(jSliderY);                                      //添加到窗口中
41              jSliderY.setValue(30);                                   //设置初始值
42              jSliderY.addChangeListener(this);                        //添加监听
43                                                                       //初始化水平分割拖拉条
44              jSliderX.setBounds(10,410,100,40);                       //设置位置和大小
45              jSliderX.setMinorTickSpacing(2);
46              jSliderX.setMajorTickSpacing(20);
47              jSliderX.setPaintTicks(true);
48              jSliderX.setPaintLabels(true);
49              this.add(jSliderX);                                      //添加到窗口中
50              jSliderX.setValue(30);                                   //设置初始值
51              jSliderX.addChangeListener(this);                        //添加监听
52              jSpinnerX.setBounds(120, 410, 50, 20);                   //设置位置和大小
53              this.add(jSpinnerX);                                     //添加到窗口中
54              jSpinnerX.setValue(30);                                  //设置初始值
55              jSpinnerX.addChangeListener(this);                       //添加监听
56              jSpinnerY.setBounds(560,120,40,20);                      //设置大小和位置
57              this.add(jSpinnerY);                                     //添加到窗口中
58              jSpinnerY.setValue(30);                                  //设置初始值
59              jSpinnerY.addChangeListener(this);                       //添加监听
60                                                                       //输入自定义地图行数的
                                                                         //文本框
61              jLabel_rows.setBounds(190,410,60,20);                    //设置大小和位置
62              this.add(jLabel_rows);                                   //添加到窗口中
63              jTextField_rows.setBounds(245,410,60,20);                //设置大小和位置
64              this.add(jTextField_rows);                               //添加到窗口中
65                                                                       //输入自定义地图列数的
                                                                         //文本框
66              jLabel_cols.setBounds(320,410,60,20);                    //设置大小和位置
67              this.add(jLabel_cols);                                   //添加到窗口中
68              jTextField_cols.setBounds(375,410,60,20);                //设置大小和位置
69              this.add(jTextField_cols);                               //添加到窗口中
70              jButton.setBounds(450,410,60,20);                        //确定按钮
71              this.add(jButton);                                       //添加按钮到窗口
72              jButton.addActionListener(this);                         //为按钮添加监听
73              this.setTitle("地图设计器 V0.1");                        //设置标题
74              this.setLayout(null);                                    //设置窗口的布局
75              this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);     //关闭按钮
76              this.setBounds(100, 100, 640, 500);                      //设置窗口的大小和位置
77              this.setVisible(true);                                   //设置窗口的可见性
78          }
79          public void actionPerformed(ActionEvent e) {                 //实现接口中的方法
80              ……//该处省略了方法的实现，将在之后的某处给出
81          }
82          public void stateChanged(ChangeEvent e) {                    //实现接口中的方法
83              ……//该处省略了方法的实现，将在之后的某处给出
84          }
```

```
85          public static void main(String[] args) {            //主方法，整个程序的入口
86               new MapEditor();
87          }}
```

- 第 6～11 行创建两个数组，分别表示菜单项以及菜单子项，通过数组管理以后再添加新的菜单是很方便的，直接添加即可，并不需要修改其他代码。
- 第 12～13 行创建一个菜单栏及文件选择窗口，第 14～15 行创建一个水平、一个竖直的滑块 JSlider 控件，第 16～29 行创建两个文本控件、两个文本框控件以及一个按钮。
- 第 24 行因为 SplitPanel 类没有开发，所以先将此处注释掉，以后会在适当的地方取消注释。
- 第 26～33 行初始化窗口的菜单栏。第 34～42 行初始化垂直分割拖拉条，第 44～51 行初始化水平分割拖拉条。
- 第 61～67 行定义了用于输入自定义地图行列数的文本框。
- 第 75 行设置窗口的关闭按钮的事件处理，第 79 行为单击按钮和菜单的事件处理方法，其内容将在之后的步骤中进行开发。
- 第 80 行为滑块 JSlider 与 JSpinner 控件状态改变的监听方法，同样稍后进行开发。

此时运行该项目将得到如图 9-9 所示的效果。

▲图 9-9　地图编辑器

（4）在 src/wyf/ytl 下创建名为 SplitPanel.java 文件，并在其中输入如下代码。

代码位置：见随书光盘中源代码/第 9 章/Sample_9_1/src/wyf/ytl 目录下的 SplitPanel.java。

```
1     package wyf.ytl;                                          //声明包语句
2     import java.awt.Color;                                    //引入相关类
3     import java.awt.Dimension;                                //引入相关类
4     import java.awt.Graphics;                                 //引入相关类
5     import java.awt.Image;                                    //引入相关类
6     import javax.swing.ImageIcon;                             //引入相关类
7     import javax.swing.JPanel;                                //引入相关类
8     public class SplitPanel extends JPanel{
9          Image bigImage;                                      //图元总图片
10         MapEditor father;                                    //MapEditor 的引用
11         public SplitPanel(String path,MapEditor father){     //构造器
12              this.father=father;
13                                                              //加载图元总图片
```

```
14              ImageIcon ii=new ImageIcon(path);
15              bigImage=ii.getImage();
16                                                      //设置面板大小为图元总图片大小
17              this.setPreferredSize(
18                   new Dimension(
19                        bigImage.getWidth(this),      //图片的宽度
20                        bigImage.getHeight(this)      //图片的高度
21                   )
22              );
23         }
24         public void paint(Graphics g){               //重写的绘制方法
25                                                      //在面板中绘制图元总图片
26              g.drawImage(bigImage,0,0,Color.white,this);
27              int imageWidth=bigImage.getWidth(this);          //图元总图片宽度
28              int imageHeight=bigImage.getHeight(this);        //图元总图片高度
29              int xSpan=father.jSliderX.getValue();            //图元宽度
30              int ySpan=father.jSliderY.getValue();            //图元高度
31                                                               //自动绘制竖线
32              g.setColor(Color.green);
33              int countS=imageWidth/xSpan+((imageWidth%xSpan==0)?0:1)+1;
34              for(int i=0;i<countS;i++){                       //循环绘制
35                   if(xSpan*i<=imageWidth){
36                        g.drawLine(xSpan*i,0,xSpan*i,imageHeight);   //画线
37                   }
38              }
39                                                                   //自动绘制横线
40              g.setColor(Color.green);
41              int countH=imageHeight/ySpan+((imageHeight%ySpan==0)?0:1)+1;
42              for(int i=0;i<countH;i++){                           //循环绘制
43                   if(ySpan*i<=imageHeight){
44                        g.drawLine(0,ySpan*i,imageWidth,ySpan*i);    //画线
45                   }
46              }}}
```

- 第 14～15 行加载程序中所用到的图元的总图片。
- 第 17～22 行设置面版大小为图元总图片的大小。
- 第 32～38 行设置画笔的颜色，然后根据图元的宽度和高度循环绘制各条竖线。
- 第 40～46 行设置画笔的颜色，然后根据图元的宽度和高度循环绘制各条横线。

（5）接下来开发开始菜单按下的事件响应，首先需要将第 3 步代码中第 24 行注释取消，然后将下列代码添加到第 3 步代码中的第 80 行。

```
1     if(e.getSource() == jFileItem[0]){                    //单击打开菜单项
2          jFileChooser.showOpenDialog(this);
3          if(jFileChooser.getSelectedFile() != null){
4               String path=jFileChooser.getSelectedFile().getAbsolutePath();
5               sp=new SplitPanel(path,this);               //创建 SplitPanel
6               //将图片分割线面板摆放到滚动窗体中
7               jsp=new JScrollPane(sp);
8               jsp.setBounds(5,5,550,400);                 //设置 SplitPanel 的大小和位置
9               this.add(jsp);                              //添加到窗口
10              this.setVisible(true);                      //设置可见性
11         }
12    }
```

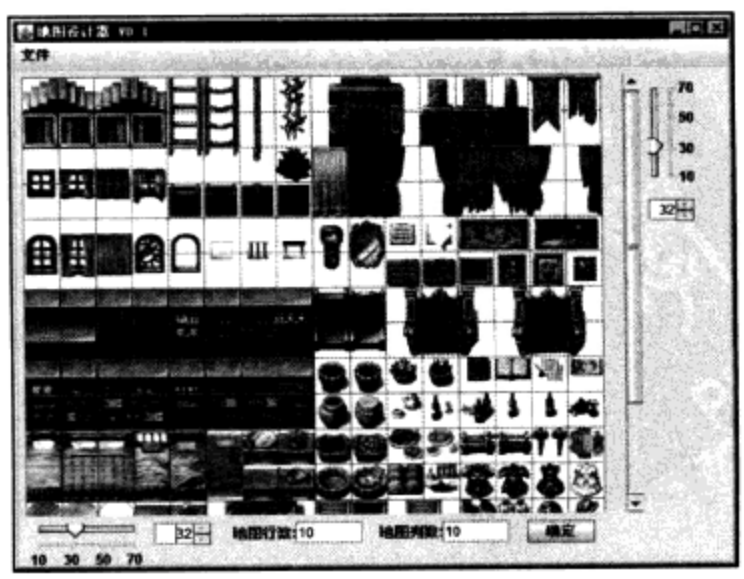
说明　第 1 行当单击菜单中的打开菜单时，先显示一个文件选择窗口，第 4 行根据选择的文件得到路径，然后创建 SplitPanel 并将其显示在 JScrollPane 中。

（6）接下来开发拖动滑块后的事件响应，同样需要将下列代码加入到第 3 步代码中的第 83 行。

```
1    if(sp != null){                                          //当 SplitePanel 引用不为空时
2         if(e.getSource() == jSliderX){
3              sp.repaint();
4              jSpinnerX.setValue(jSliderX.getValue());            //设置 jSpinnerX 的值
5         }
6         else if(e.getSource() == jSliderY){                      //拖动的是 jSliderY
7              sp.repaint();
8              jSpinnerY.setValue(jSliderY.getValue());            //设置 jSpinnerY 的值
9         }
10        else if(e.getSource() == jSpinnerX){                     //拖动的是 jSpinnerX
11             sp.repaint();
12             jSliderX.setValue((Integer)jSpinnerX.getValue());   //设置 jSliderX 的值
13        }
14        else if(e.getSource() == jSpinnerY){                     //拖动的是 jSpinnerY
15             sp.repaint();
16             jSliderY.setValue((Integer)jSpinnerY.getValue());   //设置 jSliderY 的值
17        }
18   }
19   else{
20        if(e.getSource() == jSliderX){                           //拖动的是 jSliderX
21             jSpinnerX.setValue(jSliderX.getValue());            //设置 jSpinnerX 的值
22        }
23        else if(e.getSource() == jSliderY){                      //拖动的是 jSliderY
24             jSpinnerY.setValue(jSliderY.getValue());            //设置 jSpinnerY 的值
25        }
26        else if(e.getSource() == jSpinnerX){
27             jSliderX.setValue((Integer)jSpinnerX.getValue());   //设置 jSliderX 的值
28        }
29        else if(e.getSource() == jSpinnerY){
30             jSliderY.setValue((Integer)jSpinnerY.getValue());   //设置 jSliderY 的值
31        }
32   }
```

说明　该段代码很简单，只是将表示宽度的 JSlider 控件与表示宽度的 JSpinner 中的数值同步。当 SplitePanel 不为空时，需要重新绘制 SplitePanel，也让 SplitePanel 中的横线和竖线的间隔与 JSlider 的数值同步。

（7）此时运行该项目，在界面中依次单击文件/打开菜单项，然后选择一张图元图片，调整线条的间隔后得到的效果如图 9-10 所示。



▲图 9-10　选择图片后的效果图

9.1.3　地图设计界面的实现

前一小节已经将地图编辑器的前一部分开发完毕，接下来将继续对地图编辑器进行完善，加上地图设计功能及代码生成功能。

1. 地图设计界面的实现

首先应该为项目增加地图设计界面，主要步骤如下。

（1）在 src/wtf/ytl 下创建 MapEditorMinor.java 文

件，然后先开发 MapEditorMinor 类的框架，其框架的代码如下。

代码位置：见随书光盘中源代码/第 9 章/Sample_9_1/src/wyf/ytl 目录下的 MapEditorMinor.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    import java.awt.Color;                                              //引入相关类
3    ……//该处省略了部分类的引入代码，读者可自行查阅随书光盘中的源代码
4    import javax.swing.JSplitPane;                                      //引入相关类
5    public class MapEditorMinor extends JFrame
6                    implements MouseListener, ActionListener{           //实现监听接口
7         Image imageZ;                                                  //图元总图片
8         Image imageTemp;                                               //临时图片引用
9         int xSpan;                                                     //图元宽度
10        int ySpan;                                                     //图元高度
11        int imageWidth;                                                //图元总图片宽度
12        int imageHeight;                                               //图元总图片高度
13        int countCols;                                                 //图元列表列数
14        int countRows;                                                 //图元列表行数
15        int rows;                                                      //地图行数
16        int cols;                                                      //地图列数
17        Icon tempii;                                                   //ICON 临时引用
18        JPanel jps;                                                    //上侧显示自定义地图的面板
19        JPanel jpx;                                                    //下侧显示图元列表的面板
20        JScrollPane jsps;                                              //上侧的滚动窗体
21        JScrollPane jspx;                                              //下侧的滚动窗体
22        JSplitPane jspz;                                               //总分割窗体
23        JLabel jpas[][];                                               //上侧地图块数组
24        JLabel jpax[][];                                               //下侧图元块数组
25        int result[][];
26        int tempNumber;
27        private JMenu[] jMenu = {                                      //菜单项数组
28             new JMenu("文件"),
29        };
30        private JMenuItem[] jFileItem = {                              //文件菜单中的子菜单
31             new JMenuItem("生成"),
32        };
33        private JMenuBar jMenuBar = new JMenuBar();                    //菜单栏
34        public MapEditorMinor(Image imageZ,int xSpan,int ySpan,int rows,int cols){
                                                                         //构造器
35             ……//该处省略了方法的实现，将在之后的步骤中给出
36        }
37        public void mouseClicked(MouseEvent e){                        //单击鼠标键方法
38             ……//该处省略了方法的实现，将在之后的步骤中给出
39        }
40        public void actionPerformed(ActionEvent e){
41             ……//该处省略了方法的实现，将在之后的步骤中给出
42        }
43        public void mousePressed(MouseEvent e){                        //空实现接口中的方法
44        }
45        public void mouseReleased(MouseEvent e){                       //空实现接口中的方法
46        }
47        public void mouseEntered(MouseEvent e){                        //空实现接口中的方法
48        }
49        public void mouseExited(MouseEvent e){                         //空实现接口中的方法
50        }}
```

说明　该段代码对程序中所使用到的各个对象进行了声明，并给出了该类中各个方法的框架。

（2）在 MapEditorMinor 的框架搭建完成之后，需要对其中的各个方法进行完善，接下来首先对该类的构造方法进行完善，构造方法主要的工作是界面的搭建，只需将下列代码添加到第（1）步代码中的第 35 行即可。

```
1    this.imageZ=imageZ;                                          //图片的引用
2    this.xSpan=xSpan;
3    this.ySpan=ySpan;
4    this.cols=cols;                                              //列
5    this.rows=rows;                                              //行
6    for(JMenuItem item : jFileItem){                             //将子菜单添加到文件菜单下
7         jMenu[0].add(item);
8         item.addActionListener(this);                           //添加监听
9    }
10   for(JMenu temp: jMenu){
11        jMenuBar.add(temp);                                     //将菜单添加到菜单栏
12   }
13   this.setJMenuBar(jMenuBar);                                  //设置窗口的菜单栏
14   imageWidth=imageZ.getWidth(this);                            //得到图片的宽
15   imageHeight=imageZ.getHeight(this);                          //得到图片的高
16   countCols=imageWidth/xSpan+((imageWidth%xSpan==0)?0:1);
17   countRows=imageHeight/ySpan+((imageHeight%ySpan==0)?0:1);
18   //初始化上边界面
19   jps=new JPanel();
20   jps.setPreferredSize(new Dimension((xSpan+2)*cols,(ySpan+2)*rows));
21   jps.setLayout(null);                                         //设置布局为 null
22   jpas=new JLabel[rows][cols];                                 //创建 JLabel 数组
23   result = new int[rows][cols];                                //创建 int 型数组
24   for(int i=0;i<rows;i++){
25        for(int j=0;j<cols;j++){
26             jpas[i][j]=new JLabel();                           //创建一个 JLabel 对象
27             jpas[i][j].setBackground(Color. RED);              //设置 JLabel 的背景色
28             jpas[i][j].setOpaque(true);
29             jpas[i][j].setBounds(j*(xSpan+2),i*(ySpan+2),xSpan,ySpan);
                                                                  //设置大小和位置
30             jpas[i][j].addMouseListener(this);                 //为刚创建的 JLabel 添加监听
31             jps.add(jpas[i][j]);                               //添加到 jps 中
32        }
33   }
34   jsps=new JScrollPane(jps);
35   //初始化下边界面
36   jpx=new JPanel();
37   jpx.setPreferredSize(new Dimension((xSpan+2)*countCols,(ySpan+2)*countRows));
38   jpx.setLayout(null);
39   jpax=new JLabel[countRows][countCols];
40   for(int i=0;i<countRows;i++){
41        for(int j=0;j<countCols;j++){
42             jpax[i][j]=new JLabel();                           //创建 JLabel 对象
43             jpax[i][j].setBackground(Color.BLUE);              //设置背景色
44             jpax[i][j].setBounds(j*(xSpan+2),i*(ySpan+2),xSpan,ySpan);
                                                                  //设置位置和大小
45             jpax[i][j].addMouseListener(this);                 //添加监听
46             jpx.add(jpax[i][j]);                               //添加到 jpx 中
47        }
48   }
49   jspx=new JScrollPane(jpx);
50   //总界面
51   jspz=new JSplitPane(JSplitPane.VERTICAL_SPLIT,jsps,jspx);//创建 JSplitPane 控件
52   jspz.setDividerLocation(250);                                //设置分割位置
53   jspz.setDividerSize(4);                                      //设置分割线的宽度
54   this.add(jspz);                                              //添加到窗口
```

```
55    //窗体
56    this.setTitle("地图设计器 V0.1");
57    this.setBounds(10,10,640,500);                                  //设置窗口的大小和位置
58    this.setVisible(true);                                          //设置窗口的可见性
59    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);            //关闭按钮
60    //初始化图片
61    for(int i=0;i<countRows;i++){
62          for(int j=0;j<countCols;j++){
63                imageTemp=this.createImage(xSpan,ySpan);   //创建图片
64                Graphics g=imageTemp.getGraphics();        //得到图片的画笔
65                g.drawImage(imageZ,0,0,xSpan,ySpan,j*xSpan,i*ySpan,(j+1)*xSpan,(i+1)*
ySpan,this);                                                 //绘制
66                ImageIcon ii=new ImageIcon(imageTemp);     //通过imageTemp创建ImageIcon
67                jpax[i][j].setIcon(ii);
68          }}
```

- 第 6～13 行设置了窗口的菜单栏的内容。
- 第 18～34 行初始化上边的界面，先创建一个 JPanel，然后对其大小和布局进行设置，再创建一个 JLabel 的数组，添加到窗口的上半部分。
- 第 35～49 行初始化下边的界面，同样是先创建一个 JPanle，对其大小和布局进行设置，然后将新建的 JLabel 数组中的 JLabel 添加到 JPanle。
- 第 50～54 行设置总体界面，通过一个 JSplitPane 控件将界面分成上下两部分。
- 第 55～59 行设置窗口的属性，包括标题、大小、位置、可见性以及窗口关闭按钮的使用。
- 第 60～68 行初始化所有的图片，然后根据图片创建出各个 ImageIcon 并显示到 JLabel 上。

（3）用户界面搭建完成之后应该观察一下其运行效果，所以应该为图片分割界面的确定按钮添加事件响应，当单击确定按钮后会创建并显示该界面。在9.1.2小节的第5步的代码之后，即在MapEditor类的 actionPerformed 方法中添加下列代码。

```
1     else if(e.getSource() == jButton){                          //单击“确定”按钮
2           if(sp != null){
3                 this.dispose();                                 //释放此界面
4                 new MapEditorMinor(                             //创建一个MapEditorMinor窗口
5                       sp.bigImage,                              //图元的引用
6                       jSliderX.getValue(),
7                       jSliderY.getValue(),
8                       Integer.parseInt(jTextField_rows.getText()),        //行
9                       Integer.parseInt(jTextField_cols.getText())         //列
10                );
11          }
12    }
```

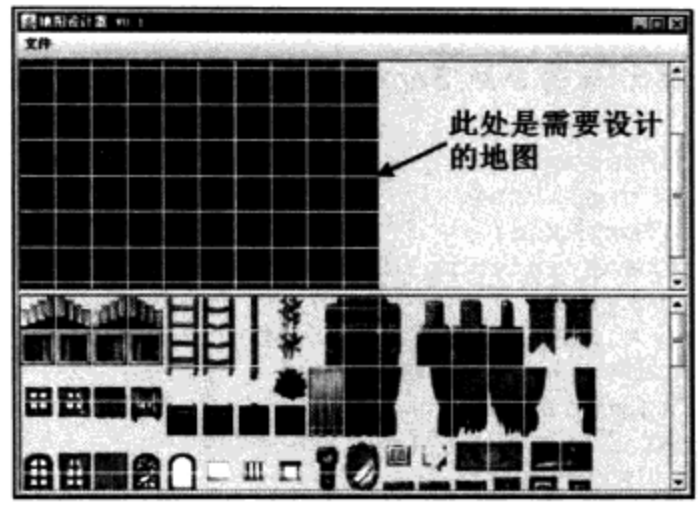
说明　该段代码的作用是，当单击“确定”按钮时，将图片分割界面释放，紧接着创建地图设计界面并将其显示。

（4）此时即可再次运行该项目，在图片分割界面中依次单击文件/打开菜单项，通过文件选择窗口选择一张图片，然后单击“确定”按钮即可切换到地图设计界面，其运行效果如图 9-11 所示。

（5）前面已经将地图设计界面搭建完成，接下来需要为上半部分界面和下半部分界面加上鼠标监听，只需要在第 1 步代码中的第 53 行处加入如下代码。

```
1     Object o=e.getSource();
2     if(o instanceof JLabel){                                    //如果单击的是JLabel
```

```
3          boolean iss=false;                                    //单击的是上面的地图元素
4                                                                //判断是否为地图元素
5          for(int i=0;i<rows;i++){
6               for(int j=0;j<cols;j++){                         //循环
7                    if(jpas[i][j]==o){
8                         iss=true;                              //将 iss 设成 true
9                    }
```



▲图 9-11　地图设计界面的效果图

```
10              }
11         }
12         if(iss){                                              //单击上面地图元素
13              if(tempii!=null){
14                   ((JLabel)o).setIcon(tempii);                //为选中的 JLable 设置 Icon
15                   tempii=((JLabel)o).getIcon();               //得到选择 JLable 的 Icon
16                   for(int i=0;i<rows;i++){
17                        for(int j=0;j<cols;j++){               //循环
18                             if(jpas[i][j]==o){
19                                  result[i][j] = tempNumber;
20                             }
21                        }
22                   }
23              }
24         }
25         else{                                                 //单击下面图元元素
26              tempii=((JLabel)o).getIcon();                    //得到选择 JLable 的 Icon
27              for(int i=0;i<countRows;i++){
28                   for(int j=0;j<countCols;j++){                   //循环
29                        if(jpax[i][j]==o){
30                             tempNumber = i*countCols + j + 1;   //得到单击的格数
31                        }}}}}
```

- 第 3 行为一个布尔类型的变量，表示单击的是否是上半部分的地图元素。
- 第 5～11 行通过对上半部分的元素循环，来判断单击的是否是上半部分的元素。
- 第 12～24 行为单击上半部分元素后的处理代码，先判断之前是否选中过下半部分的元素，当选中过时，将之前选中的下半部分的图标赋给刚刚选中的 JLable，然后将结果存储在存放结果数组中。
- 第 25～31 行为单击下半部分元素的处理代码，先得到选中的 JLbale 的图标，然后计算表示该位置的数值。

2. 生成代码界面的实现

前面已经将地图设计的基本功能开发完成，但是使用地图编辑器的目的是生成地图文件或者是

生成地图代码，所以，还需要为地图编辑器添加上生成代码的功能。在整个项目的开发之前就已经考虑到了地图的保存操作，所以，已经将表示地图元素的数组存储到了一个结果数组中，现在只需根据结果数组生成特定格式的代码即可，具体步骤如下。

（1）在 src/wyf/ytl 下创建 ResultFrame.java 文件。

（2）打开刚创建的 ResultFrame 文件，将下列代码添加到 ResultFrame.java 中去。

代码位置：见随书光盘中源代码/第 9 章/Sample_9_1/src/wyf/ytl 目录下的 ResultFrame.java。

```
1    package wyf.ytl;
2    import javax.swing.JFrame;                                    //添加相关类
3    import javax.swing.JScrollPane;                               //添加相关类
4    import javax.swing.JTextArea;                                 //添加相关类
5    public class ResultFrame extends JFrame{
6         JTextArea jta = new JTextArea();                         //创建一个文本区控件
7         JScrollPane jsp = new JScrollPane(jta);                  //创建一个可滚动窗口
8         public ResultFrame(int[][] result){
9              this.setTitle("结果");                              //设置标题
10             jta.setText("int map[][] = \n {");                  //添加到文本区控件
11             for(int i=0;i<result.length;i++){
12                  String temp = "";                              //创建临时变量
13                  jta.setText(jta.getText()+"\n\t{");            //向文本区添加内容
14                  for(int j=0;j<result[0].length;j++){
15                       temp += result[i][j] + ",";
16                  }
17                  temp = temp.substring(0, temp.length()-1);     //将最后的逗号去掉
18                  jta.setText(jta.getText()+temp);
19                  jta.setText(jta.getText()+"}");
20             }
21             jta.setText(jta.getText()+"\n};");
22             this.add(jsp);                                      //添加到窗口中
23             this.setBounds(100,50,400,500);                     //设置窗口的大小和位置
24             this.setVisible(true);                              //设置窗口的可见性
25             this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE); //关闭按钮
26        }
27   }
```

说明　该类比较简单，根据之前生成的结果数组组织成指定格式的字符串显示在文本区中，然后将文本区放在可滚动的 JScrollPane 中显示出来。

（3）为地图设计界面的菜单添加响应，使得单击生成菜单后显示结果窗口。打开 src/wyf/ytl 下的 MapEditorMinor.java 文件，在其 actionPerformed 方法中（9.1.3 小节地图设计界面的实现中第 1 步的代码的第 56 行）加入下列代码。

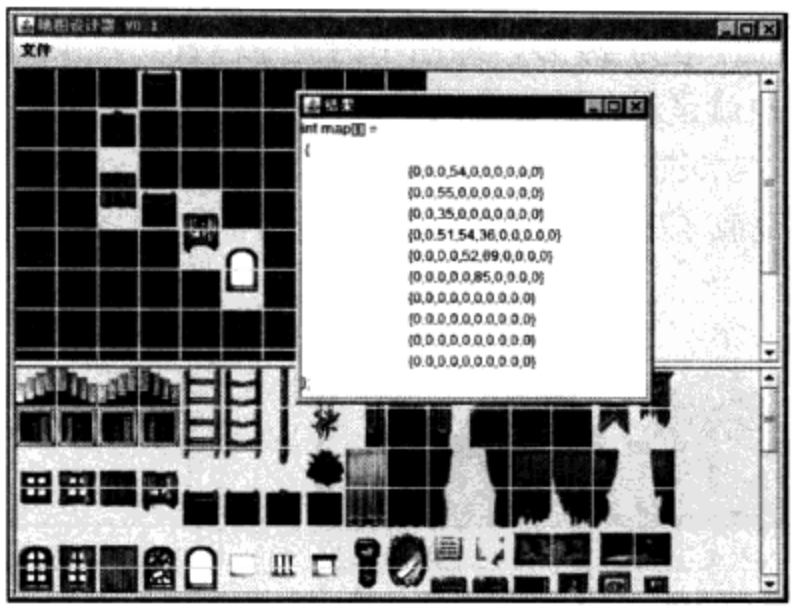
```
1    if(e.getSource() == jFileItem[0]){                            //单击打开菜单项
2         new ResultFrame(result);                                 //创建结果界面
3    }
```

说明　在单击打开菜单项时创建一个 ResultFrame 界面即可。

（4）再次运行该项目，选择一张图片后进入地图设计界面，然后随便设计一些地图用来调试，依次单击文件/生成菜单项，将看到弹出的结果界面，如图 9-12 所示。

到此，整个地图编辑器就已经开发完成，使用此地图编辑器可以可视化地设计简单的地图，然

后生成需要的地图矩阵。



▲图 9-12　生成结果代码效果图

提示

该地图编辑器只是将生成的地图代码显示在文本区中，有兴趣的读者可以运用之前讲过 IO 技术将生成的地图信息存储到文件中。

9.2 游戏中的模糊逻辑

本节主要对游戏中经常应用的模糊逻辑进行介绍，并通过简单的案例介绍，如何在 Android 中将游戏模糊化以及模糊化所带来的好处。

9.2.1　模糊的才是真实的

“模糊逻辑”是 1965 年美国工程师扎得在其改进计算机程序的论著“模糊集合理论”中提出的一个概念。日常生活中，人们经常将事物划分为难易、长短、好坏、高矮、远近等。而对于传统的计算机而言，只能识别是与否、对与错、0 与 1，而对这些模糊的概念就无能为力了。

一般来讲，模糊是相对于精准而言的，在一些受多因素影响的复杂状况下，模糊往往能够显示出更高层次的“精准”。有时，过于硬性精准则会导致过于刻板、缺乏灵活性。实际上，日常生活中有很多事情都是模糊的。例如，进行满意度的调查，如果只给出“满意”与“不满意”这种绝对精准的选项是很不科学的。这种情况下，表面上精准了，实际上结果是没有太大意义的。

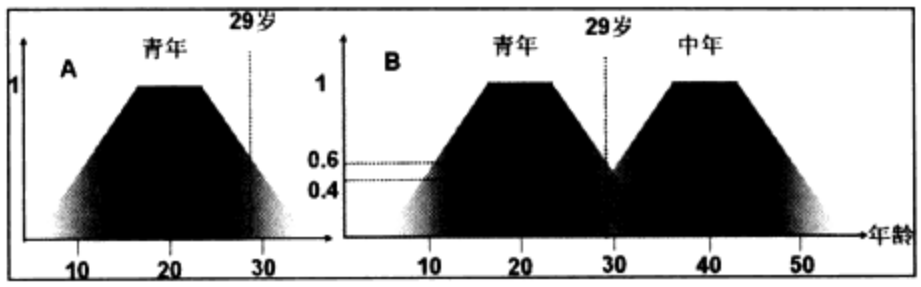
而如果给出一定的“模糊度”，增加“一般”、“有待改进”等不是很精准的选项，就可以在精准和效率之间做出平衡，达到更好的调查效果。

下面通过小强的年龄分段简单介绍什么是模糊逻辑。首先，人的年龄可以分为少年、青年、中年、老年，而小强的年龄是 29，那么小强算是青年还是中年呢？如果没有模糊逻辑，小强应该算是青年，可如果小强今天过生日，难道过完生日就突然变成中年么？这显然不是很合理，在现实生活中，对于青年和中年的分界也没有那么精准。

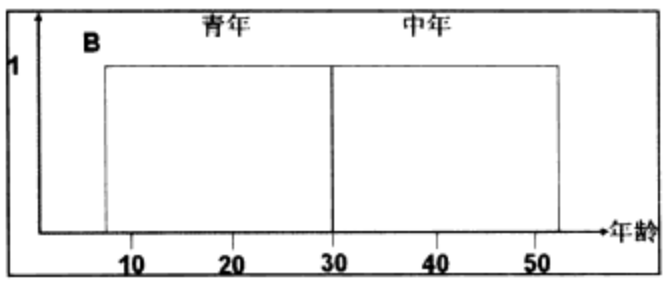
首先看图 9-13 中 A，其表示的是青年阶段人的年龄分布。如果一个人的年龄在 20 左右，那么

一定是青年人，可随着岁数的增大，其属于青年的隶属度就越来越低，到达一定程度后就完全不属于青年。

再看图 9-13 中 B 小强的年龄为 29 岁，即图中的虚线位置，其隶属于青年的隶属度为 0.6，隶属于中年的隶属度为 0.4，也就是说，大部分人们认为小强是青年，而少许人认为小强是中年。而如果没用应用模糊逻辑的情况如图 9-14 所示，如果小强是 29 岁就一定属于青年，而当其到 30 岁时（过了那一秒），马上就变成了中年，这显然过于生硬，不太科学。

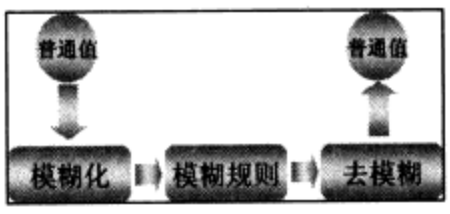


▲图 9-13　模糊逻辑示意图



▲图 9-14　传统情况年龄段示意图

说明　所谓模糊逻辑，一般来说只是让事物分类不那么生硬、绝对，就像小强所属的年龄段一样，不会这一秒是青年，下一秒就变成了中年。在这变化期间，需要一个过渡的灰色时段。



▲图 9-15　基于规则的模糊逻辑

一般情况下，在游戏中应用模糊逻辑的过程如下，如图 9-15 所示。

（1）得到游戏中的一个普通值，然后通过模糊集合分类规则将这个值分配到近似的集合。

（2）使用这个模糊值，计算机就能够根据设定的规则产生一个结果。

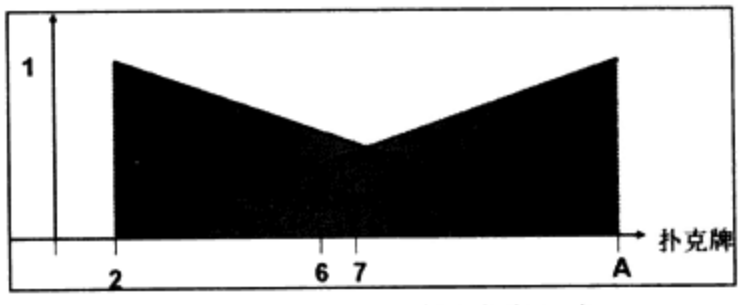
（3）这个结果可能是普通的，但更多的情况下仍然是模糊的，这时可以通过一定规则进行模糊化以得到一个普通值。

（4）在游戏中应用得到的这个普通值。

9.2.2　如何在 Android 中将游戏模糊化

前面介绍了模糊逻辑的原理，接下来通过一个简单的例子来说明模糊逻辑在 Android 中的实现。本案例是砸金花游戏的简化，每次只发一张扑克牌。游戏开始时系统随机发一张扑克牌，然后根据所发的牌自动判断输赢的概率（隶属度）。为了使读者的思路清晰，此处不考虑平局和花色的情况，且假设扑克牌的值的大小从 2～A 逐渐增大。

首先看一下各张扑克牌的输赢隶属度，如图 9-16 所示。随着所发扑克牌值的增大，玩家输的概率就越小、赢的概率就越大。例如，玩家所发的牌为 7，则隶属于输集合的隶属度为 0.59，而隶属于赢集合的隶属度为 0.41。



▲图 9-16　扑克牌的输赢隶属度

下面就根据上述简化的砸金花模型开发一个小应用，步骤如下。

（1）创建一个名为 Sample_9_2 的 Android 项目。

（2）然后打开 main.xml 文件，用下列代码代替已有的代码。

代码位置：见随书光盘中源代码/第 9 章/Sample_9_2/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                           <!-- 添加一个垂直的线性布局 -->
7      <TextView
8            android:id="@+id/myTextView1"
9            android:layout_width="fill_parent"
10           android:layout_height="wrap_content"
11           android:text="请通过发牌按钮发牌"
12       />                                          <!-- 添加一个表示扑克牌的文本 -->
13       <TextView
14           android:id="@+id/myTextView2"
15           android:layout_width="fill_parent"
16           android:layout_height="wrap_content"
17           android:text="赢的概率为："
18       />                                          <!-- 添加一个输赢概率的文本 -->
19       <Button
20           android:id="@+id/button"
21           android:layout_width="wrap_content"
22           android:layout_height="wrap_content"
23           android:text="发牌"
24       />                                          <!-- 添加一个发牌按钮 -->
25   </LinearLayout>
```

- 第 2～6 行定义一个垂直的线性布局。
- 第 7～18 行向垂直的线性布局中添加两个文本控件，分别用来显示扑克牌和输赢的概率。
- 第 19～24 行添加一个发牌按钮。

（3）开发完布局文件之后，接下来开发 Activity，打开 src/wyf/ytl 下的 Sample_9_2.java 文件，用下列代码代替原有代码。

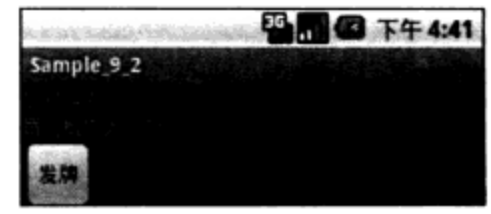
代码位置：见随书光盘中源代码/第 9 章/Sample_9_2/src/wyf/ytl 目录下的 Sample_9_2. java。

```
package wyf.ytl;                                          //声明包语句
import android.app.Activity;                              //引入相关类
import android.os.Bundle;                                 //引入相关类
import android.view.View;                                 //引入相关类
import android.view.View.OnClickListener;                 //引入相关类
import android.widget.Button;                             //引入相关类
import android.widget.TextView;                           //引入相关类
public class Sample_9_2 extends Activity implements OnClickListener{
     TextView myTextView1;                                //表示扑克牌的
     TextView myTextView2;                                //表示输赢的概率
     Button button;                                       //发牌按钮
     String[] puKePai = new String[]{                     //表示扑克牌的数组
          "2","3","4","5","6","7","8","9","10","J","Q","K","A"
     };
     int[] liShuDu = new int[]{                           //表示胜利的隶属度
               0,8,16,24,32,41,49,
               58,66,75,84,91,100
          };
    public void onCreate(Bundle savedInstanceState) {     //重写的 onCreate 回调方法
        super.onCreate(savedInstanceState);
```

```
21          setContentView(R.layout.main);
22          myTextView1 = (TextView) this.findViewById(R.id.myTextView1);
                                                          //得到文本控件的引用
23          myTextView2 = (TextView) this.findViewById(R.id.myTextView2);
                                                          //得到文本控件的引用
24          button = (Button) this.findViewById(R.id.button);  //得到按钮的引用
25          button.setOnClickListener(this);
26      }
27
28      public void onClick(View v) {
29              if(v == button){                          //单击了发牌按钮
30                  int temp = (int)(Math.random()*puKePai.length);    //得到随机数
31                  myTextView1.setText("您的牌为: "+puKePai[temp]);    //取扑克牌
32                  myTextView2.setText("赢的概率为: "+liShuDu[temp]+"%");   //取隶属度
33              }
34      }
35  }
```

- 第 12 行初始化扑克牌数组，用来表示 13 张扑克牌。
- 第 15 行初始化各张牌的胜利的隶属度数组，考虑到演示的目的，此隶属度事先已经算好，而在实际的游戏开发中，可能需要非常复杂的算法来计算隶属度。
- 第 22～23 行得到各个控件的引用。
- 第 28～34 行为监听方法，当单击发牌按钮时得到 0～12 的随机数，然后根据随机数得到扑克牌与赢的隶属度。

（4）运行该项目，单击发牌按钮将得到如图 9-17 所示的效果。



▲图 9-17　简单的模糊逻辑示例

9.3 游戏的基本优化技巧

本节先介绍游戏开发过程中的一些编程技巧以及针对 Android 平台下游戏开发的一些常用的优化技术，然后对游戏的感知性能进行讨论，希望读者在学习本节之后在代码开发方面能更上一层楼。

9.3.1　代码上的小艺术

代码也有其独有的艺术，而且每个开发人员都有其独特的编程风格，在程序员中流行一句话叫“欣赏高人写的代码也是一种享受”。要想编写出优美的代码也不是一件容易的事情，下面列出一些编程过程中最应该注意的问题。

- 代码格式

在项目开发过程中保持代码格式的整齐是一个良好的编程习惯。

- 添加注释

为代码添加适当的注释是非常必要的，就算非常简单代码也应该加上适当的注释，这样才能保证在日后对代码进行维护时快速理解代码的意图。

- 编程要严谨

在项目的开发过程中，严谨的思维是很重要的，往往许多初学者并不注重此点，例如下列代码，读者可能不会发现什么不妥的地方。

```
1    public void test(){                                    //方法
```

```
2          File f = new File("c:\\test1.txt");                          //初始化 File 对象
3          int msg = 0;                                                 //创建临时变量
4          try {
5               FileOutputStream fout = new FileOutputStream(f);        //输出流
6               FileInputStream fin = new FileInputStream(f);           //输入流
7               fout.write(1);                                          //向文件写入 1
8               msg = fin.read();                                       //从文件中读取数据
9               System.out.println(msg);                                //将读取的数据打印
10              fin.close();                                            //关闭输入流
11              fout.close();                                           //关闭输出流
12         } catch (FileNotFoundException e) {                          //捕获异常
13              e.printStackTrace();                                    //打印异常信息
14         } catch(IOException e){
15              e.printStackTrace();                                    //打印异常信息
16         }
17    }
```

其实，上面的代码是不太严谨的。可以想象，如果代码的第 8 行抛出异常，那么输入/输出流将永远不会被关闭。因此，实际开发时应该编写类似下列具有严谨风格的代码。

```
1     public void test(){                                               //方法
2          File f = new File("c:\\test1.txt");                          //初始化 File 对象
3          FileOutputStream fout = null;                                //声明输出流的引用
4          FileInputStream fin = null;                                  //声明输入流的引用
5          int msg = 0;                                                 //创建临时变量
6          try {
7               fout = new FileOutputStream(f);                         //初始化文件输出流
8               fin = new FileInputStream(f);                           //初始化文件输入流
9               fout.write(1);                                          //向文件写入 1
10              msg = fin.read();                                       //从文件中读取数据
11              System.out.println(msg);                                //将读取的数据打印
12         } catch (FileNotFoundException e) {                          //捕获异常
13              e.printStackTrace();                                    //打印异常信息
14         } catch(IOException e){
15              e.printStackTrace();                                    //打印异常信息
16         } finally{
17              try {
18                   if(fin != null){                                   //当输入流不为空时
19                        fin.close();                                  //关闭流
20                        fin = null;                                   //并将输入流的引用置空
21                   }
22              } catch (IOException e) {
23                   e.printStackTrace();                               //打印异常信息
25              }
25              try {
26                   if(fout != null){                                  //当输出流不为空时
27                        fout.close();                                 //关闭输出流
28                        fout = null;                                  //将输出流的引用置空
29                   }
30              } catch (IOException e) {                               //捕获异常
31                   e.printStackTrace();                               //打印异常信息
32              }}}
```

提示　读者可能会发现，正确严谨的代码会比普通的代码长出很多，但只有这样代码才是最安全的。

- 尽量使用常量而非字面常量

开发过程中应该尽量避免直接重复使用同样的字面常量，需要时应该将其声明为常量（由

public static final 修饰）。一般的项目中，都应该设计一个常量类，将项目中所有类用到的常量放在其中统一管理。

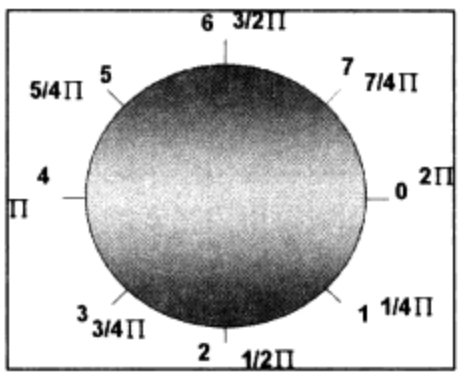
提示	开发一段实现指定功能的代码很容易，但是要想开发出既优美、又严谨的代码就不那么容易了，只有在平时练习时养成良好的编程习惯，才有可能在真正的项目中开发出优秀的、易维护的代码。

9.3.2 Android 中的查找表技术

前一节介绍了一些关于代码风格的问题，而本节将针对 Android 平台介绍在游戏开发中最常用的优化技巧——查找表技术。

查找表是一个比较古老的优化技巧，其基本思想是，事先对需要大量计算的算法进行运算并将结果存放在一个表中。在游戏中使用时，只要直接从表中取值，而不是每次重新运算，这样就可以大大提高运行速度。

例如，在游戏开发中，会经常需要计算三角函数值。这时，就可以通过查找表技术对游戏进行优化，提高游戏的运行速度。下面给出一个正弦三角函数查找表应用的例子，其详细步骤如下。



▲图 9-18 查找表示意图

（1）首先确定查找表的长度，即事先需要计算出多少个正弦三角函数值。本例中计算一周 8 个正弦值，如图 9-18 所示。

（2）为每个正弦三角函数计算其值并填写查找表，为了程序中使用方便，给 8 个值编上号，之后便可按编号顺序生成相应的数组代码，new double[]{0,0.707,1,0.707,0,–0.707,–1,–0.707}。

（3）新建名为 Sample_9_2 的 Android 项目。

（4）打开 res/layout 下的 main.xml 文件，用下列代码代替已有代码。

代码位置：见随书光盘中源代码/第 9 章/Sample_9_3/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- XML 的版本以及编码方式 -->
2    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:orientation="vertical"
4        android:layout_width="fill_parent"
5        android:layout_height="fill_parent"
6        >                                                <!-- 添加一个垂直的线性布局 -->
7      <TextView
8            android:layout_width="fill_parent"
9            android:layout_height="wrap_content"
10           android:text="请输入需要计算的角度"
11     />                                                 <!-- 添加一个文本控件 -->
12       <EditText
13             android:id="@+id/myEditText1"
14             android:layout_width="100px"
15             android:layout_height="wrap_content"
16       />                                               <!-- 添加一个 EditText 控件 -->
17     <Button
18             android:id="@+id/myButton"
19             android:layout_width="wrap_content"
20             android:layout_height="wrap_content"
21             android:text="计算正弦"
```

```
22      />                                                    <!-- 添加一个 Button 控件 -->
23      <TextView
24              android:id="@+id/myEditText2"
25              android:layout_width="fill_parent"
26              android:layout_height="wrap_content"
27              android:text="运算结果为："
28          />                                                <!-- 添加一个 TextView 控件 -->
29    </LinearLayout>
```

- 第 2～6 行添加一个垂直的线性布局。
- 第 7～28 行向垂直的线性布局中添加几个控件并为其设置 ID。

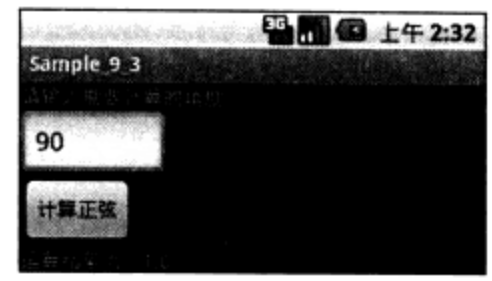
（5）开发完布局文件之后，应该开发 Activity 的代码，打开 src/wyf/ytl 下的 Sample_9_3.java 文件，在其中输入如下代码。

```
1     package wyf.ytl;                                          //声明包语句
2     import android.app.Activity;                              //引入相关类
3     import android.os.Bundle;                                 //引入相关类
4     import android.view.View;                                 //引入相关类
5     import android.view.View.OnClickListener;                 //引入相关类
6     import android.widget.Button;                             //引入相关类
7     import android.widget.TextView;                           //引入相关类
8     public class Sample_9_3 extends Activity implements OnClickListener{
9          TextView myTextView1;                                //TextView 的引用
10         TextView myTextView2;                                //TextView 的引用
11         Button myButton;//Button 的引用
12         double[] values = new double[]{0,0.707,1,0.707,0,-0.707,-1,-0.707};
                                                                //查找表数组
13             public void onCreate(Bundle savedInstanceState) {
                                                                //重写的 onCreate 方法
14             super.onCreate(savedInstanceState);
15             setContentView(R.layout.main);
16             myTextView1 = (TextView) findViewById(R.id.myEditText1);
                                                                //得到 myTextView1 的引用
17             myTextView2 = (TextView) findViewById(R.id.myEditText2);
                                                                //得到 myEditText2 的引用
18             myButton = (Button) findViewById(R.id.myButton);
                                                                //得到 myButton 的引用
19             myButton.setOnClickListener(this);               //添加监听
20             }
21         public void onClick(View v) {
22             if(v == myButton){                               //按下计算按钮
23                 int temp = 0;                                //用户输入角度
24                 int index = 0;
25                 try{
26             temp = Integer.parseInt(myTextView1.getText().toString());
                                                    //将 myTextView1 中的值转换成整数
27                 }
28                 catch(Exception e){}                         //如果不能转换成整数则不转换
29                 temp = temp%360;                             //将角度换算成 0° ~360°
30                 index = temp/45                              //得到索引值
31                 myTextView2.setText("运算结果为："+values[index]);
32             }
33         }
34    }
```

- 第 12 行为之前生成的查找表数组。
- 第 14～19 行先得到各个控件的引用，然后为按钮控件添加监听器。
- 第 21～33 行先得到用户输入的角度，然后根据输入的角度计算出所属范围的编号，从查找

表的数组中直接取得结果。

（6）最后运行该项目，在文本框中输入角度后单击“计算正弦”按钮，得到的结果如图 9-19 所示。



▲图 9-19　查找表例子的运行效果

9.3.3　游戏的感觉和性能问题

游戏开发中主要影响可玩性的问题就是感觉和性能的问题，可往往这两个问题最不容易解决。因为一个游戏可玩性的高低不只与程序员开发的代码有关，同时与美工设计、音效设计以及情节设计都有很大关系，而且还与游戏的类型、性能有关，下面列举了几种可能的情况。

- 棋牌类的游戏可玩性的高低主要取决于程序员采用的 AI 算法，也就是说机器要有恰当的“智商”，太聪明或太笨都不合适。
- RPG 游戏程序员编程所起的作用要相对小一些，反而情节设计和美工设计占据的重要性要相对大一些。
- 实时对战类游戏不但要有精美的画面，恰当的音效、性能也是决定其可玩性的重要方面。玩家显然不希望在大量对战的游戏中看到很卡的画面，因为画面中绘制的元素增多后，FPS（Frames Per Second，帧速率）就急速下降。

其实，性能中直接影响用户感觉的主要就是游戏的 FPS，因为 FPS 的设置直接影响视觉效果，而往往这些参数的设置需要多年的经验以及反复的尝试。下面来通过一个 FPS 设置的例子简单介绍人眼对动画的感觉问题。

后面章节的太空战役游戏中，背后需要有一个大场景淡入/淡出的动画，而飞机飞行时需要在屏幕中按指定路线移动。如果采用相对较低的 FPS，后面大场景的效果看起来会很好，而移动的飞机则会有些停顿。其实有经验的游戏开发人员都知道，淡入/淡出的动画在帧频很低时（10FPS 左右）效果就不错，而移动物体的帧频则必须较高时（25FPS 左右）才足以让人眼感觉不到生硬。

下面列出了一些游戏中常用的优化技术。

- 进度条的使用

在游戏适当的地方添加上进度条，如后台需要长时间加载某些资源时，不应该使画面停顿，而是应该先将屏幕切换成进度条状态，然后在后台去加载需要的资源，当加载完成后通知前台界面切换屏幕至需要的屏幕。

- 后台线程的使用

这种技术的目的也是使用户感觉到程序的流畅，将耗时的工作从主线程中提取主来，放到后台线程中，使前台的用户界面不会失去响应，以提高用户体验。

- 对图片文件进行合并

该技术对于内存较小、性能较低的移动设备尤其有用。将图片文件合并为一个大图文件会节省内存，在加载图片时，先将图元图片加载进内存，然后按照一定规则将图片分成适当的大小并存储在图片数组中，再将大图片释放掉。

- 只绘制有用部分

该技术提高游戏性能的效果是最明显的，按照一定的规则，每次需要绘制屏幕时只绘制有用的

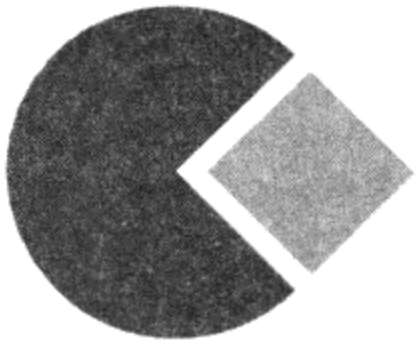
部分，而没有用的地方一律不绘制。如屏幕外的图片不需绘制、绘制多层的场景时下层中被上层遮住的部分不需要绘制等。

提示

很多情况下“性能”的好坏与原始数据和编程使用的算法关系不大，而是与用户的感觉有很大关系。例如，上述将耗时的工作放在后台线程进行，让前台线程能快速满足用户界面响应的策略，并不一定提高了系统的实际整体性能。但用户体验变好了，“性能”也就提高了。

9.4 小结

本章主要介绍了游戏开发的几个小秘技，从地图编辑器的使用到开发，从模糊逻辑到游戏的性能优化，使读者了解到游戏开发所涉及的几个重要领域。同时提醒读者，开发一款优秀的游戏是需要对游戏相关各个领域知识进行综合运用的。



第 10 章　游戏的心脏——物理引擎

汽车引擎是汽车的心脏，其决定了汽车的性能和稳定性，是人们在购车时最关注的。而游戏中的物理引擎就如汽车的引擎一样，占据了非常重要的位置。一款好的物理引擎可以非常真实地模拟现实世界，使得游戏更加逼真，提供更好的娱乐体验。

10.1 物理引擎很重要

经过前面章节的学习，读者可能对物理引擎已经有所了解，但那是远远不够的。本节将为读者简单介绍物理引擎的一些基本概念。

10.1.1　什么是物理引擎

物理引擎通过给物体赋予真实的物理属性来模拟物体的运动，包括碰撞、移动、旋转等。并不是所有的游戏都必须使用独立的物理引擎，一些简单的游戏的物理功能可以通过自行开发碰撞检测及实现力学公式来实现对刚体及质点的模拟，就像前面第 7 章介绍的一样。

当游戏需要实现比较复杂的刚体碰撞、滚动或者弹跳时，通过全部自行编程的方式实现就非常困难，成本也很高。遇到这种情况时就可以使用独立的物理引擎来模拟物体的运动，使用物理引擎不仅可以得到更加真实的结果，对于开发人员来说也比自行开发要耗时短、效率高。

一款好的物理引擎不仅会帮助实现碰撞检测、力学公式模拟，而且还会提供很多机械结构的实现，如滑轮、齿轮、铰链等。这些主要是通过关节来实现的，详细内容在后继部分进行介绍。更高级的物理引擎不但可以提供刚体的模拟，甚至还可以提供软体及流体的模拟，这些都能帮助游戏提升真实感和吸引力。

10.1.2　常见的物理引擎

市面上存在的物理引擎数量是很多的，著名的物理引擎有 Havok、Bullet、PhysX、ODE 以及 Box2D 等。其中 ODE、Bullet、Box2D 等是开源的物理引擎，而 PhysX 的前身是 Novodex，当被 Ageia 收购之后改名为 PhysX，是一款可以免费用于非商业用途的引擎，商业用途及源代码需要付费，Havok 在许可方面也是如此。上述几种物理引擎的基本信息如表 10-1 所列。

表 10-1　　知名物理引擎的基本信息

物理引擎名称	Havok	PhysX	Bullet	Box2D	ODE
持有公司/人员	Intel	Nvidia	AMD	Erin Catto	Russell Smith
是否开源	否	否	是	是	是
是否支持 C/C++	是	是	是	是	是
最新版本	5.5	2.1.2	2.78	2.1.2	0.11.1
文档情况	详细	详细	尚可	尚可	一般

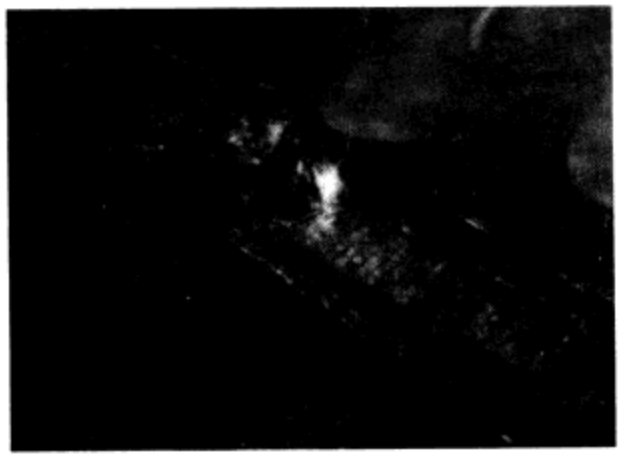
提示　三大 3D 物理引擎为 Havok、PhysX 和 Bullet。另外 2008 年 2 月 4 日，NVIDIA 成功收购 AGEIA，支援 CUDA 技术的显卡，就可以启动硬件 PhysX 加速。

- Havok

Havok 成立于 1998 年，总部位于都柏林。在 2000 年游戏开发者大会上发布了 Havok 1.0 ，最新版本为 Havok5.5，该引擎基于 C/C++。2007 年 9 月，Intel 宣布成功收购 Havok。之后，Intel 宣布 Havok 引擎开放源代码并允许游戏开发人员免费用于非商业用途。

由于 Havok 的开放性和不依赖特定硬件的特点，很多大型游戏均使用 Havok 引擎。其中著名的有星际争霸 2、暗黑破坏神 3 等，效果分别如图 10-1 和图 10-2 所示。

▲图 10-1　星际争霸 2



▲图 10-2　暗黑破坏神 3

提示　在 2011 年 3 月 6 号，Intel 宣布 Havok 引擎开始支持 Android。并且已经与索爱达成合作关系，Havok 将为索爱的 Xperia Play 游戏手机进行优化以便用于游戏的开发。

- PhysX

PhysX 不仅可以由 CPU 计算，而且其程序本身在设计上可以使用独立的浮点处理器来计算。正是由于这个原因，其可以非常轻松地完成像流体力学那样计算量非常大的物理模拟计算。该引擎可以在 Windows，Linux，Xbox360，Playstation3，Mac 等多种平台上运行。

到目前为止在 X86-PC 平台上，共有 222 种游戏采用 PhysX 引擎。被大家熟知的有雪域危机、虚幻竞技场 3，其运行效果如图 10-3 和图 10-4 所示。

- Bullet

Bullet 是一款开源的物理引擎，是 AMD 开放物理计划成员之一。同时其也是一个跨平台的物理引擎，支持 Windows、Linux、MAC、Playstation3、XBOX360 以及 Nintendo Wii 等主流平台。

使用 Bullet 物理引擎开发的游戏主要有侠盗猎车 4，其效果如图 10-5 所示。同时使用该引擎制作的电影也不乏好莱坞大作，如 2012 就是使用的该引擎，其效果如图 10-6 所示。

▲图 10-3　雪域危机

▲图 10-4　虚幻竞技场 3

▲图 10-5　侠盗猎车 4

▲图 10-6　电影 2012 场景

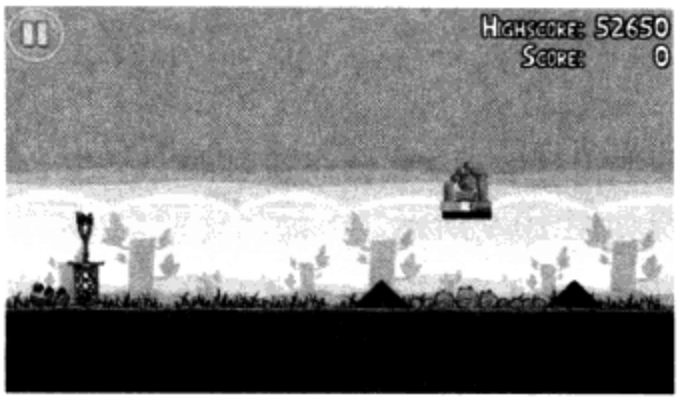
- ODE

ODE（Open Dynamic Engine）是一款免费的具有工业品质的刚体动力学引擎。其可以非常好的仿真现实中物体的移动、旋转等，具有快速、强健和可移植性，并且内置碰撞检测系统。ODE 目前可以支持球窝、铰链、滑块、定轴、角电机和 hinge-2 等连接类型，还可以支持各种碰撞形式（如球面碰撞和平面碰撞）和多个碰撞空间。

提示　由于 ODE 现在还没有正式发布 1.0 版本，最新版为 0.11.1，还属于不成熟阶段，因此应用不如前几种广泛。

- Box2D

Box2D 是一款非常著名的 2D 物理引擎，主要用于 2D 刚体仿真，有 C++、Flash 和 Java 等版本，其可以使物体的运动更加真实可信。Box2D 最早是用可移植的 C++开发的，后来随着需求的变化陆续推出了 Java 与 Flash 的版本。非常受欢迎的游戏《愤怒的小鸟》就是基于该引擎开发的，其运行效果如图 10-7、图 10-8 所示。



▲图 10-7　游戏开始

▲图 10-8　游戏进行中

10.2　2D 的王者 JBox2D

JBox2D 是开源的物理引擎 Box2D 的 Java 版本，可以直接用于 Android。由于 JBox2D 的图形渲染使用的是 Processing 库，因此在 Android 平台上使用 JBox2D 时，图形渲染工作只能自行开发。该引擎能够根据开发人员设定的参数，如重力、密度、摩擦系数和弹性系数等，自动地进行 2D 刚体物理运动的全方位模拟。

每种物理引擎都有其独特的概念，在学习开源的物理引擎时，首先需要弄明白的就是其基本概念。因此，本节主要为读者复习一下物理学中的一些基本概念，并介绍 JBox2D 中的一些常用类与概念。

提示　由于本书主要介绍 2D 方面的游戏开发，因此本章主要的篇幅用于为读者介绍 2D 的王者 JBox2D（身为 Android 的开发人员自然选用 Java 版）。

10.2.1　基本的物理学概念

游戏是对真实世界的仿真，其中用到了许多物理学知识，如密度、质量、质心、摩擦力、扭矩以及碰撞（恢复）系数等。接下来，本小节将简要介绍用 JBox2D 开发游戏时经常用到的一些物理学概念。

- 密度

物理学中密度指的是单位体积的质量，符号为“ρ”，常用单位为 kg/m^3。其是物质的一种基本特性，不随物体的质量、体积的改变而改变，同种物质的密度相同。

- 质量

质量指的是物体中所含物质的量，即物体惯性的大小，国际单位是 kg。同一物体的质量通常是一个常量，不因高度、经度或者纬度的改变而变化。但是根据爱因斯坦的相对论，同一物体的质量会随着速度的变化而改变。只有运动接近光速才能感觉到这种变化，因此在游戏中一般不考虑速度对质量的影响。

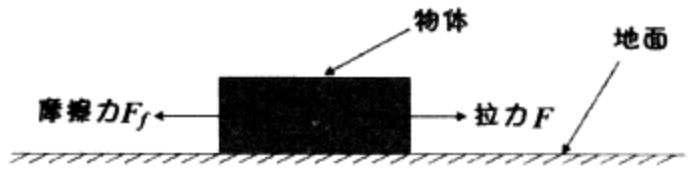
- 质心

物体（或物体系）的质量中心，是研究物体（或物体系）机械运动的一个重要参考点。当作用力（或合力）通过该点时，物体只作移动而不发生转动；否则在发生移动的同时物体将绕该点转动。

研究质心的运动时，可将物体的质量看作集中于质心。理论上，质心是对物体的质量分布用“加权平均法”求出的平均中心。

- 摩擦力

当两个互相接触的物体，如果要发生或者已经发生相对运动。就会在接触面上产生一种阻碍该相对运动的力，这种力就称之为摩擦力。其基本情况如图 10-9 所示。



▲图 10-9　摩擦力示意图

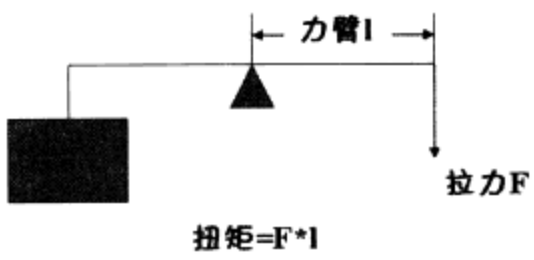
提示　根据物体是否发生相对运动可以分为静摩擦力与滑动摩擦力，实际开发中可以进行简化，但若要模拟更加真实的效果就需要分别开发。

- 扭矩

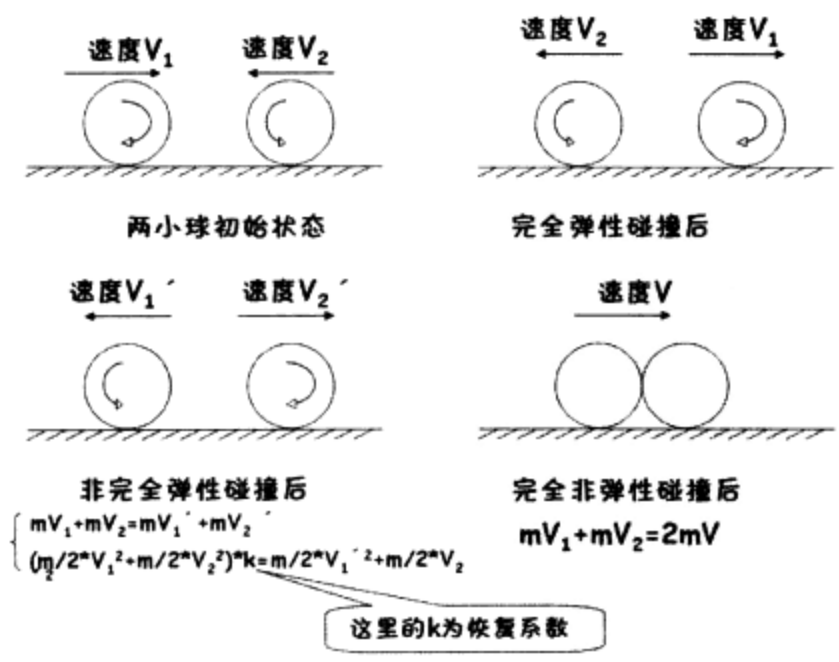
扭矩在物理学中就是力矩的大小，等于力与力臂的乘积，国际单位是 Nm（牛米）。在力臂不变的情况下，力越大，扭矩越大。基本情况如图 10-10 所示。

- 恢复系数

两物体碰撞后的总动能与碰撞前的总动能之间的比称之为恢复系数，其取值范围为 0～1。如果恢复系数为 1，则碰撞为完全弹性碰撞，满足机械能守恒；如果恢复系数小于 1 并且大于 0，则为非完全弹性碰撞，不满足机械能守恒，这种情况是最常见的；如果恢复系数为 0，则为完全非弹性碰撞，两个物体会粘在一起。基本情况如图 10-11 所示。



▲图 10-10　扭矩示意图



▲图 10-11　各种碰撞情况

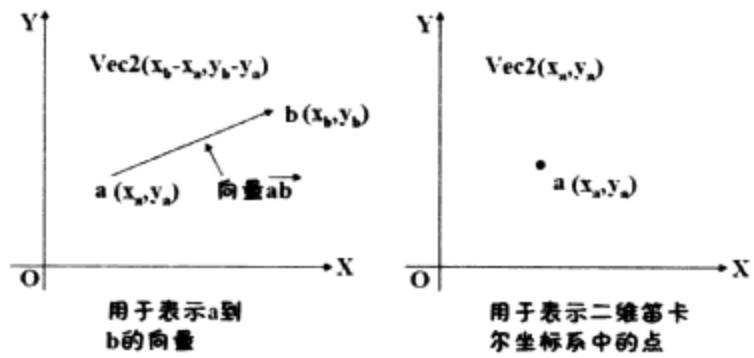
说明　在图 10-11 中，两个小球的质量相等，小球的初速度 V_1 大于 V_2。

10.2.2　JBox2D 中常用类的介绍

俗话说得好“基础不牢，地动山摇”。在学习新技术时，首先要学习的就是该技术的一些基本概念，因此本小节主要介绍 JBox2D 中一些必知必会的类。

1. Vec2 类（二维向量）

该类属于 org.jbox2d.common 包，继承自 java.lang.Object 类，其对象用于表示一个二维向量或二维笛卡尔坐标。二维向量 Vec2 类的基本功能如图 10-12 所示。



▲图 10-12　二维向量示意图

该类在 JBox2D 中的使用频率非常高，通常用于表示物体的位置、速度等，其构造器与常用方法及属性如表 10-2 所列。

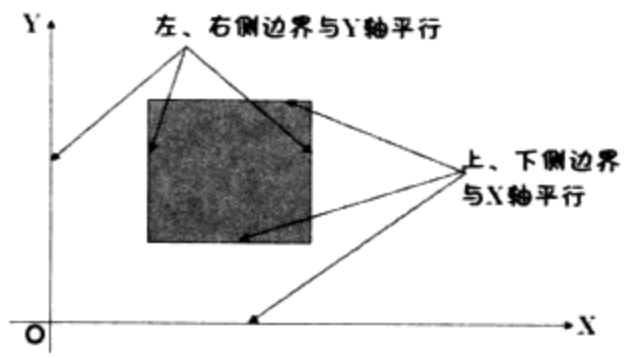
表 10-2　　构造器与主要方法

方法、属性或构造器签名	说　　明	类　　型
public Vec2(float x,float y)	创建一个 Vec2 的对象，该对象表示 *X* 轴分量为 x，*Y* 轴分量为 y 的一个向量或坐标系中的(x,y)点	构造器
public static Vec2 abs(Vec2 a)	求向量的绝对值	方法
public Vec2 add(Vec2 v)	返回该向量与另一向量的和，并不改变其中的任何一个的值	方法
public Vec2 addLocal(Vec2 v)	返回该向量与另一向量的和，并改变当前的向量	方法
public Vec2 clone()	复制向量	方法
public float length()	返回该向量的长度	方法
public float lengthSquared()	返回该向量长度的平方	方法
public static Vec2 max(Vec2 a,Vec2 b)	返回两个向量中最大的	方法
public static Vec2 min(Vec2 a,Vec2 b)	返回两个向量中最小的	方法
public void set(float x,float y)	设置向量的两个分量	方法
public void set(Vec2 v)	设置当前向量为向量 v	方法
public void setZero()	将向量设置为 0 向量	方法

2. AABB 包围盒

该类属于 org.jbox2d.collision 包，继承自 java.lang.Object 类，表示轴对齐的边界盒子。所谓的

轴对齐是指，盒子左、右侧边界与 *Y* 轴平行，同时上、下侧边界与 *X* 轴平行，基本情况如图 10-13 所示。



▲图 10-13　轴对齐

该类的构造器与常用方法及属性如表 10-3 所列。

表 10-3　　构造器与常用方法

方法、属性或构造器签名	含　义	类　型
public AABB()	创建 AABB 轴对齐的边界盒子对象	构造器
public Vec2 lowerBound	此属性用于指定包围盒的左、上边界	属性
public Vec2 upperBound	此属性用于指定包围盒的右、下边界	属性
public boolean isValid()	检查 AABB 对象是否合法	方法
public boolean testOverlap(AABB box)	检测两个 AABB 对象是否重叠（碰撞）	方法

提示　使用此类对象创建边界时，使用的度量单位为现实世界中的单位（m-米）。同时 AABB 应该永远大于等于物体实际所占区域。

3. BodyDef 类（刚体描述）

该类属于 org.jbox2d.dynamics 包，继承自 java.lang.Object 类，其对象用于存储刚体的一些描述性信息，主要是在创建刚体时调用。该类的构造器与常用方法及属性如表 10-4 所列。

表 10-4　　构造器与常用方法

方法、属性或构造器签名	含　义	类　型
public BodyDef()	创建 BodyDef 对象	构造器
public boolean allowSleep	是否允许刚体休眠的标志位	属性
public float angle	该刚体在世界坐标系中的角度	属性
public boolean isBullet	设置是不是时刻检测碰撞（一般为子弹类刚体服务）	属性
public boolean isSleeping	表示刚体当前是否处于休眠状态的标志位	属性
public MassData massData	刚体的质量	属性
public Vec2 position	刚体的位置	属性

4. World 类（世界）

该类属于 org.jbox2d.dynamics 包，继承自 java.lang.Object 类，表示的是物理世界。一个物理世界就是物体、形状和约束相互作用的集合。JBox2D 支持同时创建多个世界，但这通常是不必要的。要创建一个世界对象，首先需要创建一个世界的 AABB 包围盒，JBox2D 使用 AABB 包围盒来加速碰撞检测。

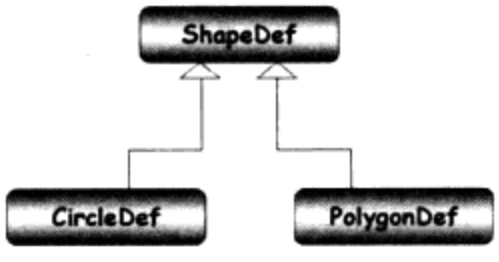
该类构造器与常用方法及属性如表 10-5 所列。

表 10-5　　构造器与常用方法

方法、属性或构造器签名	含　义	类　型
public World(AABB worldAABB,Vec2 gravity,boolean doSleep)	创建 World 类的对象 worldAABB ——整个世界的 AABB 包围盒 gravity——重力向量 doSleep——是否允许刚体休眠	构造器
public Body createBody(BodyDef def)	根据给出的 BodyDef 对象创建刚体	方法
public Joint createJoint(JointDef def)	根据给出的 JointDef 对象创建关节	方法
public void destroyBody(Body b)	从世界中销毁刚体，如游戏中的物体爆炸后消失就可以采用其完成	方法
public void destroyJoint(Joint j)	销毁关节，这个可能影响连接的刚体进行碰撞	方法
public int getBodyCount()	得到刚体的数量	方法
public Body getBodyList()	得到刚体的列表	方法
public Vec2 getGravity()	得到重力向量	方法
public AABB getWorldAABB()	得到世界的 AABB 包围盒对象	方法
public void setGravity(Vec2 gravity)	设置重力向量	方法
public void step(float dt,int iterations)	进行一轮迭代模拟 dt——模拟的频率 iterations——迭代次数。迭代越大，模拟越精确、同时性能越低	方法

5. ShapeDef 类（形状描述）

该类属于 org.jbox2d.collision 包，继承自 java.lang.Object 类，其对象用来存储形状的描述性信息。该类有两个子类，分别为 CircleDef（圆形描述），PolygonDef（多边形描述），3 个类的继承关系如图 10-14 所示。



▲图 10-14　ShapeDef 类的继承树

该类的基本属性如表 10-6 所列。

表 10-6　　ShapeDef 类属性

属性签名	含　义
public float density	密度，单位为 kg/m^2
public float friction	摩擦系数
public float restitution	弹性系数（碰撞恢复系数）
public java.lang.Object userData	用这个来存储自定义的用户数据

6. CircleDef 类（圆形描述）

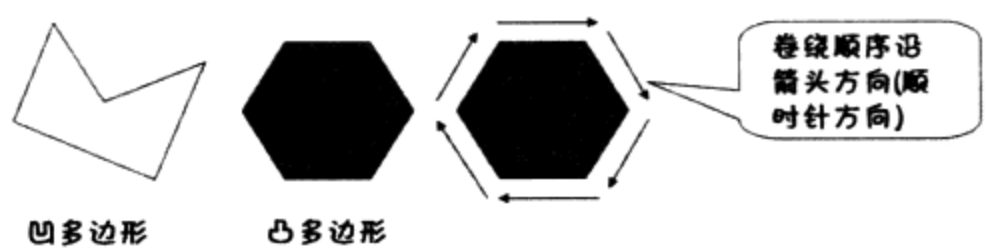
该类属于 org.jbox2d.collision 包，继承自 ShapeDef 类，其对象用于存储圆形的一些描述性信息，如圆心坐标、半径等。该类的构造器及属性如表 10-7 所列。

表 10-7　　CircleDef 类构造器及属性

构造器及属性签名	含　义	类　型
public CircleDef()	创建 CircleDef 对象	构造器
public float radius	圆的半径	属性
public Vec2 localPosition	圆心坐标	属性

7. PolygonDef 类（多边形描述）

该类属于 org.jbox2d.collision 包，继承自 ShapeDef 类，其对象用于存储多边形的顶点信息。要特别注意的是，这里的多边形指的是凸多边形，并且在卷绕凸多边形顶点时，顺序是顺时针的。基本情况如图 10-15 所示。



▲图 10-15　凹凸多边形及卷绕顺序

该类的构造器与主要方法如表 10-8 所列。

表 10-8　　PolygonDef 类构造器与常用方法

构造器及方法签名	含　义	类　型
public PolygonDef()	创建 PolygonDef 对象	构造器
public void addVertex(Vec2 v)	为该多边形添加一个顶点	方法
public void clearVertices()	移除所有的顶点	方法
public int getVertexCount()	得到顶点的数量	方法

续表

构造器及方法签名	含　义	类　型
public java.util.List<Vec2> getVertexList()	得到所有顶点的集合	方法
public void setAsBox(float hx,float hy)	把多边形设定为一个矩形的形状 hx — 宽度的一半 hy — 高度的一半	方法

8. Body 类（刚体）

该类属于 org.jbox2d.dynamics 包，继承自 java.lang.Object 类，其对象表示一个刚体。在创建该类的对象时，不可以直接从构造器创建，必须通过 World 类对象的.createBody(BodyDef bd)方法创建。然后调用其 createShape(ShapeDef sd)方法创建刚体对应的几何形状，再调用 setMassFromShapes()方法计算出该刚体的质量。

该类常用的方法如表 10-9 所列。

表 10-9　Body 类的常用方法

方法签名	含　义
public Shape createShape(ShapeDef def)	创建一个几何形状，并将其添加进刚体中
public void destroyShape(Shape s)	销毁一个几何形状
public float getAngle()	获得角度（单位为弧度）
public float getAngularVelocity()	获得角速度
public Vec2 getLinearVelocityFromWorldPoint (Vec2 worldPoint)	获得此刚体质心的平移速度
public float getMass()	获得刚体的质量
public Body getNext()	获得列表中的下一个刚体
public Vec2 getPosition()	获得刚体定位点的位置
public Shape getShapeList()	获得刚体包含的几何形状的列表
public World getWorld()	获得刚体所属的世界对象
public Vec2 getWorldCenter()	获得刚体质心在世界坐标系的位置
public boolean isSleeping()	判断该刚体是不是活动的
public boolean isStatic()	判断该刚体是不是静止的
public void setMassFromShapes()	根据密度及几何形状自动计算该刚体的质量

本小节对 JBox2D 中的一些常用类的基本原理与 API 进行了简要的介绍，详细的使用将在后继章节的具体案例中一一给出。

10.3 球体撞击木块金字塔案例

上一小节主要介绍了 JBox2D 的一些基本概念，但是只讲概念就相当于纸上谈兵——不切实际。

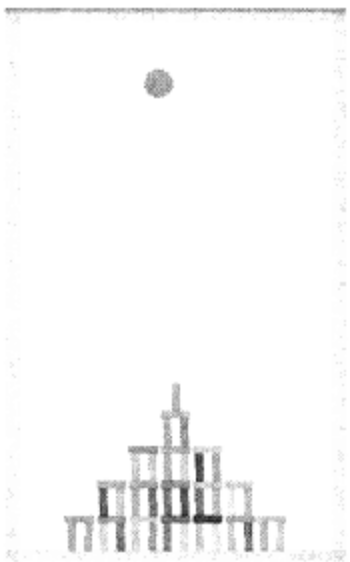
而且考虑到大篇幅的概念介绍只会使得本章更显得枯燥，因此特意准备了几个小案例，以供参考。首先给出一个钢球撞击木块金字塔的案例。

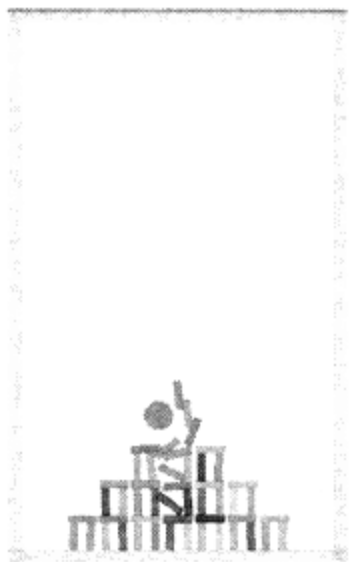
提示

由于本案例开发时使用了第三方的物理引擎 JBox2D，因此需要在项目中添加相应的支持 jar 包“jbox2d-2.0.1-library-only.jar”。如果读者不清楚如何在 Eclipse 中为项目添加 jar 包，可以查阅相关资料，本书篇幅所限不再赘述。没有 jar 包的读者可以到 JBox2D 的官方网站“www.jbox2d.org”下载并解压即可。

10.3.1 案例运行效果

本小节给出的是一个钢球撞击木块金字塔的案例。案例一旦开始运行，从屏幕上方有一只球按照一定的初速度加速下落，并撞击下方的金字塔。组成金字塔的木块被撞飞，其效果如图 10-16、图 10-17、图 10-18 及图 10-19 所示。


▲图 10-16　案例开始运行


▲图 10-17　开始碰撞

▲图 10-18　碰撞至中间

▲图 10-19　碰撞至最后

说明

图 10-15 表示的是钢球开始撞击木块金字塔，图 10-16 为钢球继续向下碰撞时的效果图，图 10-17 为钢球碰撞至木块金字塔中间的效果图，图 10-18 为钢球碰撞至木块金字塔底层的效果图。另外，请读者注意，由于某些自制 Android 系统的 2D 绘制问题，案例运行时可能会出现砖块及其他形状空心的现象，这并不影响程序的正确性，读者可以刷机或修改绘图方式来解决。

10.3.2 案例的基本框架结构

介绍本案例之前，首先需要介绍本案例的框架结构，理解本案例的框架结构有助于读者对本案例的学习。本案例的框架结构如图 10-20 所示。

- MyBox2dActivity 类为本案例的主控制类。案例开始运行时首先调用此类中的 onCreate(Bundle savedInstanceState)方法，在该方法中主要是生成 MyRectColor 类与 MyCircleColor 类的对象，并存入到 ArrayList<MyBody>集合中。最后跳转至 GameView 界面。
- Constant 类为本案例的常量类。该类主要是声明屏幕与现实世界的比例、线程标志位、模

拟频率及迭代次数，屏幕宽度以及屏幕高度。

▲图 10-20　案例基本框架结构

- MyBody 类为自定义的抽象类，是所有自定义刚体的父类。此类中的成员变量为刚体（Body）与颜色，同时自定义了绘制图形的抽象方法。
- MyCircleColor 类为自定义的圆形刚体类。此类继承自 MyBody 类，首先通过构造器初始化相应的成员变量，然后重写 MyBody 类中自定义的抽象方法。
- MyRectColor 类为自定义的矩形刚体类。此类也是继承自 MyBody 类，与 MyCircleColor 类相似。首先通过构造器初始化相应的成员变量，然后重写 MyBody 类中自定义的抽象方法。
- Box2DUtil 类为生成 MyCircleColor 类与 MyRectColor 类对象的工具类。该类的主要作用是创建圆形或者矩形刚体，然后创建 MyCircleColor 类与 MyRectColor 类对象，并返回。
- ColorUtil 类是自定义的颜色工具类。其主要作用是在固定的颜色二维数组中，随机地选取其中的一组，作为其中某一个刚体的颜色。
- GameView 类是案例要显示的界面。此类主要是创建画笔、获取画布以及开启刷帧线程，并绘制案例的场景。
- DrawThread 类为本案例的线程类。主要是不断地刷新画布使得游戏场景时时更新。同时为了使 JBox2D 物理引擎起到实质作用，需要调用 step(float dt,int iterations)方法。

10.3.3　常量类——Constant

该类为本案例中的常量类，主要作用是声明本案例中使用到的常量与变量，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 Constant.java。

```
1    package com.bn.box2d.bheap;                                    //声明包名
2    public class Constant{
3         public static final float RATE = 10;                      //屏幕与现实世界的比例
4         public static final boolean DRAW_THREAD_FLAG=true;       //绘制线程工作标志位
5         public static final float TIME_STEP = 2.0f/60.0f;        //模拟的频率
6         public static final int ITERA = 10;                      //迭代次数
7         public static int SCREEN_WIDTH;                          //屏幕宽度
8         public static int SCREEN_HEIGHT;                         //屏幕高度
9    }
```

说明　该类主要声明的常量有 RATE（屏幕与现实世界的比例）、DRAW_THREAD_FLAG（线程标志位）、TIME_STEP（模拟频率）及 ITERA（迭代次数），变量有 SCREEN_WIDTH（屏幕宽度）及 SCREEN_HEIGH（屏幕高度）。

10.3.4 抽象类——MyBody

该类为自定义的抽象类，是所有自定义刚体类的基类。由于 JBox2D 中的刚体类（Body）对象仅具有物理仿真计算的功能，并没有提供 Android 平台下的绘制功能，直接使用不是很方便。因此，这里定义了 MyBody 类对自定义刚体的绘制及 JBox2D 物理仿真对象（Body 类对象）进行了封装，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 MyBody.java。

```
1    package com.bn.box2d.bheap;                                    //声明包名
2    import org.jbox2d.dynamics.Body;                               //导入相关包
3    import android.graphics.Canvas;
4    import android.graphics.Paint;                                 //导入相关包
5    public abstract class MyBody{                                  //自定义刚体根类
6         Body body;                                                //JBox2D 物理引擎中的刚体
7         int color;                                                        //刚体的颜色
8         public abstract void drawSelf(Canvas canvas,Paint paint);         //绘制的方法
9    }
```

提示　读者可能发现此类非常简单，只有两个成员变量及一个抽象方法。这是因为抽象基类中一般只有描述所有子类共性的内容，因此，一般都不复杂，具体不同的业务逻辑留待非抽象子类根据各自的需要实现。

10.3.5 圆形刚体类——MyCircleColor

上一小节主要介绍的是自定义的刚体抽象类 MyBody，在本小节中主要介绍的是其子类 MyCircleColor。该类的主要作用是对圆形刚体的功能进行封装，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 MyCircleColor.java。

```
1    package com.bn.box2d.bheap;                                    //声明包名
2    import org.jbox2d.dynamics.Body;                               //导入相关包
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.bheap.Constant.*;                   //静态导入
5    public class MyCircleColor extends MyBody{                     //自定义的圆形类
6         float radius;                                             //圆形半径
7         public MyCircleColor(Body body,float radius,int color){   //构造器
8              this.body=body;                                      //初始化成员变量
9              this.radius=radius;                                  //初始化成员变量
10             this.color=color;                                    //初始化成员变量
11        }
12        public void drawSelf(Canvas canvas,Paint paint){          //绘制方法
13             paint.setColor(color&0x8CFFFFFF);                    //设置颜色
14             float x=body.getPosition().x*RATE;                   //设置圆心 X 坐标
15             float y=body.getPosition().y*RATE;                   //设置圆心 Y 坐标
16             canvas.drawCircle(x, y, radius, paint);              //绘制圆
17             paint.setStyle(Paint.Style.STROKE);                  //设置画笔样式
18             paint.setStrokeWidth(1);                             //设置画笔宽度
19             paint.setColor(color);                               //设置颜色
20             canvas.drawCircle(x, y, radius, paint);              //绘制边
21             paint.reset();                                       //恢复画笔设置
22        }
```

```
23    }
```

- 第 7-11 行为此类的构造器，其主要作用是初始化相关成员变量。
- 第 12-22 行为此类的绘制逻辑。首先设置颜色，并计算出圆心位置，然后绘制圆形，接着设置画笔的样式和颜色为圆形绘制边框。最后恢复画笔的设置，以免影响后继的绘制。

> **提示**　由于 MyRectColor 类与该类结构相同，同样是继承自 MyBody 类，只是作用为对矩形刚体的封装。因此，在本书中就不再对其进行赘述，读者可自行查阅随书光盘中本案例的代码。

10.3.6　生成刚体形状的工具类——Box2DUtil

上一小节中介绍了自定义的圆形刚体类，可以看出其中有一个成员变量 body 是 JBox2D 中的刚体（Body）。由于创建 Body 的代码并不十分简单，因此，为了使用方便，本案例中开发了一个工具类 Box2Dutil，它负责提供两个工厂方法，分别接收参数生成 MyCircleColor 类与 MyRectColor 类的对象。生成 MyCircleColor 类与 MyRectColor 类对象的方法分别为 createCircle 与 createBox，其具体代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 Box2DUtil.java。

```
1    package com.bn.box2d.bheap;                                    //声明包
2    import org.jbox2d.collision.CircleDef;                         //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.bheap.Constant.*;                   //静态导入
5    public class Box2DUtil{                                        //生成刚体形状的工具类
6         public static MyRectColor createBox(                      //创建矩形物体(颜色)
7              float x,                                             //x 坐标
8              float y,                                             //y 坐标
9              float halfWidth,                                     //半宽
10             float halfHeight,                                    //半高
11             boolean isStatic,                                    //是否为静止的
12             World world,                                         //世界
13             int color)                                           //颜色
14             {
15             PolygonDef shape = new PolygonDef();                 //创建多边形描述对象
16             if(isStatic){                                     //判断isStatic是否为true
17                  shape.density = 0;                           //密度为 0
18             }else{
19                  shape.density = 1.0f;                        //密度为 1.0f
20             }
21             shape.friction = 0.0f;                            //设置摩擦系数
22             shape.restitution = 0.6f;                         //设置能量损失率
23             shape.setAsBox(halfWidth/RATE, halfHeight/RATE);
24             BodyDef bodyDef = new BodyDef();                     //创建刚体描述对象
25             bodyDef.position.set(x/RATE, y/RATE);                //设置位置
26             Body bodyTemp= world.createBody(bodyDef);            //在世界中创建刚体
27             bodyTemp.createShape(shape);                         //指定刚体形状
28             bodyTemp.setMassFromShapes();                        //设置物体质量
29             return new MyRectColor(bodyTemp,halfWidth,halfHeight,color);
30        }
31        public static MyCircleColor createCircle(                 //创建圆形（颜色）
32             float x,                                             //x 坐标
33             float y,                                             //y 坐标
```

```
34                  float radius,                                          //半径
35                  World world,                                           //世界
36                  int color){                                            //颜色
37                  CircleDef shape = new CircleDef();                     //创建圆描述对象
38                  shape.density = 2;                                     //设置密度
39                  shape.friction = 0.0f;                                 //设置摩擦系数
40                  shape.restitution = 0.95f;                             //设置能量损失率
41                  shape.radius = radius/RATE;                            //设置半径
42                  BodyDef bodyDef = new BodyDef();                       //创建刚体描述对象
43                  bodyDef.position.set(x/RATE, y/RATE);                  //设置位置
44                  Body bodyTemp = world.createBody(bodyDef);             //在世界中创建刚体
45                  bodyTemp.createShape(shape);                           //指定刚体形状
46                  bodyTemp.setMassFromShapes();
47                  return new MyCircleColor(bodyTemp,radius,color);  //返回 MyCircleColor 对象
48              }
49      }
```

- 第 7-14 行是 createBox 方法的参数，主要有矩形的中心点坐标、矩形的半宽和半高、是否静止标志、刚体所属 World 类对象及颜色。
- 第 15-23 行为创建多边形描述对象，同时设置密度、摩擦系数以及能量损失率，并创建包围盒。
- 第 24-25 行为创建刚体描述对象并设置其具体位置。
- 第 26-28 行为创建 Body（刚体）类对象的具体方法。在创建刚体对象时不可以直接使用构造器，必须通过 createBody（BodyDef bd）创建 Body 对象，然后调用 createShape（ShapeDef sd）为刚体指定形状，最后需要通过调用 setMassFromShapes()方法为刚体指定质量。

提示

第 31-48 行为创建圆形刚体的方法，该方法与 6-30 行的创建矩形刚体相似。在这里就不再对该方法进行赘述，读者在看该方法时可以参考创建矩形刚体的方法。

10.3.7 颜色工具类——ColorUtil

细心的读者会发现，本案例每次运行后，小球和木块的颜色与上一次运行均不相同。这种效果是基于本小节将要介绍的颜色工具类——ColorUtil 来实现的，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 ColorUtil.java。

```
1     package com.bn.box2d.bheap;                                  //声明包
2     public class ColorUtil{                                      //创建颜色类
3        static int[][] result={                                   //创建颜色数组
4              ......//该处省略了该数组的具体数据，读者可自行查阅随书光盘中源代码
5        };
6        public static int getColor(int index){                    //创建静态获得颜色的方法
7              int[] rgb=result[index%result.length];              //创建 int 类型的数组
8              int result=0xff000000;                              //声明常量
9              result=result|(rgb[0]<<16);                         //红色色彩通道左移 16 位并入结果
10             result=result|(rgb[1]<<8);                          //绿色色彩通道左移 8 位并入结果
11             result=result|(rgb[2]);                             //蓝色色彩通道并入结果
12             return result;                                      //返回结果
13    } }
```

- 第 3-5 行表示的是一个二维数组，该数组中存储了许多组自定义颜色的（RGB）值。

- 第 6-13 行为获取颜色的方法，调用者传入颜色索引号 index，此方法将 RGB 3 个色彩通道的值组装成 ARGB 顺序 32 比特颜色整数返回。

10.3.8　主控制类——MyBox2dActivity

上一小节介绍的是颜色工具类，在本节中主要介绍本案例的主控制类的框架。此类的主要任务为，初始化整个物理世界中的各个物体（包括球和组成金字塔的所有木块），并将物体放入列表中，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 MyBox2dActivity.java。

```
1    package com.bn.box2d.bheap;                                  //声明包
2    import java.util.ArrayList;                                  //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.bheap.Constant.*;                 //静态导入
5    public class MyBox2dActivity extends Activity{               //创建继承 Activity 的类
6        AABB worldAABB;                                          //创建 一个管理碰撞的世界
7        World world;                                             //成员变量 world
8        Random random=new Random();                              //创建生成伪随机数的类
9        ArrayList<MyBody> bl=new ArrayList<MyBody>();            //存储物体列表的集合
10       public void onCreate(Bundle savedInstanceState){         //继承 Activity 类需要重写的方法
11           super.onCreate(savedInstanceState);                  //调用父类
12           requestWindowFeature(Window.FEATURE_NO_TITLE);//设置为全屏
13           getWindow().setFlags(WindowManager.LayoutParams. FLAG_FULLSCREEN ,
14           WindowManager.LayoutParams. FLAG_FULLSCREEN);
15             setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
                                                                  //设置为横屏模式
16           DisplayMetrics dm=new DisplayMetrics();              //获取屏幕尺寸
17           getWindowManager().getDefaultDisplay().getMetrics(dm);
                                                                  //得到屏幕的宽度和高度
18           if(dm.widthPixels<dm.heightPixels){                  //判断宽度与高度的大小
19                   SCREEN_WIDTH=dm.widthPixels;                 //为 SCREEN_WIDTH 赋值
20                   SCREEN_HEIGHT=dm.heightPixels;               //为 SCREEN_HEIGHT 赋值
21           }else{
22                   SCREEN_WIDTH=dm.heightPixels;                //为 SCREEN_WIDTH 赋值
23                   SCREEN_HEIGHT=dm.widthPixels;                //为 SCREEN_HEIGHT 赋值
24           }
25           worldAABB = new AABB();                              //创建 AABB 对象
26           worldAABB.lowerBound.set(-100.0f,-100.0f);           //设置包围盒子下边界
27           worldAABB.upperBound.set(100.0f, 100.0f);            //设置包围盒子上边界
28           Vec2 gravity = new Vec2(0.0f,10.0f);                 //设置重力加速度
29           boolean doSleep = true;                              //设置 doSleep 为 true
30           world = new World(worldAABB, gravity, doSleep);          //创建世界
31           final int kd=40;                                         //宽度或高度
32           MyRectColor mrc=Box2DUtil.createBox                      //创建 MyRectColor 类对象
33                              (kd/4,SCREEN_HEIGHT/2,kd/4,
34                               SCREEN_HEIGHT/2,true,world,0xFFe6e4FF);
35           bl.add(mrc);                                             //添加进集合中
36           ......//该处省略了创建其他 3 个边界的代码，读者可以查阅随书光盘中源代码
37           final int bs=20;                                         //砖块间距
38           final int bw=(int)((SCREEN_WIDTH-2*kd-11*bs)/18);        //砖块宽度
39           ......//该处省略了创建木块金字塔的代码，将在后面给出
40           MyCircleColor ball=Box2DUtil.createCircle                //创建球
41                                  (SCREEN_WIDTH/2-24, kd, kd/2, world,
42                                   ColorUtil.getColor(Math.abs(random.nextInt())));
43           bl.add(ball);                                            //添加进集合中
44           ball.body.setLinearVelocity(new Vec2(0,50));             //设置初速度
```

```
45          GameView gv= new GameView(this);                    //创建 GameView 对象
46          setContentView(gv);                                 //跳转到 GameView 界面
47      }
48  }
```

- 第 12-15 行表示的是设置本案例为全屏模式，并强制为横屏。
- 第 16-24 行表示的是获得屏幕的尺寸，并根据屏幕的宽度和高度为 SCREEN_WIDTH 与 SCREEN_HEIGHT 赋值。其中 SCREEN_WIDTH 永远为最大值。
- 第 25-27 行为创建 AABB 对象，并设置其包围盒子的范围，该范围一般应该要比物体所在区域要大。
- 第 28-30 行为创建重力加速度向量，设置 doSleep 标志位为 true，并创建 World 类的对象。
- 第 40-44 行是通过调用 Box2DUtil 类中的 createCircle 方法获取与 MyCircleColor 类的对象，并将其存入之 ArrayList<MyBody>集合中，然后设置球体的初速度。
- 第 45-46 行表示的是获得 GameView 类的对象，并跳转至 GameView 界面。

开发完了主控制类 MyBox2dActivity 类框架之后，下面就应该开发木块金字塔。由于木块金字塔中的木块很多，并且有一定规律，因此用固定的代码一一直接生成很不合算。笔者这里根据木块摆放的规律用了一个二层 for 循环实现，将下列代码插入到 MyBox2dActivity 框架的第 39 行。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 MyBox2dActivity.java。

```
1     for(int i=2;i<10;i++){                                    //for 循环
2         if((i%2)==0){                                         //木块竖直情况
3             for(int j=0;j<9-i;j++){                           //竖着的左侧木块
4                 mrc=Box2DUtil.createBox(                      //调用 createBox 方法
5                     kd/2+bs+bw/2+i*(kd+5)/2+j*(kd+5)+3,       //该矩形的 X 坐标
6                     SCREEN_HEIGHT+bw-i*(bw+kd)/2,             //该矩形的 Y 坐标
7                     bw/2,                                     //矩形半宽
8                     kd/2,                                     //矩形半高
9                     false,                                    //设置 isStatic 为 false
10                    world,                                    //World 类对象
11                    ColorUtil.getColor(Math.abs(random.nextInt()))     //调用颜色工具类
12                );
13                bl.add(mrc);                                  //MyRectColor 对象存入集合
14            }
15            for(int j=0;j<9-i;j++){                           //竖着的右侧木块
16                mrc=Box2DUtil.createBox(                      //调用 createBox 方法
17                    3*kd/2+bs-bw/2+i*(kd+5)/2+j*(kd+5)-3,     //该矩形的 X 坐标
18                    SCREEN_HEIGHT+bw-i*(bw+kd)/2,             //该矩形的 Y 坐标
19                    bw/2,                                     //矩形的半宽
20                    kd/2,                                     //矩形半高
21                    false,                                    //isStatic 为 false
22                    world,                                    //World 类对象
23                    ColorUtil.getColor(Math.abs(random.nextInt()))     //调用颜色工具类
24                );
25                bl.add(mrc);                                  // MyRectColor 对象存入集合
26        } }
27        if((i%2)!=0){                                         //木块横向情况
28            for(int j=0;j<10-i;j++){                          //for 循环
29                mrc=Box2DUtil.createBox(                      //调用 createBox 方法
30                    kd/2+bs+kd/2+(i-1)*(kd+5)/2+j*(kd+5),         //矩形的 X 坐标
31                    SCREEN_HEIGHT-(kd-bw)/2-(i-1)*(bw+kd)/2,      //矩形的 Y 坐标
32                    kd/2,                                         //矩形半宽
33                    bw/2,                                         //矩形半高
```

```
34                  false,                                          //isStatic 为 false
35                  world,                                          //World 类对象
36                  ColorUtil.getColor(Math.abs(random.nextInt()))      //调用颜色工具类
37              );
38              bl.add(mrc);                                    // MyRectColor 对象存入集合
39    } } }
```

- 第 2-26 行为木块金字塔中竖直的木块，通过调用 Box2DUtil 类中的 createBox 方法获得 MyRectColor 类的对象，并将该对象存入 ArrayList<MyBody>集合中。
- 第 27-39 行为木块金字塔中水平放置的木块，通过调用 Box2DUtil 类中的 createBox 方法获得 MyRectColor 类的对象，并将该对象存入 ArrayList<MyBody>集合中。

10.3.9　显示界面类——GameView

上一小节中最后要跳转的目标是 GameView 界面，该界面功能为对本案例中的场景进行渲染。该类需要继承 Android 系统中的 SurfaceView 类，并实现 SurfaceHolder.Callback 接口，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 GameView.java。

```
1     package com.bn.box2d.bheap;                                   //声明包
2     import android.graphics.Canvas;                               //导入相关类
3     ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4     import android.view.SurfaceView;                              //导入相关类
5     public class GameView extends SurfaceView
6     implements SurfaceHolder.Callback{                            //创建类
7          MyBox2dActivity activity;                                //成员变量
8          Paint paint;                                             //画笔
9          DrawThread dt;                                           //绘制线程
10         public GameView(MyBox2dActivity activity){               //构造器
11              super(activity);                                    //调用父类
12              this.activity = activity;                           //初始化成员变量
13              this.getHolder().addCallback(this);        //设置生命周期回调接口的实现者
14              paint = new Paint();                                //创建画笔
15              paint.setAntiAlias(true);                           //打开抗锯齿
16              dt=new DrawThread(this);                            //创建线程对象
17              dt.start();                                         //开启线程
18         }
19         public void onDraw(Canvas canvas){                       //绘制方法
20              if(canvas==null){                                   //判断 canvas 是否为空
21                   return;                                        //canvas 为空则返回
22              }
23              canvas.drawARGB(255,255, 255, 255);                 //设置背景颜色
24              for(MyBody mb:activity.bl){                         //绘制场景中的物体
25                   mb.drawSelf(canvas, paint);                    //绘制球与木块金字塔
26         } }
27         public void surfaceChanged(SurfaceHolder arg0,int arg1,int arg2,int arg3){ }
28         public void surfaceCreated(SurfaceHolder holder){ repaint(); }
                                                                    //创建时被调用
29         public void surfaceDestroyed(SurfaceHolder arg0){ }      //销毁时被调用
30         public void repaint(){
31              SurfaceHolder holder=this.getHolder();              //得到回调接口的对象
32              Canvas canvas = holder.lockCanvas();                //获取画布
33              try{
34                   synchronized(holder){                          //同步处理
```

```
35                          onDraw(canvas);                          //绘制
36                  } }catch(Exception e){                           //捕获异常
37                      e.printStackTrace();                         //打印堆栈信息
38                  }finally{
39                      if(canvas != null){                          //判断 canvas 是否为空
40                          holder.unlockCanvasAndPost(canvas);      //解锁
41      } } } }
```

- 第 10-18 行为该类的构造器，在该类的构造器中主要是初始化成员变量，同时创建画笔，打开抗锯齿，创建 DrawThread 类对象，并开启线程。
- 第 19-26 行为绘制方法。首先判断 canvas 是否为空，如果为空，则返回。然后设置背景颜色为白色。最后再绘制本案例中的球以及木块金字塔。
- 第 27-29 行为创建实现回调接口的类必须重写的 3 个方法。
- 第 30-41 行为刷帧方法，首先获得 SurfaceHolder 的对象，并获取画布，然后使用同步控制绘制方法，最后如果画布不为空，则需要为画布解锁。

10.3.10 绘制线程类——DrawThread

上一小节介绍的是本案例中的场景渲染，在本小节中将为读者介绍如何使 JBox2D 的物理引擎动起来。此任务主要由一个线程来定时完成，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBHeap/src/com.bn.box2d.bheap 目录下的 DrawThread.java。

```
1     package com.bn.box2d.bheap;                                    //声明包
2     import static com.bn.box2d.bheap.Constant.*;                   //静态导入
3     public class DrawThread extends Thread{                        //绘制线程
4         GameView gv;                                               //GameView 对象
5         public DrawThread(GameView gv){                            //构造器
6             this.gv=gv;                                            //初始化成员变量
7         }
8         @Override
9         public void run(){                                       //重写的 run()方法
10            while(DRAW_THREAD_FLAG){                             //判断标志位是否为 true
11                gv.activity.world.step(TIME_STEP, ITERA);        //开始模拟
12                gv.repaint();                                    //刷帧
13                try {
14                    Thread.sleep(20);                            //线程休息 20ms
15                }catch (InterruptedException e){                 //捕获异常
16                    e.printStackTrace();                         //打印堆栈信息
17    } } } }
```

- 第 5-7 行为本类的构造器，其主要作用是初始化相应成员变量。
- 第 9-17 行为继承 Thread 类必须重写的 run()方法，在该方法中需要调用 step(float dt,int iterations)方法使得 JBox2D 开始模拟，并不断地定时刷帧更新界面。

提示　本案例中使用 JBox2D 引擎进行物理模拟时为了方便，将定时模拟与定时刷帧的任务写到了一个线程中。如果读者以后有需要可以开辟独立的线程完成不同的任务，但要注意好线程间的同步问题。

10.4 简易打砖块案例

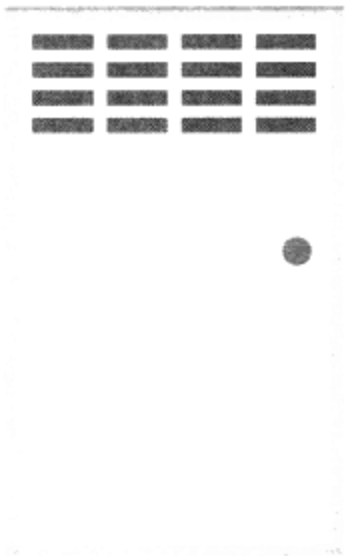
上一节中介绍了球体撞击木块金字塔的案例，在本节中将介绍简易打砖块案例的开发。该案例最终需要的效果是砖块碰撞后消失。首先开发出类似于球体撞击木块金字塔的一般碰撞版，然后再在此基础上开发最终的碰撞后消失版。

10.4.1　一般碰撞版

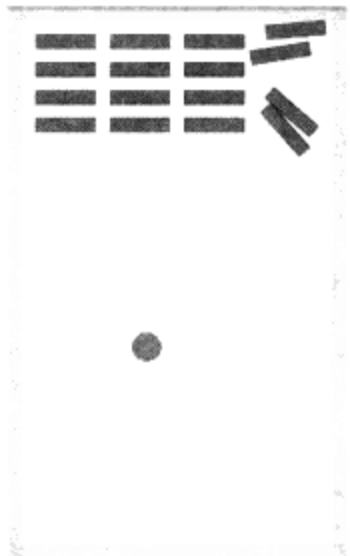
一般碰撞版与球体撞击木块金字塔案例类似，只是改变了木块摆放的位置，并将世界的重力加速度设置为 0，同时为小球设置一个斜向上的初速度。

1. 案例运行效果

本小节给出的是打砖块的一般碰撞版，案例一旦运行，会从屏幕下方发射具有斜向上初速度的小球，撞击砖块，砖块四散，但是并不消失。其效果如图 10-21、图 10-22、图 10-23 和图 10-24 所示。



▲图 10-21　案例开始运行



▲图 10-22　碰撞 5 秒

▲图 10-23　碰撞 10 秒

▲图 10-24　碰撞 20 秒

说明　图 10-21 为一般碰撞版开始运行，图 10-22 为碰撞已经持续 5 秒，图 10-23 为小球碰撞 10 秒后，图 10-24 为碰撞已经持续了 20 秒。

2. 主控制类——MyBox2dActivity

由于本案例中其他类与球体撞击木块金字塔基本一致，因此本小节中只介绍改变小球与木块基本设置的主控制类，该类继承自 Android 系统中的 Activity 类，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBlock/src/com.bn.box2d.block 目录下的 MyBox2dActivity.java。

```
1    package com.bn.box2d.block;                                    //声明包名
2    import java.util.ArrayList;                                    //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
```

```
4    import static com.bn.box2d.block.Constant.*;                    //静态导入
5    public class MyBox2dActivity extends Activity{                  //创建继承 Activity 的类
6        AABB worldAABB;                                             //创建一个管理碰撞的世界
7        World world;
8        ArrayList<MyBody> bl=new ArrayList<MyBody>();               //物体列表
9        public void onCreate(Bundle savedInstanceState){  //继承 Activity 类需要重写的方法
10           super.onCreate(savedInstanceState);                 //调用父类
11           requestWindowFeature(Window.FEATURE_NO_TITLE);      //设置为全屏
12           getWindow().setFlags(WindowManager.LayoutParams. FLAG_FULLSCREEN ,
13           WindowManager.LayoutParams. FLAG_FULLSCREEN);
14             setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
                                                                     //设置为横屏模式
15           DisplayMetrics dm=new DisplayMetrics();                 //创建 DisplayMetrics 对象
16           getWindowManager().getDefaultDisplay().getMetrics(dm);      //获取屏幕尺寸
17           if(dm.widthPixels<dm.heightPixels){                     //判断屏幕宽度与高度的大小
18                   SCREEN_WIDTH=dm.widthPixels;                    //为 SCREEN_WIDTH 赋值
19                   SCREEN_HEIGHT=dm.heightPixels;                  //为 SCREEN_HEIGHT 赋值
20           }else{
21                   SCREEN_WIDTH=dm.heightPixels;                   //为 SCREEN_WIDTH 赋值
22                   SCREEN_HEIGHT=dm.widthPixels;                   //为 SCREEN_HEIGHT 赋值
23           }
24           worldAABB = new AABB();                                 //创建 AABB 对象
25           worldAABB.lowerBound.set(-100.0f,-100.0f);              //设置包围盒子下边界
26           worldAABB.upperBound.set(100.0f, 100.0f);               //设置包围盒子上边界
27           Vec2 gravity = new Vec2(0.0f,0.0f);                     //设置重力加速度
28           boolean doSleep = true;                                 //设置 doSleep 为 true
29           world = new World(worldAABB, gravity, doSleep);         //创建世界
30           final int kd=20;                                        //宽度或高度
31           MyRectColor mrc=Box2DUtil.createBox                     //创建左侧边界
32                           (kd/2,SCREEN_HEIGHT/2,
d/2,SCREEN_HEIGHT/2,true,world,Color.YELLOW);
33           bl.add(mrc);                                        //将 MyRectColor 对象存入集合
34           ......//该处省略了创建其他三个边界的代码，读者可以查阅随书光盘中源代码
35           final int bs=20;                                    //砖块间距
36           final int bw=(SCREEN_WIDTH-2*kd-5*bs)/4;            //砖块宽度
37           for(int i=0;i<4;i++){                               //外层 for 循环
38               for(int j=0;j<4;j++){                           //里层 for 循环
39                   mrc=Box2DUtil.createBox(                    //调用 createBox 方法
40                               kd+bs+bw/2+j*(bw+bs),           //矩形中心 X 坐标
41                               kd+bs+kd/2+i*(kd+bs),           //矩形中心 Y 坐标
42                               bw/2, kd/2,                     //半宽与半高
43                               false,
44                               world,                          //World 类对象
45                               Color.RED                       //颜色为红色
46                   );
47             bl.add(mrc);                                      //将 MyRectColor 对象存入集合
48           } }
49           MyCircleColor ball=Box2DUtil.createCircle           //调用 createCircle 方法
50                                   (SCREEN_WIDTH/2,SCREEN_HEIGHT-100,kd,world,Color.
                                     MAGENTA);
51           bl.add(ball);                                       //将 MyCircleColor 对象存入集合
52           ball.body.setLinearVelocity(new Vec2(5,-40));       //设置小球的初速度
53           GameView gv= new GameView(this);                    //创建 GameView 对象
54           setContentView(gv);                                 //跳转界面
55   } }
```

- 第 6-8 行为本类的成员变量，主要是声明 AABB 及 World 的引用，同时创建 ArrayList<MyBody>的集合，该集合中存放 MyBody 及其子类的对象。
- 第 11-14 行为设置本案例为全屏模式，并强制为横屏。

- 第 15-23 行为获得屏幕尺寸，并根据屏幕的宽度、高度的大小为 SCREEN_WIDTH 以及 SCREEN_HEIGHT 赋值，使得 SCREEN_HEIGHT 总是最大值。
- 第 24-26 行为创建 AABB 对象，同时设置包围盒子的范围，该范围一般比物体所在区域要大。
- 第 27-29 行为创建 Vec2 向量，设置 doSleep 为 true，同时创建 World 类的对象。
- 第 30-34 行为设置木块的宽度与高度，并创建四周的边界。
- 第 35-48 行为设置砖块间距，计算出砖块宽度，使用嵌套的 for 循环创建 MyRectColor 类的对象充当砖块，并将得到的对象存入 ArrayList<MyBody>集合中。
- 第 49-51 行为调用 Box2DUtil 中的 createCircle 方法获得 MyCircleColor 类对象，并将该对象存入 ArrayList<MyBody>集合中。
- 第 52-54 行为设置小球的初速度，获得 GameView 类的对象，并跳转界面至 GameView 界面。

10.4.2　碰撞后消失版

最终的碰撞后消失版是基于一般碰撞版进行进一步开发完成的，因此，只需要将一般碰撞版中的所有的代码复制到本项目中，修改其包名为本项目的包名，然后修改相应类中的内容即可。

提示　由于碰撞后消失版中的大部分类与一般碰撞版中的类相同，所以在这里就只对改变的类进行介绍，其余相同的类就不再重复介绍，感兴趣的读者可以自行查看随书光盘中源代码。

1. 案例运行效果

本小节给出的是最终的碰撞后消失版，程序运行开始，小球具有一定的初速度，然后小球碰撞砖块后砖块消失，小球速度改变。其效果如图 10-25、图 10-26、图 10-27 及图 10-28 所示。

说明　图 10-25 为最终的碰撞后消失版开始运行，图 10-26 为小球撞击 5 块砖块后，图 10-27 为小球撞击 10 块砖块，图 10-28 为小球撞击砖块只剩余 3 块。

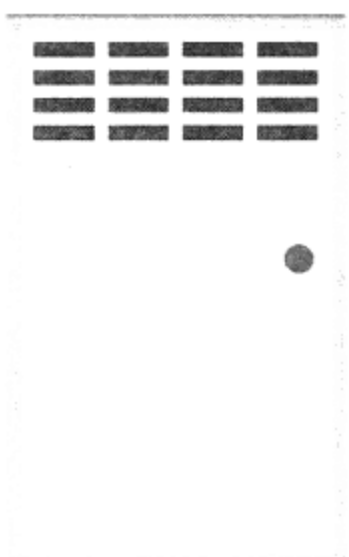
2. 矩形刚体类——MyRectColor

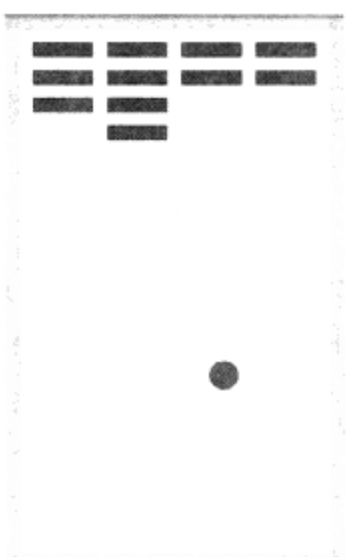
首先介绍的是修改后的 MyRectColor 类，该类为自定义抽象类 MyBody 类的子类。修改后增加了 3 个属性，并且修改了构造器与绘制方法，其代码如下。

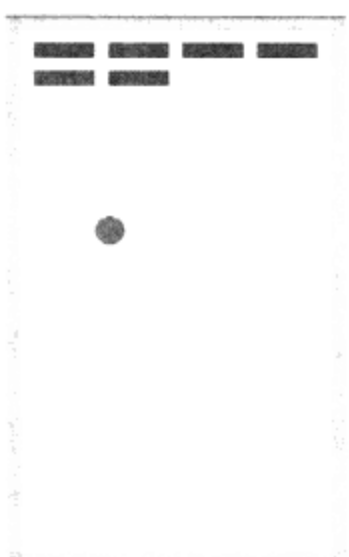
代码位置：见随书光盘中源代码/第 10 章/Box2DBlockDispose/src/com.bn.box2d.blockl 目录下的 MyRectColor.java。

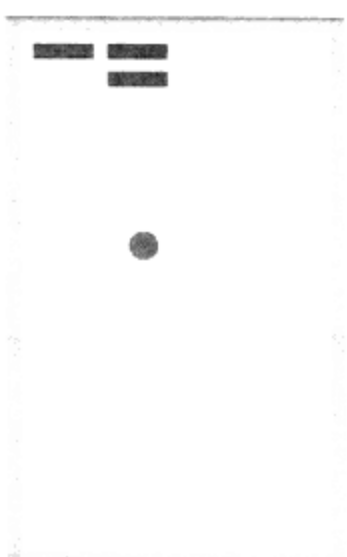
```
1    package com.bn.box2d.blockl;                                  //声明包名
2    import org.jbox2d.dynamics.Body;                              //导入相关包
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.blockl.Constant.*;                 //静态导入
5    public class MyRectColor extends MyBody{                      //自定义的矩形类(颜色)
6         float halfWidth;                                         //半宽
7         float halfHeight;                                        //半高
8         boolean isBlock;                                         //是否为砖块
9         boolean isLive=true;                                     //是否绘制
10        boolean isDeleted=false;                                 //是否被删除
```

```
11          public MyRectColor(Body body,float halfWidth,   //构造器
12                                  float halfHeight,int color,boolean isBlock){
13               this.body=body;                                //初始化 body 成员变量
14               this.halfWidth=halfWidth;                      //初始化 halfWidth 成员变量
15               this.halfHeight=halfHeight;                    //初始化 halfHeight 成员变量
16               this.color=color;                              //初始化 color 成员变量
17               this.isBlock=isBlock;                          //初始化 isBlock 成员变量
18          }
19          public void drawSelf(Canvas canvas,Paint paint){//重写的绘制方法
20               if(!isLive){return;}                           //如果 isLive 为 false 则返回
21               ......//该处省略了部分绘制代码，读者可以自行查阅随书光盘中源代码
22     } }
```


▲图 10-25　发射小球


▲图 10-26　撞击掉 5 块砖块


▲图 10-27　撞击掉 10 块砖块


▲图 10-28　剩余 3 块砖块

- 第 8-10 行为新添加的成员变量，有 isBlock（是否为砖块）、isLive（是否绘制）以及 isDeleted（是否被删除）。
- 第 17 行为本类的构造器中初始化新添加的成员变量 isBlock。
- 第 20 行为本类的绘制方法中新添加的条件，该行主要是判断 isLive 是否为 false。如果为 false，则直接返回不绘制。否则，继续向下执行进行绘制。

3. 生成刚体形状工具类——Box2DUtil

由于 Box2DUtil 类中的 createBox 方法，最后返回的是 MyRectColor 类的对象，所以在修改过 MyRectColor 类之后，需要对 Box2DUtil 工具类中的 createBox 方法进行修改，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBlockDispose/src/com.bn.box2d.blockl 目录下的 MyBox2dActivity.java。

```
package com.bn.box2d.blockl;                                   //声明包名
import org.jbox2d.collision.CircleDef;                         //导入相关包
......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
import static com.bn.box2d.block.Constant.*;                   //静态导入
public class Box2DUtil{                                        //生成物理形状的工具类

     public static MyRectColor createBox(                      //创建矩形物体(颜色)
          float x,                                             //x 坐标
          float y,                                             //y 坐标
          float halfWidth,                                     //半宽
          float halfHeight,                                    //半高
```

```
11              boolean isStatic,                                   //是否为静止的
12              World world,                                        //世界
13              int color,                                          //颜色
14              boolean isBlock                                     //是否为砖块
15        ){
16              ......//该处省略了创建矩形刚体方法的具体代码，读者可自行查阅随书光盘中源代码
17              return new MyRectColor(bodyTemp,halfWidth,halfHeight,color,isBlock);
18          }
19          public static MyCircleColor createCircle(               //创建圆形物体(颜色)
20              float x,                                            //x 坐标
21              float y,                                            //y 坐标
22              float radius,                                       //半径
23              World world,                                        //世界
24              int color                                           //颜色
25          ){
26              ......//该处省略了创建圆形刚体的方法的具体代码，读者可自行查阅随书光盘中源代码
27    } }
```

- 第 14 行为 createBox 方法中新添加的参数 isBlock，该参数代表这创建的矩形是否为砖块。
- 第 17 行为使用新参数创建 MyRectColor 矩形刚体的对象，并作为返回值返回。

4. 刚体查找工具类——BodySearchUtil

本案例中小球碰撞砖块，砖块会消失。而将 MyRectColor 类中的 isLive 标志位为 false 时，对应的砖块就不会再绘制，所以创建一个刚体查找工具类——BodySearchUtil，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBlockDispose/src/com.bn.box2d.blockl 目录下的 BodySearchUtil.java。

```
1     package com.bn.box2d.blockl;                                  //声明包
2     import java.util.ArrayList;                                   //导入相关类
3     import org.jbox2d.dynamics.Body;                              //导入相关类
4     public class BodySearchUtil{                                  //创建 BodySearchUtil 类
5          public static void doAction(Body body1,Body body2,ArrayList<MyBody> bl){
6               for(MyBody mpi:bl){                                 //增强 for 循环
7                    if(body1==mpi.body||body2==mpi.body){
8                         if(mpi instanceof MyRectColor){
                                                        //判断 mpi 是否属于 MyRectColor 类
9                              MyRectColor mrc=(MyRectColor)mpi;    //强制转换
10                             if(mrc.isBlock){                     //判断是否为砖块
11                                  mrc.isLive=false;               //设置 isLive 为 false
12    } } } } } }
```

- 第 8 行为判断 mpi 是否属于 MyRectColor 类，如果其属于该类，说明其为砖块或者是四周的墙壁。
- 第 10-12 行为判断其是否为砖块，如果为砖块，则设置 isLive 为 false。

5. 主控制类——MyBox2dActivity

主控制类 MyBox2dActivity 中，调用了 Box2DUtil 类中的 createBox 方法，并在 initContactListener()方法中创建触控监听器对象，同时在该内部类中调用了 BodySearchUtil 类中的 doAction 方法，其详细代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBlockDispose/src/com.bn.box2d.blockl 目录下的 MyBox2dActivity.java。

```
1    package com.bn.box2d.blockl;                                    //声明包名
2    import java.util.ArrayList;                                     //导入相关包
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.block.Constant.*;                    //导入相关包
5    public class MyBox2dActivity extends Activity{                  //创建继承 Activity 的类
6        AABB worldAABB;                                             //创建一个管理碰撞的 World
7        World world;                                                //创建 World 类的引用
8        ArrayList<MyBody> bl=new ArrayList<MyBody>();               //物体列表
9        public void onCreate(Bundle savedInstanceState){  //继承 Activity 类需要重写的方法
10           ......//该处省略了部分 onCreate 方法中的代码，读者可以自行查阅随书光盘的源代码
11           final int kd=20;                                   //宽度或高度
12           MyRectColor mrc=Box2DUtil.createBox   //调用 Box2DUtil 类中的 createBox 方法
13           (kd/2, SCREEN_HEIGHT/2, kd/2, SCREEN_HEIGHT/2, true,world,Color.YELLOW,
             false);
14           bl.add(mrc);                                  //得到的 MyRectColor 对象存入集合
15           ......//该处省略了创建其他 3 个边界的代码，读者可以查阅随书光盘中源代码
16           final int bs=20;                                        //砖块间距
17           final int bw=(SCREEN_WIDTH-2*kd-5*bs)/4;                //砖块的宽度
18           for(int i=0;i<4;i++){                                   //外层 for 循环
19               for(int j=0;j<4;j++){                               //里层 for 循环
20                   mrc=Box2DUtil.createBox(                        //调用 createBox 方法
21                           kd+bs+bw/2+j*(bw+bs),                   //矩形中心点 X 坐标
22                           kd+bs+kd/2+i*(kd+bs),                   //矩形中心点 Y 坐标
23                           bw/2, kd/2,                             //半宽、半高
24                           true,                                   //isStatic 为 true
25                           world,                                  //World 类的对象
26                           Color.RED,                              //颜色
27                           true                                    //是否为砖块标志位
28                   );
29               bl.add(mrc);                              //得到的 MyRectColor 对象存入集合
30           } }
31           MyCircleColor ball=Box2DUtil.createCircle
                                                           //调用 Box2DUtil 中的 createCircle 方法
32           (SCREEN_WIDTH/2, SCREEN_HEIGHT-100, kd, world,Color.MAGENTA);
33           bl.add(ball);                                 //得到的 MyCircleColor 对象存入集合
34           ball.body.setLinearVelocity(new Vec2(5,-40));           //设置质心的线速度
35           initContactListener();                                  //添加监听器
36           GameView gv= new GameView(this);                        //GameView 类的对象
37           setContentView(gv);                                     //跳转界面
38       }
39         public void initContactListener(){                        //加载碰撞监听器
40               ContactListener cl=new ContactListener(){           //内部类
41                   @Override
42                   public void add(ContactPoint arg0){             //重写的方法
43                       BodySearchUtil.doAction(arg0.shape1.getBody(),arg0.shape2.
                         getBody(),bl);
44                   }
45                   @Override
46                   public void persist(ContactPoint arg0){}  //重写 persist 方法
47                   @Override
48                   public void remove(ContactPoint arg0){}   //重写 remov 方法
49                   @Override
50                   public void result(ContactResult arg0){}  //重写 result 方法
51               };
52               world.setContactListener(cl);
53   } }
```

- 第 11-14 行为设置木块的宽度与高度，调用新的 createBox 方法创建四周的边界，并将得到

的 MyRectColor 类的对象存入 ArrayList<MyBody>集合中。

- 第 20-28 行是调用 Box2DUtil 类中新的 createBox 方法创建砖块。
- 第 35 行是调用自己新添加的 initContactListener()方法。
- 第 39-53 行为自己新添加的 initContactListener()方法，在该方法内部主要是使用内部类加载监听器，最后为 World 类的对象添加监听器。

6. 线程类——DrawThread

上一小节介绍了主控制类，在本小节中将为读者介绍线程类——DrawThread。在该线程类中定时使 JBox2D 的物理引擎动起来，调用 repaint()方法实现刷帧，然后判断 isLive 与 isDeleted 是否全部为 false，如果全为 false，则销毁对应的刚体，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2DBlockDispose/src/com.bn.box2d.blockl 目录下的 DrawThread.java。

```
1    package com.bn.box2d.blockl;                                  //声明包名
2    import static com.bn.box2d.blockl.Constant.*;                 //静态导入
3    public class DrawThread extends Thread{                       //绘制线程
4         GameView gv;                                             //GameView 的引用
5         public DrawThread(GameView gv){                          //构造器
6              this.gv=gv;                                         //初始化成员变量
7         }
8         @Override
9         public void run(){                                       //重写的 run()方法
10             while(DRAW_THREAD_FLAG){                            //while 循环
11                  gv.activity.world.step(TIME_STEP, ITERA);      //开始模拟
12                  gv.repaint();
13                  try{
14                       Thread.sleep(10);                         //休息 10ms
15                  }catch (InterruptedException e){               //捕获异常
16                       e.printStackTrace();                      //打印堆栈信息
17                  }
18                  for(MyBody mpi:gv.activity.bl){                //for 循环
19                       if(mpi instanceof MyRectColor){ //判断 mpi 是否属于 MyRectColor
20                            MyRectColor mrc=(MyRectColor)mpi;    //强制转换
21                            if(!mrc.isLive&&!mrc.isDeleted){
22                                 mrc.isDeleted=true;   //设置 isDeleted 为 true
23                                 gv.activity.world.destroyBody(mrc.body);
                                                         //销毁刚体
24                  } } }
25                  try{
26                       Thread.sleep(5);                          //休息 5ms
27                  }catch (InterruptedException e){               //捕获异常
28                       e.printStackTrace();                      //打印堆栈信息
29   } } } }
```

- 第 18-24 行为新添加的增强 for 循环，其主要作用是，判断 mpi 是否属于 MyRectColor 类，强制转换后，判断 isLive 与 isDeleted 是否全部为 false。如果全部为 false，则设置 isDeleted 为 true，并调用 destroyBody(Body b)方法销毁刚体。
- 第 25-29 行为新添加的捕获异常，这是由于调用 Thread.sleep()方法时需要捕获异常。

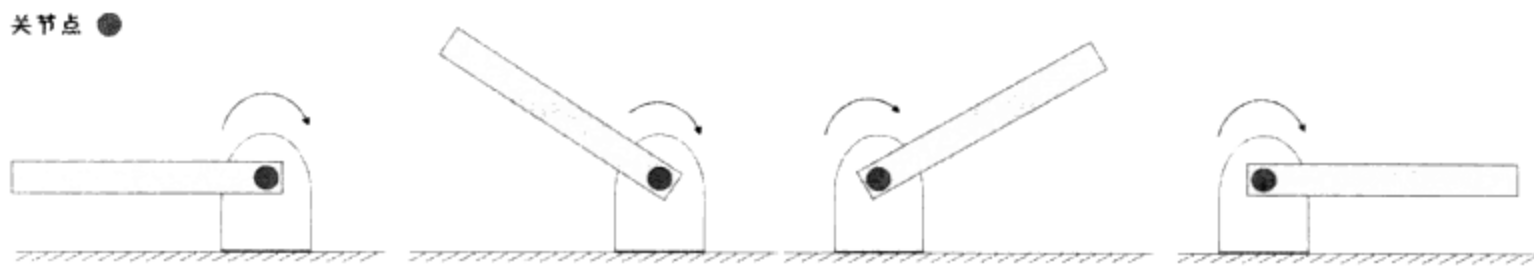
提示　经过前几个案例的讲解，细心的读者可能已经发现，如果应用项目的框架搭建的比较好，那么再次添加新的功能时，会起到事半功倍的效果。

10.5 旋转关节跷跷板案例

前面两节的案例主要介绍了刚体与刚体之间碰撞效果应用的开发，本节将给出一个用旋转关节实现跷跷板的案例。学会旋转关节的开发后，模拟现实世界的能力将大大增强。

10.5.1 旋转关节介绍

旋转关节是指物体的运动受限，只能绕物体上某一固定的点旋转。现实生活中旋转关节的例子很多，如门绕轴转动、起重机臂绕轴升降，图 10-29 给出了一个旋转关节的基本原理。



▲图 10-29 旋转关节示意图

10.5.2 多边形刚体类 MyPolygonColor 的开发

前面两个案例中只用到了圆形与矩形的刚体，本案例中需要用到三角形作为跷跷板的支撑。因此，首先需要开发多边形刚体类 MyPolygonColor，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_QQB/src/com.bn.box2d.qqb 目录下的 MyPolygonColor.java。

```
1    package com.bn.box2d.qqb;                                    //声明包名
2    import org.jbox2d.dynamics.Body;                             //导入相关包
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.qqb.Constant.*;                   //静态导入
5    public class MyPolygonColor extends MyBody{                  //自定义的矩形类(颜色)
6         Path path;                                              //创建 Path 的引用
7         float[][] points;                                       //顶点坐标二维数组
8         public MyPolygonColor(Body body,int color,float[][] points){      //构造器
9              this.body=body;                                    //初始化 body 成员变量
10             this.color=color;                                  //初始化 color 成员变量
11             path=new Path();                                   //初始化 path 成员变量
12             this.points=points;                                //初始化 points 成员变量
13        }
14        public void drawSelf(Canvas canvas,Paint paint){        //绘制方法
15             paint.setColor(color&0x8CFFFFFF);                  //画笔颜色
16             float x=body.getPosition().x*RATE;                 //刚体的中心点 X 坐标
17             float y=body.getPosition().y*RATE;                 //刚体的中心点 Y 坐标
18             float angle=body.getAngle();                       //刚体的旋转角度
19             canvas.save();                                     //保存画布状态
20             Matrix m1=new Matrix();                            //矩阵
```

```
21              ml.setRotate((float)Math.toDegrees(angle),x, y);          //旋转矩阵
22              canvas.setMatrix(ml);
23              path.reset();                                             //清空路径
24              path.moveTo(x+points[0][0], y+points[0][1]);    //顺时针旋转绕到下一个点
25              for(int i=1;i<points.length;i++){               //for 循环
26                   path.lineTo(x+points[i][0], y+points[i][1]);         //顺时针旋转绕到
                                                                          //下一个点
27              }
28              path.lineTo(x+points[0][0], y+points[0][1]);    //顺时针旋转绕到下一个点
29              canvas.drawPath(path, paint);                   //绘制
30              paint.setStyle(Paint.Style.STROKE);             //设置画笔样式
31              paint.setStrokeWidth(1);                        //画笔样式的宽度
32              paint.setColor(color);                          //设置颜色
33              canvas.drawPath(path, paint);                   //绘制
34              paint.reset();                                  //清空
35              canvas.restore();                               //恢复画布状态
36    } }
```

- 第 6-7 行为此类的成员变量，主要有 Path 的引用以及一个 float 类型的顶点坐标二维数组。
- 第 8-13 行为此类的构造器，在该构造器中，主要是初始化相应的成员变量。
- 第 14-36 行为多边形的绘制方法。首先设置画笔的颜色，并计算出中心点以及旋转的角度。然后保存画布，设置旋转矩阵，再顺时针旋转绕行该多边形的各个定点，存入 Path 对象中，并绘制。然后使用该 Path 对象绘制边框。

提示　这里的多边形为凸多边形，并且其旋转绕行各个顶点的顺序是顺时针方向。

10.5.3　生成刚体形状的工具类——Box2DUtil

上一小节介绍了多边形刚体类 MyPolygonColor 的开发，本小节将要介绍生成刚体形状的工具类 Box2DUtil 的开发。该类中有 createPolygon 与 createCircle 两个方法，这两个方法分别返回 MyPolygonColor 类与 MyCircleColor 类的对象，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_QQB/src/com.bn.box2d.qqb 目录下的 Box2DUtil.java。

```
package com.bn.box2d.qqb;                                         //声明包名
import org.jbox2d.collision.CircleDef;                            //导入相关包
......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
import static com.bn.box2d.qqb.Constant.*;                        //静态导入
public class Box2DUtil{                                           //生成物理形状的工具类
     public static MyPolygonColor createPolygon(                  //创建多边形物体(颜色)
          float x,                                                //x 坐标
          float y,                                                //y 坐标
          float[][] points,                                       //点序列
          boolean isStatic,                                       //是否为静止的
          World world,                                            //World 类
          int color                                               //颜色
     ){
          ......//该处省略了 createPolygon 方法中的具体代码，读者可以自行查阅随书光盘中源代码
          return new MyPolygonColor(bodyTemp,color,points);
                                                          //返回 MyPolygonColor 类对象
     }
     ......//该处省略了创建圆形刚体的方法，读者可自行查阅随书光盘中源代码
}
```

- 第 7-12 行为创建多边形物体 createPolygon 方法的参数，有 x 坐标、y 坐标，多边形的定点构成的二维数组，是否为静止标志位，World 类对象以及颜色。
- 第 15 行为返回 MyPolygonColor 类的对象。

10.5.4 将场景中的刚体摆放到位

在本案例中所有刚体摆放的位置全部在主控制类中实现，主要是调用 Box2DUtil 类中的 createPolygon 方法创建多边形刚体，调用 createCircle 方法创建圆形刚体。首先介绍 MyBox2dActivity 类的框架，该类继承自 Android 系统中的 Activity 类，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_QQB/src/com.bn.box2d.qqb 目录下的 MyBox2dActivity.java。

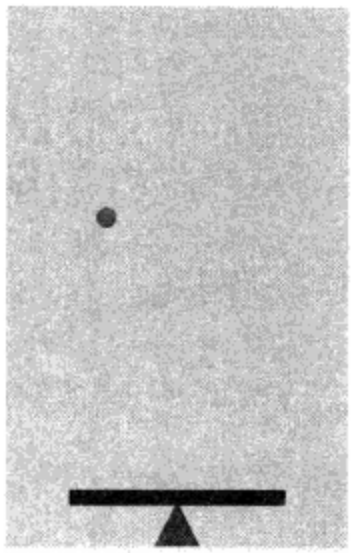
```
1    package com.bn.box2d.qqb;                                    //声明包名
2    import java.util.ArrayList;                                  //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.qqb.Constant.*;                   //静态导入
5    public class MyBox2dActivity extends Activity{               //创建继承 Activity 的类
6        AABB worldAABB;                                          //创建一个管理碰撞的 World
7        World world;                                             //创建 World 类的引用
8        ArrayList<MyBody> bl=new ArrayList<MyBody>();            //物体列表
9        public void onCreate(Bundle savedInstanceState){         //继承 Activity 需要重写的
                                                                  //方法
10           ......//该处省略了该方法中的代码，读者可以自行查阅随书光盘的源代码
11           final int kd=20;                                     //宽度或高度
12           MyPolygonColor mrc=Box2DUtil.createPolygon(          //调用 createPolygon 方法
13               0,                                               //x 坐标
14               SCREEN_HEIGHT-kd,                                //y 坐标
15               new float[][]{                                   //点序列
16                 {0,0},{SCREEN_WIDTH,0},
17                 {SCREEN_WIDTH,SCREEN_HEIGHT},{0,SCREEN_HEIGHT}
18               },
19               true,                                            //isStatic 标志位
20               world,                                           //World 类的对象
21               Color.YELLOW                                     //颜色
22           );
23           bl.add(mrc);                                         //MyPolygonColor 对象存
                                                                  //入集合
24           ......//该处省略了创建三角形以及木板的代码，读者可以自行查阅随书光盘的源代码
25           ......//该处注释掉了创建旋转关节的代码，将在后面给出
26           ......//该处省略了创建球与跳转界面的代码，读者可以自行查阅随书光盘的源代码
27   } }
```

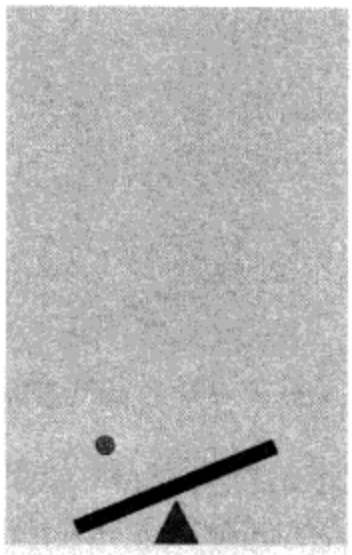
- 第 6-8 行为本类的成员变量，主要是声明 AABB 及 World 的引用，同时创建 ArrayList<MyBody>的集合，该集合中存放 MyBody 及其子类的对象。
- 第 12-23 行为创建本案例下侧的地面，主要是调用 Box2DUtil 类中的 createPolygon 方法创建 MyPolygonColor 的对象，并将该对象存入到 ArrayList<MyBody>集合中。

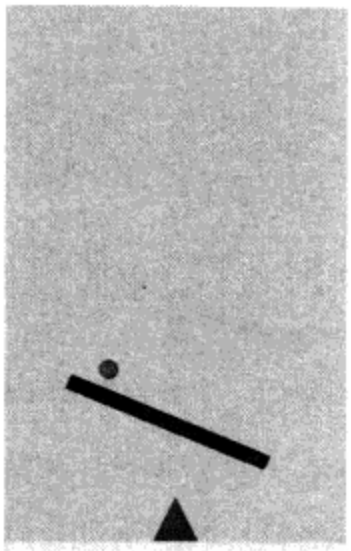
提示

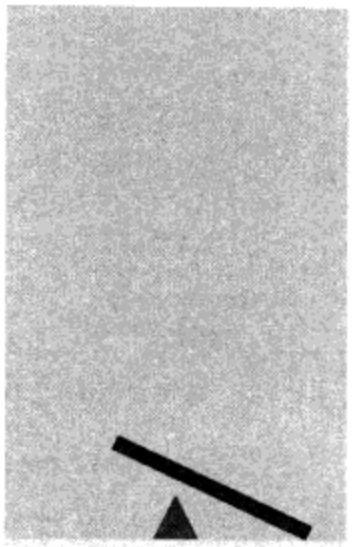
由于本案例中许多类的结构与前两个案例相似，因此，在这里就不再赘述。还有因为创建三角形以及木板的代码与第 12-22 行的代码结构相同，均是调用 createPolygon 方法，所以在这里将其省略，读者可以查看随书光盘中源代码。

运行本案例后小球将会垂直加速下降，直至撞击木板。如果将创建旋转关节的代码注释掉，该案例运行效果如图 10-30、图 10-31、图 10-32 和图 10-33 所示。


▲图 10-30　案例开始运行


▲图 10-31　小球碰撞木板


▲图 10-32　全部飞起


▲图 10-33　木板下落

说明　图 10-30 为小球开始下落，图 10-31 为小球撞击木板后小球反弹，图 10-32 为小球与木板全部弹起，图 10-33 为小球飞出屏幕，木板下降。

10.5.5　增加旋转关节

在 10.5.4 节中 MyBox2dActivity 类框架的第 25 行插入创建旋转关节的代码，使得三角形与木板共同构成旋转关节，其代码如下。

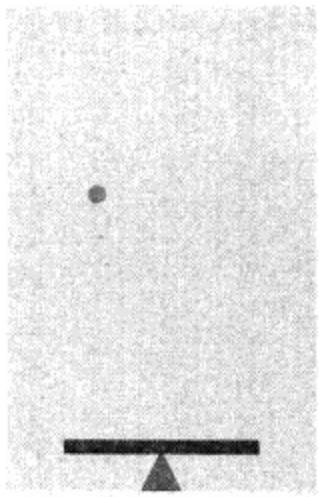
代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_QQB/src/com.bn.box2d.qqb 目录下的 MyBox2dActivity.java。

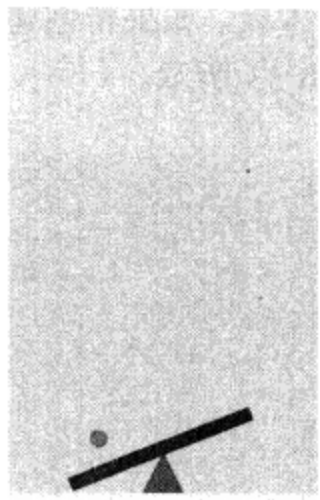
```
1    RevoluteJointDef rjd=new RevoluteJointDef();      //创建旋转关节描述对象
2    rjd.initialize(mrca.body, mrcb.body,              //调用 initialize 方法初始化关节描述
3                        new Vec2(SCREEN_WIDTH/2/RATE,(SCREEN_HEIGHT-kd-60)/RATE));
4    world.createJoint(rjd);                           //调用 createJoint 方法在 World 中创
                                                       //建关节
```

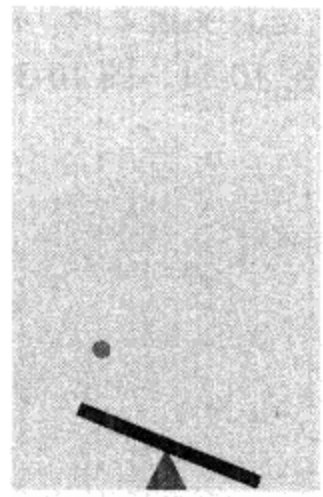
- 第 1 行为创建旋转关节描述对象，在创建旋转关节之前，首先需要创建旋转关节描述对象。
- 第 2-3 行为调用 initialize 方法。在本案例中，该方法的第一个参数是三角形刚体，第二个参数是木板刚体，第三个参数为关节点的坐标。一般调用 initialize 方法时，该方法的第一个参数为关节点相邻的第一个刚体，第二个参数为关节点相邻的第二个刚体，第三个参数为关节点坐标。
- 第 4 行是使用 World 类对象调用 createJoint 方法创建关节。

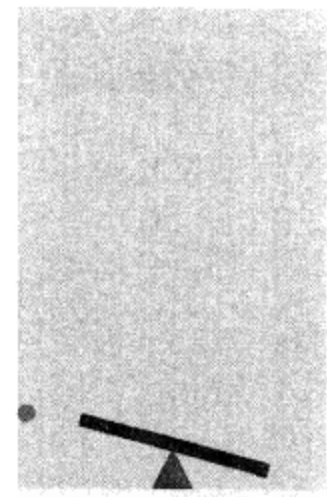
增加上旋转关节之后，使得小球在碰撞木板后，木板不会弹起来，只是绕关节点旋转，其运行效果如图 10-34、图 10-35、图 10-36 和图 10-37 所示。

说明　图 10-34 为小球开始下落，图 10-35 为小球与木板碰撞后小球弹起，图 10-36 为木板向右旋转一定角度，图 10-37 为小球与木板再次碰撞。


▲图 10-34　案例开始运行


▲图 10-35　小球与木板碰撞


▲图 10-36　木板向右旋转


▲图 10-37　再次碰撞

在上述的几个旋转关节中，三角形的 isStatic 标志始终为 true，如果 isStatic 改为 false，则小球碰撞木板后，木板与三角形连成一体并弹起，修改后的创建三角形的代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_QQB/src/com.bn.box2d.qqb 目录下的 MyBox2dActivity.java。

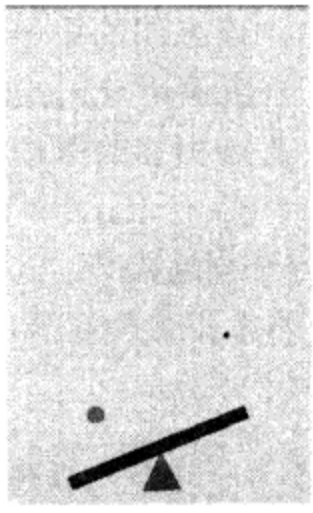
```
1    MyPolygonColor mrca=Box2DUtil.createPolygon(              //调用 createPolygon 方法
2         SCREEN_WIDTH/2-30,                                   //x 坐标
3         SCREEN_HEIGHT-kd-62,                                 //y 坐标
4         new float[][]{                                       //点序列
5              {30,0},{60,60},{0,60}},
6         false,                                               //isStatic 标志位
7         world,                                               //World 类对象
8         Color.RED                                            //颜色
9         );
```

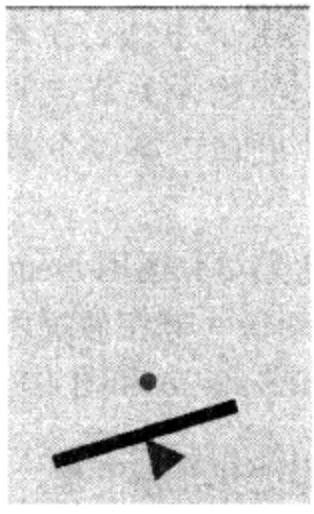
说明　在第 6 行更改了 isStatic 标志位为 false，该标志位的作用是判断其是否为静态。

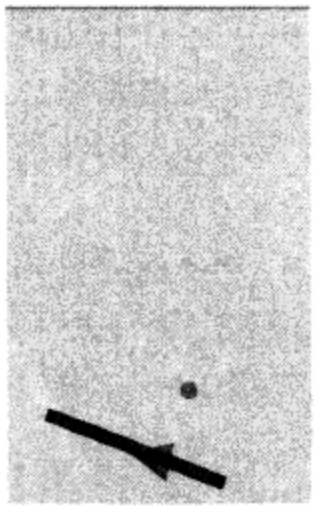
将三角形的 isStatic 标志位设置为 false 后，三角形变为非静态，其运行效果如图 10-38、图 10-39、图 10-40 和图 10-41 所示。

说明　图 10-38 为案例开始运行，图 10-39 为小球与木块碰撞后小球弹起，图 10-40 为小球、木块以及三角形均飞起，图 10-41 为小球与木块碰撞后再次飞起。

▲图 10-38　案例开始运行


▲图 10-39　小球与木板碰撞


▲图 10-40　全部飞起


▲图 10-41　再次碰撞后飞起

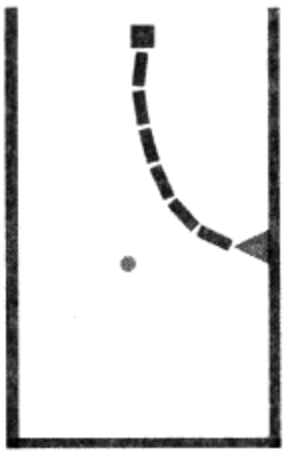
10.6 旋转关节链条摆案例

前一个案例讲解了旋转关节构成的跷跷板案例，接下来将为读者介绍，如何同时使用多个关节，实现链条摆动的案例。学会本案例，有助于进一步帮助读者加深对关节的理解。

10.6.1　案例运行效果

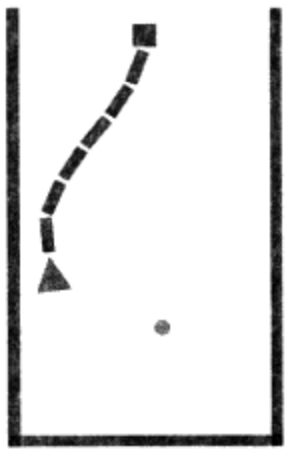
本案例用多个关节实现了一个链条摆动的场景，案例开始运行后，从屏幕左侧飞出一个具有水平初速度的小球。该小球撞击链条下侧的三角形，使得链条开始摆动，其运行效果如图 10-42、图 10-43、图 10-44 和图 10-45 所示。

▲图 10-42　案例开始运行



▲图 10-43　链条摆动至右侧

▲图 10-44　再次碰撞

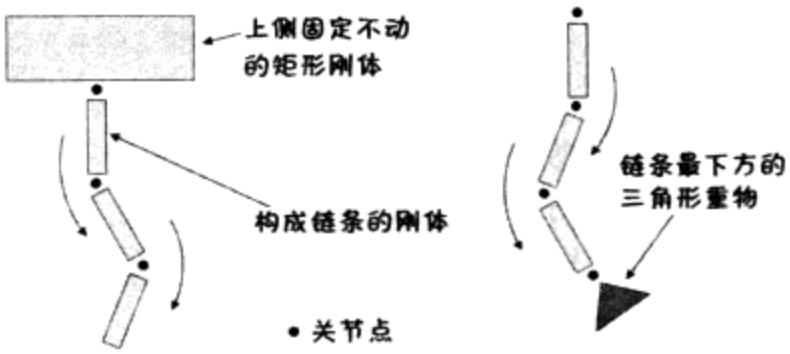


▲图 10-45　链条摆动至左侧

说明　图 10-42 为案例开始运行时，图 10-43 为小球碰撞链条后链条摆动至右侧，图 10-44 为小球与链条再次碰撞，图 10-45 为链条摆动至左侧。

10.6.2　案例的机械结构

上一小节介绍了本案例的运行效果，从图 10-42～图 10-45 中可以看出，该链条是由 6 个长条形刚体以及 5 个旋转关节构成的，其机械结构如图 10-46 所示。



▲图 10-46　链条的机械机构

链条顶端的长条矩形用一个旋转关节连接到了一块固定的矩形刚体上，链条的最下面用旋转关节与一个三角形重物连接。整个机械系统中用到了 7 个旋转关节。

10.6.3 主控制类——MyBox2dActivity

本案例中的大部分代码与旋转关节跷跷板案例相似，只是在主控制类——MyBox2dActivity 中更改了刚体的位置、数量、尺寸，关节的位置以及小球的初速度大小、方向。改动后的 MyBox2dActivity 类代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_LT/src/com.bn.box2d.lt 目录下的 MyBox2dActivity.java。

```
1    package com.bn.box2d.lt;                                          //声明包名
2    import java.util.ArrayList;                                       //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.lt.Constant.*;                         //静态导入
5    public class MyBox2dActivity extends Activity{                    //创建继承 Activity 的类
6        ......//该处省略了声明此类成员变量的代码，读者可自行查阅随书光盘
7        public void onCreate(Bundle savedInstanceState){              //重写 onCreate 方法
8            ......//该处省略了该方法的部分代码，读者可自行查阅随书光盘中源代码
9            final int kd=20;                                          //宽度或高度
10           MyPolygonColor mrc=Box2DUtil.createPolygon(               //调用 createPolygon 方法
11               0,                                                    //x 坐标
12               SCREEN_HEIGHT-kd,                                     //y 坐标
13               new float[][]{                                        //点序列
14                 {0,0},{SCREEN_WIDTH,0},{SCREEN_WIDTH,kd},{0,kd}},
15               true,                                                 //isStatic 标志位
16               world,                                                //World 类的对象
17               0xff0000ff);                                          //颜色
18           bl.add(mrc);                                            // MyPolygonColor 对象存入集合
19           ......//该处省略了创建其他两个面以及上面方块的代码，读者可自行查阅随书光盘中的源代码
20           int ltCount=6;                                            //链条数量
21           MyPolygonColor[] ltA=new MyPolygonColor[ltCount];         //创建MyPolygonColor
                                                                       //类数组
22           for(int i=0;i<ltCount;i++){                               //循环创建每个链条
23               ltA[i]=Box2DUtil.createPolygon(                       //调用 createPolygon
                                                                       //方法
24                   SCREEN_WIDTH/2-kd/2,                              //x 坐标
25                   4.5f*kd+i*3.5f*kd,                                //y 坐标
26                   new float[][]{                                    //点序列
27                     {0,0},{kd,0},{kd,3*kd},{0,3*kd}},
28                   false,                                            //isStatic 标志位
29                   world,                                            //World 类对象
30                   0xff0000ff);                                      //颜色
31               bl.add(ltA[i]);                                     // MyPolygonColor 对象存入集合
32           }
33           ......//该处省略了创建三角形的代码，读者可自行查阅随书光盘中的源代码
34           for(int i=0;i<ltCount;i++){                             //循环创建关节
35               if(i==0){                                           //i 为 0 的情况
36                   RevoluteJointDef rjd=new RevoluteJointDef();
                                                                     //创建 RevoluteJointDef 类对象
37                   rjd.initialize(fk.body,ltA[i].body,             //调用 initialize 方法
38                           new Vec2((SCREEN_WIDTH/2)/RATE,(4.25f*kd+i*3.5f*kd)/RATE));
39                   world.createJoint(rjd);                         //创建关节
40               }else{
41                   RevoluteJointDef rjd=new RevoluteJointDef();
                                                                     //创建 RevoluteJointDef 类对象
42                   rjd.initialize(ltA[i-1].body,ltA[i].body,    //调用 initialize 方法
43                           new Vec2((SCREEN_WIDTH/2)/RATE,(4.25f*kd+i*3.5f*kd)/RATE));
44                   world.createJoint(rjd);                         //创建关节
45           } }
```

```
46          RevoluteJointDef rjd=new RevoluteJointDef();   //创建 RevoluteJointDef 类对象
47          rjd.initialize(ltA[ltCount-1].body,sjx.body,   //调用 initialize 方法
48                  new Vec2((SCREEN_WIDTH/2)/RATE,(4.5f*kd+ltCount*3.5f*kd)/RATE));
49          world.createJoint(rjd);                        //创建关节
50          ......//该处省略了创建球与跳转界面的代码，读者可以自行查阅随书光盘的源代码
51    } }
```

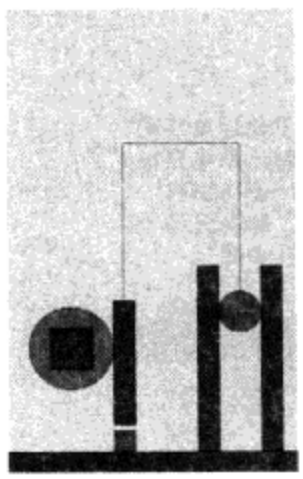
- 第 9-18 行为创建地面，首先声明其 kd 的值为 20，然后调用 Box2DUtil 类中的 createPolygon 方法创建下侧地面的矩形刚体，最后需要将得到的 MyPolygonColor 对象存入到 ArrayList<MyBody>集合中。
- 第 20-32 行为循环创建链条，首先设置链条中刚体的数量，然后采用 for 循环，调用 Box2DUtil 类中的 createPolygon 方法创建链条中的刚体，最后将得到的 MyPolygonColor 对象存入到 ArrayList<MyBody>集合中。
- 第 34-45 行为给链条中的每个刚体设置旋转关节，需要判断其是否为链条的第一个刚体，如果是，则为其与上侧固定不动的方块之间创建旋转关节。否则，创建该刚体与上一刚体之间的旋转关节。
- 第 46-49 行为最下面的链条与三角形创建关节点。首先创建 RevoluteJointDef 类对象，然后调用 initialize 方法，最后再创建关节。

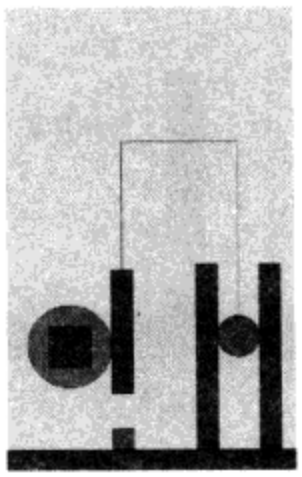
10.7 组合机械结构案例

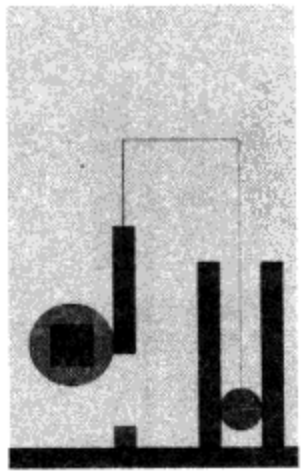
前面两个小节主要介绍了旋转关节的使用，但强大的 JBox2D 中还有几种其他类型的关节。本小节将通过一个综合使用几种关节的案例向读者进行介绍，掌握了多种不同的关节后，模拟现实世界的能力将进一步提高。

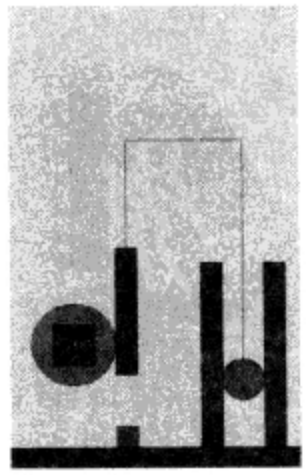
10.7.1　案例的运行效果

介绍本案例的具体结构之前，先介绍本案例的运行效果。开始运行后，左下侧的圆形齿轮逆时针/顺时针旋转，使得左侧的矩形刚体上升/下降，对应右侧的小球下落/上升，其运行效果如图 10-47、图 10-48、图 10-49 和图 10-50 所示。


▲图 10-47　程序开始运行


▲图 10-48　齿轮逆时针旋转


▲图 10-49　逆时针至最大


▲图 10-50　齿轮顺时针旋转

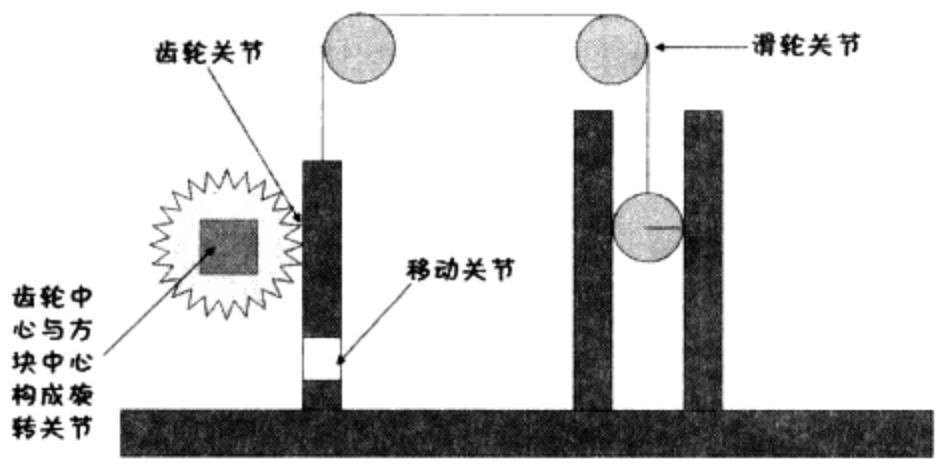
说明　图 10-47 为案例开始运行，图 10-48 为齿轮逆时针旋转一定角度，图 10-49 为齿轮逆时针旋转至最大，图 10-50 为齿轮顺时针旋转一定角度。

10.7.2 整体场景的机械结构

本案例中使用了几种不同的关节，有旋转关节、齿轮关节、移动关节以及滑轮关节，每个关节有着不同的作用。关节在本案例中起到了重要的作用，缺少上述的任何一个关节，本案例将无法运行成功。下面将为读者介绍本案例的机械结构，如图 10-51 所示。

- 齿轮关节

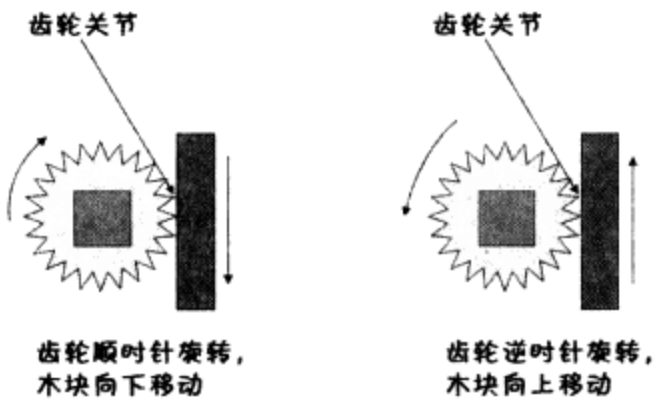
齿轮关节的主要作用是齿轮旋转，使得齿轮右侧的木块也随之上下移动。即齿轮顺时针转动，木块就向下移动；齿轮逆时针转动，木块就会向上移动。齿轮关节的机械结构如图 10-52 所示。



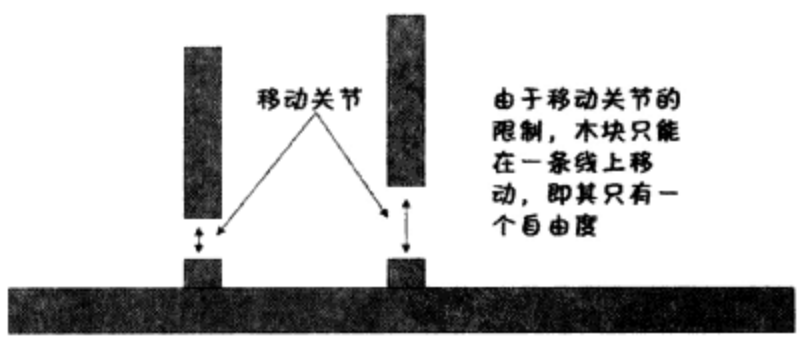
▲图 10-51　本案例的机械结构图

- 移动关节

移动关节的作用为限制连接到关节的两个刚体只能沿关节指定的方向移动，即只有一个自由度。本案例中使用移动关节的目的是，限制左侧的矩形刚体只能上下移动，避免跑偏。移动关节的机械结构如图 10-53 所示。



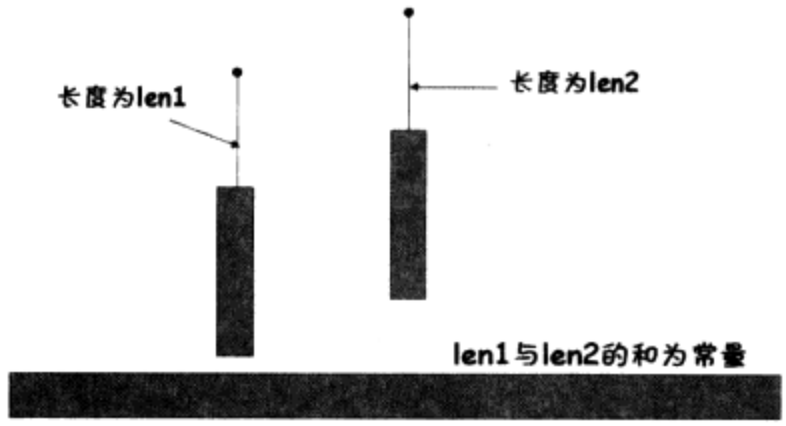
▲图 10-52　齿轮关节机械结构



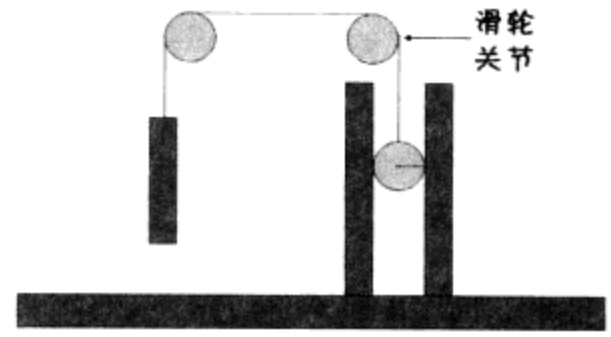
▲图 10-53　移动关节机械结构

- 滑轮关节

滑轮关节的作用是连接两个刚体，使得两个刚体到各自指定位置的距离和为常量，其基本原理如图 10-54 所示。本案例中用滑轮关节联系了左侧上下移动的刚体与右侧上下移动的圆球，使得他们的运动是线性相关的，其结构如图 10-55 所示。



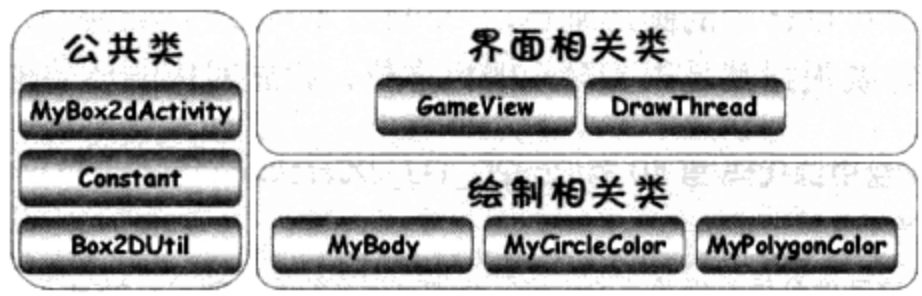
▲图 10-54　滑轮基本结构图



▲图 10-55　本案例滑轮关节机械结构

10.7.3　案例的基本框架结构

介绍本案例的具体代码之前，首先介绍本案例的基本框架结构。也是在开发本案例时的基本思路，希望读者认真理解本案例的基本框架结构，这非常有助于后面细节的学习。该框架结构如图 10-56 所示。



▲图 10-56　案例基本框架结构

说明　本案例的框架结构图与 10.3.2 小节中球体撞击木块金字塔案例的基本框架结构类似，只是将球体撞击木块金字塔中的 MyRectColor（矩形）类，改变为 MyPolygonColor（多边形）类，并去除了 ColorUtil（颜色工具）类。因此在本小节中，就不再对各个类的具体情况进行赘述，感兴趣的读者可以参考球体撞击木块金字塔基本框架中相关类的解释。

10.7.4　圆形刚体类——MyCircleColor

本案例中使用圆形刚体代替齿轮，圆形刚体不断的旋转，并通过齿轮关节使得木块可以随着该圆形刚体上下移动。因此，需要根据当前圆形刚体的角度，绘制标志该圆形刚体转动的半径，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_JX/src/com.bn.box2d.jx 目录下的 MyCircleColor.java。

```
1    package com.bn.box2d.jx;                                          //声明包名
2    import org.jbox2d.dynamics.Body;                                  //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.jx.Constant.*;                         //静态导入
```

```
5    public class MyCircleColor extends MyBody{                          //自定义的圆形类
6         ......//该处省略了成员变量以及构造器的代码，读者可以自行查阅随书光盘的源代码
7         public void drawSelf(Canvas canvas,Paint paint){            //绘制方法
8              ......//该处省略了部分绘制的代码，读者可以自行查阅随书光盘的源代码
9              canvas.save();                                          //保存画布状态
10             float angle=body.getAngle();                            //得到刚体的角度
11             Matrix m1=new Matrix();                                 //创建矩阵对象
12             m1.setRotate((float)Math.toDegrees(angle),x, y);        //旋转矩阵
13             canvas.setMatrix(m1);
14             paint.reset();                                          //清空画笔
15             paint.setColor(0xFFE0E0E0);                             //设置画笔颜色
16             canvas.drawLine(x, y, x+radius, y, paint);              //绘制线(标志作用)
17             canvas.restore();                                       //恢复画布状态
18   } }
```

- 第 9 行为保存画布状态，使得改变图形的一些属性不影响上面已经创建的。
- 第 10-13 行为得到刚体旋转的角度，创建 Matrix 对象，同时设置旋转矩阵。
- 第 14-17 行为清空画笔，设置画笔颜色，绘制标志半径，并恢复画布状态。

10.7.5 生成刚体形状的工具类——Box2DUtil

本案例中有一个自身可以转动的圆形刚体，一个自身不可以转动的圆形刚体。因此在 Box2DUtil 类中创建圆形刚体时，需要根据其是否为静止的标志位，设置对应刚体的密度，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_JX/src/com.bn.box2d.jx 目录下的 Box2DUtil.java。

```
1    package com.bn.box2d.jx;                                          //声明包名
2    import org.jbox2d.collision.CircleDef;                            //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.jx.Constant.*;                         //静态导入
5    public class Box2DUtil{                                           //生成物理形状的工具类
6         ......//该处省略了创建多边形刚体的代码，读者可以自行查阅随书光盘中源代码
7         public static MyCircleColor createCircle(                    //创建 createCircle 方法
8              float x,                                                //x 坐标
9              float y,                                                //y 坐标
10             float radius,                                           //半径
11             World world,                                            //World 类
12             int color,                                              //颜色
13             boolean isStatic                                        //是否为静止的
14        ){
15             ......//该处省略了创建圆形刚体的具体代码，读者可以自行查阅随书光盘中的源代码
16             return new MyCircleColor(bodyTemp,radius,color);
                                                                       //返回 MyCircleColor 类的对象
17   } }
```

- 第 8-13 行为 Box2DUtil 类中的 createCircle 方法的具体参数，主要是 *x* 坐标、*y* 坐标、圆形半径、World 类的对象、颜色以及是否为静止的标志位。
- 第 16 行为 createCircle 方法的返回值，该返回值为 MyCircleColor 类的对象。

10.7.6 主控制类——MyBox2dActivity

本案例的主控制类主要实现了旋转关节、马达、齿轮关节、移动关节以及滑轮关节的开发。首先介绍 MyBox2dActivity 类的框架，其代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_JX/src/com.bn.box2d.jx 目录下的 MyBox2dActivity.java。

```
1    package com.bn.box2d.jx;                                          //声明包名
2    import java.util.ArrayList;                                       //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.jx.Constant.*;                         //静态导入
5    public class MyBox2dActivity extends Activity{                    //创建继承 Activity 的类
6        AABB worldAABB;                                               //创建一个管理碰撞的 World
7        World world;
8        ArrayList<MyBody> bl=new ArrayList<MyBody>();                 //物体列表
9        RevoluteJoint rj1;                                            //左侧齿轮的旋转关节
10       PrismaticJoint pj1;                                           //左侧捶打的移动关节
11       MyPolygonColor clt2;                                          //创建左侧结合齿轮的矩形块
12       MyCircleColor ball;                                           //创建右侧的球
13       public void onCreate(Bundle savedInstanceState){
14           ......//该处省略了该方法的部分代码，读者可以自行查阅随书光盘中的源代码
15           MyCircleColor cl1=Box2DUtil.createCircle                  //创建圆形刚体对象
16           (3*kd, SCREEN_HEIGHT-6*kd, 2*kd, world,Color.MAGENTA,false);
17           bl.add(cl1);                                              //将该刚体添加进集合中
18           RevoluteJointDef rjd1=new RevoluteJointDef();             //创建左侧齿轮的旋转关节描述
19           rjd1.initialize(                                          //调用 initialize 方法初始
                                                                       //化关节描述
20               clt1.body,                                            //第一个刚体
21               cl1.body,                                             //第二个刚体
22               cl1.body.getWorldCenter()                             //关节点的坐标
23           );
24           rjd1.lowerAngle=(float) (-0.95);                          //最小角度
25           rjd1.upperAngle=(float) (0.95);                           //最大角度
26           rjd1.enableLimit=true;                                    //是不是开启角度限制
27           rjd1.maxMotorTorque=100000;                               //最大马达
28           rjd1.motorSpeed=(float) (-Math.PI/2);                     //马达速度
29           rjd1.enableMotor=true;                                    //是否开启马达
30           rj1=(RevoluteJoint) world.createJoint(rjd1);              //调用 createJoint 方法在
                                                                       //World 中创建关节
31           ......//该处省略了创建左侧两个矩形块的代码，读者可以自行查阅随书光盘中的源代码
32           ......//该处省略了创建移动关节的代码，将在后面给出
33           ......//该处省略了创建右侧的齿轮关节的代码，将在后面给出
34           ......//该处省略了创建右侧球的代码，读者可以自行查阅随书光盘中的源代码
35           ......//该处省略了创建左右侧滑轮关节的代码，将在下面给出
36           ......//该处省略了左侧的两个限制矩形以及跳转界面的代码，读者可以自行查阅随书光盘的源代码
37   } }
```

- 第 6-12 行为此类中的成员变量，有 AABB（管理碰撞的 World）、World（世界）、物体列表、左侧齿轮的旋转关节、左侧捶打的移动关节、与左侧齿轮结合的矩形块及圆形刚体类。
- 第 15-17 行为创建左侧圆形刚体对象，并存入到 ArrayList<MyBody>集合中。
- 第 24-26 行是设置该旋转关节的角度限制，并设置 enableLimit 为 true 开启角度限制。
- 第 27-29 行为设置该旋转关节的最大马达，设置马达速度，并设置 enableMotor 为 true 开启马达。

（1）上述代码主要介绍了 MyBox2dActivity 类的框架，接下来将介绍如何创建移动关节，将下列代码插入到 MyBox2dActivity 类框架的第 32 行。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_JX/src/com.bn.box2d.jx 目录下的 MyBox2dActivity.java。

```
1    PrismaticJointDef pjd=new PrismaticJointDef();                    //创建移动关节描述对象
```

```
2    pjd.initialize(clt3.body, clt2.body, clt3.body.getWorldCenter(), new Vec2(0,1));
3    pj1=(PrismaticJoint) world.createJoint(pjd);  //调用 createJoint 方法在世界中创建关节
```

说明 创建移动关节首先需要获得移动关节描述对象，然后调用 initialize 方法初始化移动关节描述对象，最后调用 createJoint 方法在 World 中创建关节。

（2）本案例中使用到了齿轮关节，其使得木块可以随着齿轮的旋转而上下移动。接下来介绍如何创建齿轮关节，将下列代码插入到 MyBox2dActivity 框架的第 33 行。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_JX/src/com.bn.box2d.jx 目录下的 MyBox2dActivity.java。

```
1    GearJointDef gjd=new GearJointDef();          //创建齿轮关节描述对象
2    gjd.body1=cl1.body;                           //设置该齿轮关节的第一个刚体
3    gjd.body2=clt2.body;                          //设置该齿轮关节的第二个刚体
4    gjd.joint1=rj1;                               //设置该齿轮关节的第一关节
5    gjd.joint2=pj1;                               //设置该齿轮关节的第二关节
6    gjd.ratio=(-1.0f/(2.0f*kd)*RATE);
7    world.createJoint(gjd);                       //调用 createJoint 方法在世界中创建关节
```

- 第 1-3 行为创建该齿轮关节的描述对象，并设置该齿轮关节的两个刚体。
- 第 4-6 行为设置该齿轮的旋转关节以及移动关节，并设置该齿轮的半径。
- 第 7 行为调用 createJoint 方法在 World 中创建关节。

（3）本案例中使用到了滑轮关节，使得左侧木块与右侧小球的位移大小相等，方向相反，将下列代码插入到 MyBox2dActivity 框架的第 35 行。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_JX/src/com.bn.box2d.jx 目录下的 MyBox2dActivity.java。

```
1    PulleyJointDef pjd2=new PulleyJointDef();        //创建滑轮关节描述对象
2    pjd2.initialize(                                 //调用 initialize 方法初始化关节描述
3         clt2.body,                                  //第一个刚体
4         ball.body,                                  //第二个刚体
5         new Vec2(3*kd,SCREEN_HEIGHT-16*kd),         //滑轮关节左面上侧定位点的坐标
6         new Vec2(11*kd,SCREEN_HEIGHT-16*kd),        //滑轮关节右面上侧定位点的坐标
7         clt2.body.getWorldCenter(),                 //左侧结合齿轮矩形块的中心点
8         ball.body.getWorldCenter(),                 //右侧小球的中心点
9         1f
10        );
11   world.createJoint(pjd2);                         //调用 createJoint 方法在 World 中创建关节
```

- 第 1 行为创建滑轮关节描述对象。
- 第 2-10 行为调用 initialize 方法初始化关节描述，该方法的主要参数有木块刚体、右侧圆形刚体、上侧滑轮的坐标、左侧结合齿轮的矩形块、小球的中心点以及半径。
- 第 11 行为调用 createJoint 方法在 World 中创建关节。

10.7.7 游戏界面类——GameView

本案例中，链接左侧木块与右侧小球的线段并不是刚体，而是直接在 GameView 类中 onDraw 方法里绘制的。这些线段的作用仅仅是视觉上提示用户滑轮的存在而已，相关代码如下。

代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_JX/src/com.bn.box2d.jx 目录下的 GameView.java。

```
1    paint.setColor(Color.RED);                                          //设置画笔颜色
2    canvas.drawLine(11*kd,SCREEN_HEIGHT-16*kd,5.5f*kd,SCREEN_HEIGHT-16*kd, paint);
                                                                         //绘制滑轮横线
3    canvas.drawLine(11*kd, SCREEN_HEIGHT-16*kd, 11*kd,                  //绘制滑轮右侧竖线
4                        activity.ball.body.getWorldCenter().y*RATE-kd,paint);
5    canvas.drawLine(5.5f*kd, SCREEN_HEIGHT-16*kd, 5.5f*kd,              //绘制滑轮左侧竖线
6                        activity.clt2.body.getWorldCenter().y*RATE-3*kd,paint);
```

- 第 1 行为设置画笔的颜色为红色。
- 第 2-6 行为调用 drawLine 方法，绘制用于表示滑轮的左侧线段、右侧线段以及上侧的线段。

10.7.8　绘制线程类——DrawThread

由于本案例中左下侧的圆形齿轮在转动时有范围限制，所以，需要在线程中判断圆形齿轮的角度，要增加的代码如下。

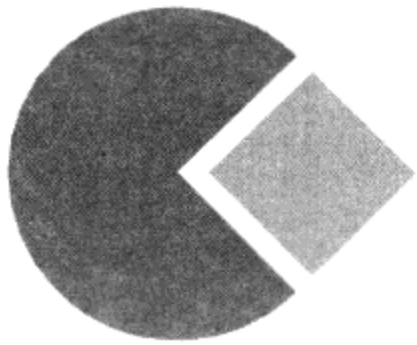
代码位置：见随书光盘中源代码/第 10 章/Box2D_GJ_JX/src/com.bn.box2d.jx 目录下的 DrawThread.java。

```
1    float angle=gv.activity.rj1.getJointAngle();          //得到左下侧圆形刚体转动的角度
2    if(angle<(float) (-0.95)){                            //角度小于-0.95(单位为弧度)
3         gv.activity.rj1.setMotorSpeed((float)(Math.PI/2));      //设置马达的速度
4    }else if(angle>(float) (0.95)){                       //角度大于 0.95(单位为弧度)
5         gv.activity.rj1.setMotorSpeed((float)(-Math.PI/2));     //设置马达速度
6    }
```

- 第 1 行为得到左下侧圆形刚体转动的角度。
- 第 2-6 行为判断齿轮转动的角度是否已经超出规定的范围，如果超出，则将马达速度置反。

10.8　小结

本章介绍了物理引擎的基本知识，并简要介绍了几款流行的物理引擎。同时深入讨论了开源物理引擎 Box2D 的 Java 版本——JBox2D，通过多个案例为读者讲解了一些必知必会的知识。主要有如何使用 JBox2D 物理引擎实现碰撞检测、旋转关节、齿轮关节、移动关节以及滑轮关节。学会使用 JBox2D 物理引擎后，开发 2D 游戏的能力将大大提高。



第 11 章　OpenGL ES 应用开发基础

OpenGL ES 是针对手持移动平台设计的 3D 图形应用程序编程接口，本章将介绍 OpenGL ES 在 Android 平台下的应用，及一些基本图形的案例，使读者掌握 OpenGL ES 应用开发的基础。

11.1 OpenGL ES 概述及 3D 基本图形绘制

本节将要对 OpenGL ES 及 3D 基本图形绘制进行简单介绍，通过本节的学习，读者可以对 OpenGL ES 有初步的认识，并掌握最基本图形的开发。

11.1.1　OpenGL 及 OpenGL ES 简介

OpenGL 是由 SGI 公司开发的一套 3D 图形软件接口标准，由于具有体系结构简单合理、使用方便、与操作平台无关等优点，OpenGL 迅速成为一种 3D 图形接口的工业标准，并陆续在各种平台上得以实现，其优越性为。

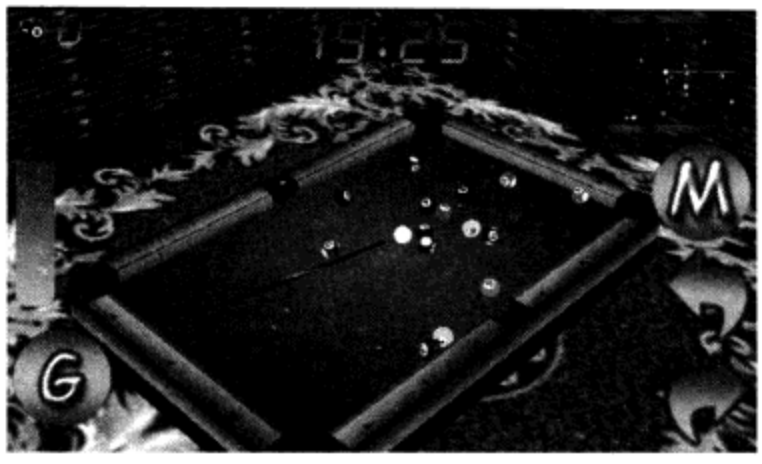
- 其作为一个性能优越的图形应用程序编程接口，适合于广泛的计算环境。从个人计算机工作站和超级计算机。OpenGL 都能很好地实现高性能的 3D 图形运算。
- 由于许多在 IT 界具有领导地位的公司纷纷采用 OpenGL 作为他们的三维图形应用程序编程接口，这使得 OpenGL 应用程序具有广泛的可移植性。

OpenGL ES 是根据手持移动平台的特点对 OpenGL 3D 图形 API 标准进行裁剪定制而成的。OpenGL ES 官方网址是 http://ww.khronos.org/。该组织关注手持和移动平台上 3D 应用程序开发的 API，并致力于为这些 API 建立无权限费用的开放标准。OpenGL ES 的优越性有如下几点。

- 需要 3D UI 的移动嵌入式应用，由于受随时移动等因素的影响，一般不能提供无限的电源连接，只能在有限的电能限制下工作。如何以更低的功耗完成 3D 场景的渲染，成为 OpenGL 必须要面对的问题，进而推出了 OpenGL ES 标准。应该说其在高效渲染 3D 界面的同时，达到了降低功耗的效果。
- OpenGL ES 基本上是 OpenGL 1.3 的子集，同时加了一些扩展。这使得该 API 更加灵活，比如现在一些用不到的功能可以暂时删除，当内嵌硬件发展到一定水平后，相应的功能可以重新添加回来。

目前的 OpenGL ES 两方面基本内容，Common Profile 支持全 3D 功能，保证游戏正常运行。Safety Critical Profile，这是商业软件需要的，华丽的界面在这里不是最优考虑，安全性成为关键，

它只能提供最小化的 3D 功能。大家也不要觉得 OpenGL ES 是 OpenGL 的缩水版，新发布的 Sony PlayStation3 就是采用的 OpenGL ES。可见 OpenGL ES 还是比较强大的，足以承载各种 3D 游戏的渲染。用 OpenGL ES 开发的游戏如图 11-1、图 11-2 所示。



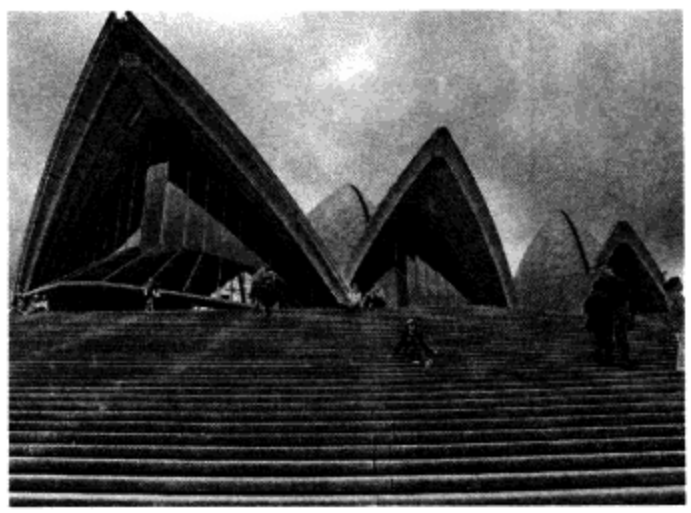
▲图 11-1　3D 桌球

▲图 11-2　3D 赛车

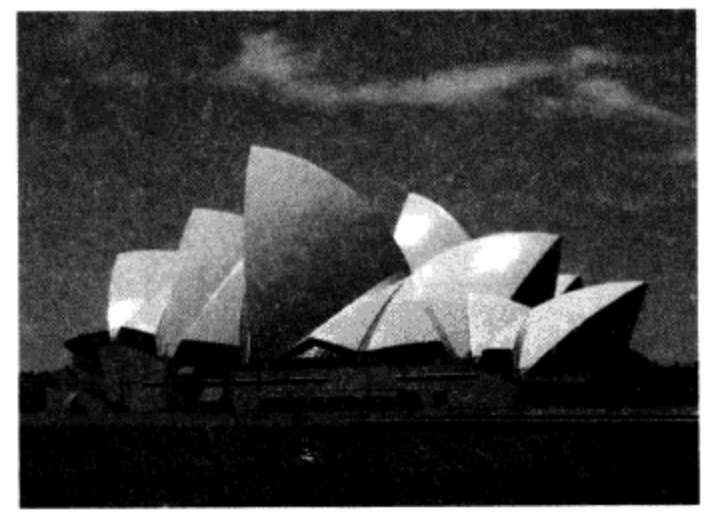
说明　图 11-1 与图 11-2 中的游戏都是 Android 手机游戏，这些游戏都是用 OpenGL ES 实现的，可见 OpenGL ES 功能之强大。

11.1.2　3D 基本知识

3D 图形的最基本单位称为顶点 vertex，他们代表三维空间中的一个点并可以用来建造更复杂的物体。多边形就是由点构成，而三维物体是由多个多边形组成的，如图 11-3 所示。



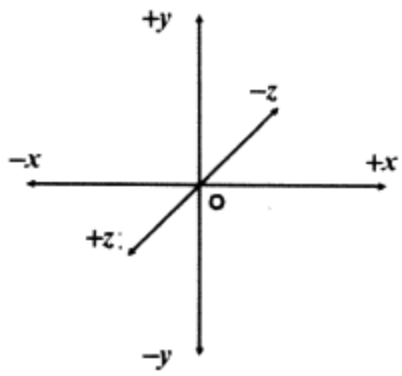
▲图 11-3　悉尼歌剧院近景



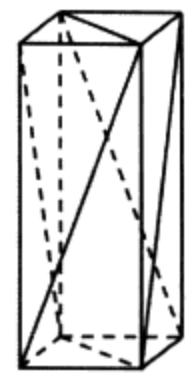
▲图 11-4　悉尼歌剧院远景

说明　三维物体是由多个多边形组成的，图 11-3 和图 11-4 中悉尼歌剧院也是由多个多边形组成。

尽管通常 OpenGL 支持多种多边形，但目前 OpenGL ES 只支持三角形，这主要是出于性能的原因。从能力上来说只支持三角形与支持多种多边形是一样的，因为任何多边形都可以拆分成多个三角形，缺点是开发时会麻烦一些。图 11-5 显示了 OpenGL ES 采用的三维迪卡尔坐标系。



▲图 11-5　OpenGL ES 采用的三维迪卡尔坐标系

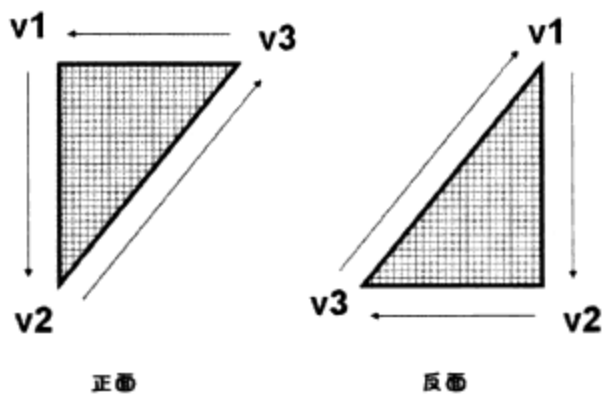


▲图 11-6　长方体

OpenGL ES 中，通常将场景内构成所有或部分物体的顶点 *X*、*Y*、*Z* 坐标值以顶点数组的形式给出。一个顶点数组是包括场景中部分或所有顶点坐标数据的简单数组。如场景中有 *n* 个顶点，则坐标值有 3*n* 个，则顶点数组的尺寸为 3*n*。

前面已经讲过，OpenGL ES 中只允许使用三角形进行填充。下面的示意图给出了用平面三角形搭建立体图形的原理。图 11-6 中是一个长方体，有 6 个面，每个面是一个矩形，可以切分成两个三角形。因此，在 OpenGL ES 中，此长方体可以用 12 个三角形填充。

OpenGL 中有一项叫背面剪裁的功能，含义就是打开背面剪裁功能后，视角在一个三角形的背面时不渲染此三角形，此功能可以提高渲染效率。因此很重要的就是确定在既定的观察方向上要渲染三角形，否则就有可能出现看不到图形的情况。三角形的正反面是这样确定的，当面对一个三角形时，若顶点的顺序是逆时针的则位于三角形的正面，反之则在反面，如图 11-7 所示。



▲图 11-7　三角形的正反面

提示　上述是默认情况，也可以自己设置正反面的顶点旋转绕行顺序。

11.1.3　使用索引的不同绘制方式

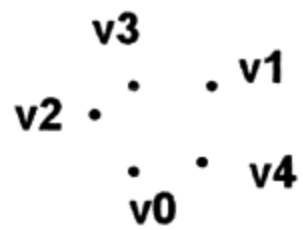
OpenGL ES 中可以利用顶点的索引来绘制基本图形，其支持很多种绘制方式，每种绘制方式分别针对不同的情况，各种绘制方式如下所列。

- GL_POINTS

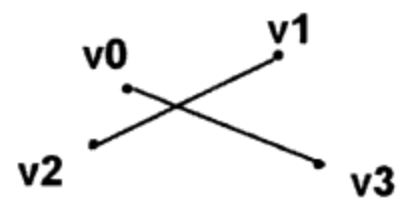
把每个顶点作为一个点进行处理，索引数组中的第 *n* 个顶点即定义了点 *n*，共绘制 *N* 个点。例如，索引数组{0,1,2,3,4}，如图 11-8 所示。

- GL_LINES

把每两个顶点作为一条独立的线段，索引数组中的第 $2n$ 和 $2n+1$ 顶点定义了第 n 线段，总共绘制 $N/2$ 条线段。如果 N 为奇数，则忽略最后一个顶点。例如，索引数组{0, 3, 2, 1}，如图 11-9 所示。



▲图 11-8　GL_POINTS 方式



▲图 11-9　GL_LINES 方式

- GL_LINE_STRIP

绘制索引数组中从第 0 个顶点到最后一个顶点依次相连的一组线段，第 n 和 $n+1$ 个顶点定义了线段 n，总共绘制 $N-1$ 条线段。例如，索引数组{0, 3, 2, 1}，如图 11-10 所示。

- GL_LINE_LOOP

绘制索引数组中从第 0 个顶点到最后一个顶点依次相连的一组线段，最终最后一个顶点与第 0 个顶点。第 n 个和第 $n+1$ 个顶点定义了线段 n，最后一条线段是由顶点 $N-1$ 和 0 之间定义，总共绘制 N 条线段。例如，索引数组{0, 3, 2, 1}，如图 11-11 所示。

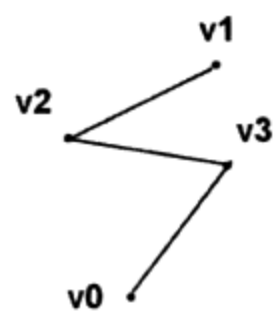
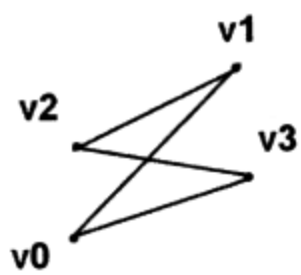


图 11-10　GL_LINE_STRIP 方式



▲图 11-11　GL_LINE_LOOP 方式

提示　n 代表顶点编号，N 代表顶点的个数。索引数组中的值代表顶点数组中的顶点编号。

- GL_TRIANGLES

把索引数组中每 3 个顶点作为一个独立三角形。索引数组中第 $3n$，$3n+1$ 和 $3n+2$ 顶点定义了第 n 个三角形，总共绘制 $N/3$ 个三角形。例如，索引数组{0,1,2,2,1 3}，如图 11-12 所示。

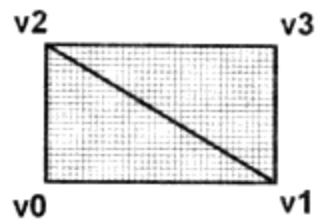
- GL_TRIANGLE_STRIP

绘制一组相连的三角形。对于索引数组中的第 n 个点：若 n 为奇数，第 $n+1$，n 和 $n+2$ 顶点定义了第 n 个三角形；若 n 为偶数，第 n，$n+1$ 和 $n+2$ 顶点定义了第 n 个三角形。总共绘制 $N-2$ 个三角形。例如，索引数组{0,1,2,3,4}，如图 11-13 所示。

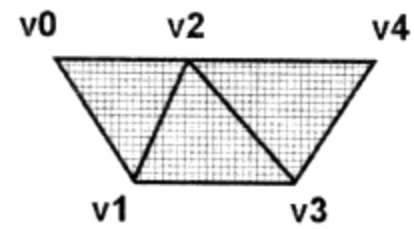
- GL_TRIANGLE_FAN

绘制一组相连的三角形。三角形是由索引数组中第 0 个顶点及其后给定的顶点所确定。顶点 0，

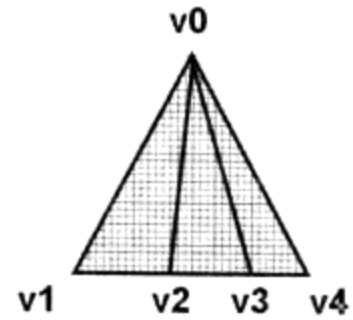
n+1 和 n+2 定义了第 n 个三角形。例如，索引数组{0,1,2,3,4}，如图 11-14 所示。



▲图 11-12　GL_TRIANGLES 方式



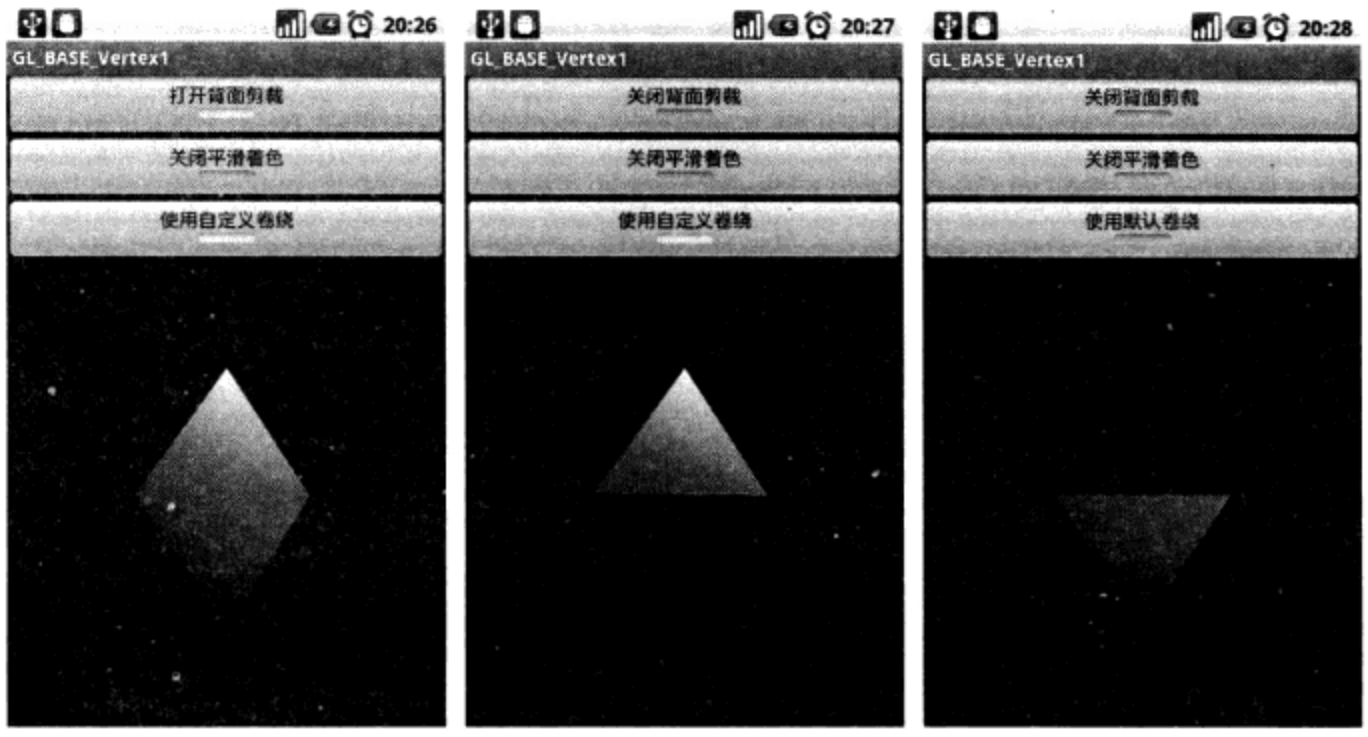
▲图 11-13　GL_TRIANGLE_STRIP 方式



▲图 11-14　GL_TRIANGLE_FAN 方式

11.1.4　用索引法绘制三角形的案例

上一小节介绍了使用索引的不同绘制方式，本小节将给出一个利用索引法绘制三角形的具体案例 SAMPLE_11_1。案例的运行效果如图 11-15 所示。



▲图 11-15　案例 SAMPLE_11_1 的运行效果图

提示　图 11-15 中从左至右依次为案例开始界面、打开背面剪裁并旋转图形的界面、打开背面剪裁后使用自定义旋转绕行方式的界面。

接下来介绍实现本案例的代码。代码中的各个文件的详细讲解如下。

（1）首先介绍布局文件 main.xml 中 3 个按钮，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ SAMPLE_11_1/res/layout 目录下的 main.xml。

```
1    <?xml version="1.0" encoding="utf-8"?>          <!-- 版本号和编码方式-->
2    <LinearLayout xmlns:android=                    <!-- 总的 LinearLayout -->
3    "http://schemas.android.com/apk/res/android"
4        android:orientation="vertical"              <!--设置 linearLayout 布局方向-->
5        android:layout_width="fill_parent"          <!--设置 linearLayout 的布局方式-->
```

```
6        android:layout_height="fill_parent"
7        android:gravity="bottom"                          <!--设置 linearLayout 的重力方向-->
8        android:id="@+id/mainLinear"                      <!--设置 linearLayout 的 id-->
9        >
10         <ToggleButton                                   <!--设置背面剪裁的 ToggleButton-->
11              android:id="@+id/ToggleButton01"           <!--设置 ToggleButton 的 id-->
12              android:layout_width="fill_parent"         <!--设置 ToggleButton 的布局方式-->
13              android:layout_height="wrap_content"
14              android:textOn="关闭背面剪裁"          <!--设置 ToggleButton 上显示的文字-->
15              android:textOff="打开背面剪裁"
16              android:checked="false"                <!--设置 ToggleButton 默认为关闭-->
17              >
18         </ToggleButton>
19         <ToggleButton                              <!--设置平滑着色的 ToggleButton-->
20         android:id="@+id/ToggleButton02"           <!--设置 ToggleButton 的 id-->
21         android:layout_width="fill_parent"         <!--设置 ToggleButton 的布局方式-->
22         android:layout_height="wrap_content"
23         android:textOn="关闭平滑着色"              <!--设置 ToggleButton 上显示的文字-->
24         android:textOff="打开平滑着色"
25         android:checked="false"                    <!--设置 ToggleButton 默认为关闭-->
26         >
27         </ToggleButton>
28         <ToggleButton                              <!--设置旋转绕行方式的 ToggleButton-->
29         android:textOff="使用自定义卷绕"           <!--设置 ToggleButton 上显示的文字-->
30         android:textOn="使用默认卷绕"
31         android:checked="false"                    <!--设置 ToggleButton 默认为关闭-->
32         android:id="@+id/ToggleButton03"           <!--设置 ToggleButton 的 id 属性-->
33         android:layout_width="fill_parent"         <!--设置 ToggleButton 的布局方式-->
34         android:layout_height="wrap_content"
35         >
36         </ToggleButton>
37   </LinearLayout>
```

- 代码第 3-9 行定义了一个 LinearLayout 布局，在布局中设置了方向、方式、重力及 id 等属性。
- 代码第 10-36 行定义了 3 个 ToggleButton，分别用于设置本案例中的用来背面剪裁、平滑着色及旋转绕行方式，各个 ToggleButton 的属性基本相同。

（2）MyGLActivity 类的开发。该类是本案例的主控制类，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ SAMPLE_11_1 /src/com/bn/game/chap11/ex1 目录下的 MyGLActivity.java。

```
1    package com.bn.game.chap11.ex1;                        //声名包
2    import android.app.Activity;                           //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.widget.CompoundButton.OnCheckedChangeListener; //引入相关类
5    public class MyGLActivity extends Activity {
6         MyGLView myGLView;                                //MyGLView 的引用
7       public void onCreate(Bundle savedInstanceState) {
8          super.onCreate(savedInstanceState);
9          setContentView(R.layout.main);                   //设置主界面
10         myGLView=new MyGLView(this);                     //创建 myGLView
11         LinearLayout ll=(LinearLayout) this.findViewById(R.id.mainLinear);
12         ll.addView(myGLView);                            //在 LinearLayout 中加入 myGLView
13         ToggleButton btn1=(ToggleButton) this.findViewById(R.id.ToggleButton01);
14         btn1.setOnCheckedChangeListener(                 //为背面剪裁按钮添加监听
15                new OnCheckedChangeListener() {           //创建监听器
16                       public void onCheckedChanged(CompoundButton buttonView,Boolean
```

```
                        isChecked) {
17                          myGLView.setCullFaceFlag(!myGLView.isCullFaceFlag());
18                      }
19                  }
20              );
21          ToggleButton btn2=(ToggleButton) this.findViewById(R.id.ToggleButton02);
22          btn2.setOnCheckedChangeListener(            //为平滑着色按钮添加监听
23                  new OnCheckedChangeListener() {     //创建监听器
24                      public void onCheckedChanged(CompoundButton buttonView,boolean
                        isChecked) {
25                          myGLView.setSmoothFlag(!myGLView.isSmoothFlag());
26                      }
27                  }
28              );
29          ToggleButton btn3=(ToggleButton) this.findViewById(R.id.ToggleButton03);
30          btn3.setOnCheckedChangeListener(            //为旋转绕行方式按钮添加监听
31                  new OnCheckedChangeListener() {     //创建监听器
32                      public void onCheckedChanged(CompoundButton buttonView,boolean
                        isChecked) {
33                          myGLView.setCwFlag(!myGLView.isCwFlag());
34                      }
35                  }
36              );
37  }}
```

- 代码第 9-12 行为把 main.xml 文件设置为界面内容后，获得主 LinearLayout 的引用，并将创建的 MyGLView 类的对象添加到界面中。
- 代码第 13-36 行为给界面中的 3 个 ToggleButton 按钮添加监听，使各个按钮能控制 MyGLView 界面中的对应 3 个变量。

（3）主界面类 MyGLView 的开发。该类是案例中的主界面类，负责主界面中各种资源的初始化和界面中各对象的绘制工作，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ SAMPLE_11_1 /src/com/bn/game/chap11/ex1 目录下的 MyGLView.java。

```
1   package com.bn.game.chap11.ex1;                                  //声名包
2   import javax.microedition.khronos.egl.EGLConfig;                 //引入相关类
3   ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4   import android.view.MotionEvent;                                 //引入相关类
5   public class MyGLView extends GLSurfaceView{
6       private final float TOUCH_SCALE_FACTOR = 180.0f/480;//角度缩放比例
7       private SceneRenderer renderer;                              //场景渲染器
8       private float previousX;                                     //上次的触控位置
9       private float previousY;
10      private boolean cullFaceFlag=false;                          //背面剪裁标志位
11      private boolean smoothFlag=false;                            //平滑着色标志位
12      private boolean cwFlag=false;                                //自定义旋转绕行标志位
13      public MyGLView(Context context) {
14          super(context);
15          renderer=new SceneRenderer();                            //创建渲染器
16          this.setRenderer(renderer);                              //设置渲染器
17          this.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
                                                                     //设置为主动渲染
18      }
19    public boolean onTouchEvent(MotionEvent e) {                   //触摸事件回调方法
20        float y = e.getY();                                        //获得触摸事件发生的位置
21        float x = e.getX();
22        switch (e.getAction()) {
```

```
23          case MotionEvent.ACTION_MOVE:
24              float dy = y - previousY;                                  //计算触控笔 Y 位移
25              float dx = x - previousX;                                  //计算触控笔 X 位移
26              renderer.diamond.yAngle += dx * TOUCH_SCALE_FACTOR; //设置沿 y 轴旋转角度
27              renderer.diamond.xAngle+= dy * TOUCH_SCALE_FACTOR;  //设置沿 x 轴旋转角度
28              requestRender();                                           //重绘画面
29          }
30          previousY = y;                                                 //记录触控笔位置
31          previousX = x;                                                 //记录触控笔位置
32          return true;
33      }
34        private class SceneRenderer implements GLSurfaceView.Renderer {
35              Diamond diamond=new Diamond(1,1,1.5f);
36              public SceneRenderer(){}
37              public void onDrawFrame(GL10 gl) {
38              if(cullFaceFlag){                                          //背面剪裁标志位
39                  gl.glEnable(GL10.GL_CULL_FACE);                        //设置为打开背面剪裁
40              }
41              else{
42                  gl.glDisable(GL10.GL_CULL_FACE);                       //设置为关闭背面剪裁
43              }
44              if(smoothFlag){                                        //平滑着色标志位
45                 gl.glShadeModel(GL10.GL_SMOOTH);                    //设置着色模型为平滑着色
46              }
47              else{
48                  gl.glShadeModel(GL10.GL_FLAT);                     //设置着色模型为不平滑着色
49              }
50              if(cwFlag){                                            //自定义旋转绕行标志位
51                  gl.glFrontFace(GL10.GL_CW);  //设置为自定义旋转绕行顺序——顺时针为正面
52              }
53              else{
54                  gl.glFrontFace(GL10.GL_CCW); //设置为默认旋转绕行顺序——逆时针为正面
55              }
56              gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
57              gl.glMatrixMode(GL10.GL_MODELVIEW);                    //设置当前矩阵为模式矩阵
58              gl.glLoadIdentity();                                   //设置当前矩阵为单位矩阵
59              gl.glTranslatef(0, 0f, -4f);
60              diamond.drawSelf(gl);
61              }
62              public void onSurfaceChanged(GL10 gl, int width, int height) {
63              gl.glViewport(0, 0, width, height);                    //设置视窗大小及位置
64              gl.glMatrixMode(GL10.GL_PROJECTION);                   //设置当前矩阵为投影矩阵
65              gl.glLoadIdentity();                                   //设置当前矩阵为单位矩阵
66              float ratio = (float) width / height;                  //计算透视投影的比例
67              gl.glFrustumf(-ratio, ratio, -1, 1, 1.5f, 10);
                                                        //调用此方法计算产生透视投影矩阵
68              }
69              public void onSurfaceCreated(GL10 gl, EGLConfig config) {
70              gl.glDisable(GL10.GL_DITHER);                          //关闭抗抖动
71              gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
72              gl.glClearColor(0,0,0,0);                              //设置屏幕背景色黑色 RGBA
73              gl.glEnable(GL10.GL_DEPTH_TEST);                       //启用深度测试
74              }
75          }
76          ……//该处省略了部分成员方法，读者可自行查看随书光盘的源代码
77  }
```

- 代码第 6-12 行为该类中成员变量的定义，其中 10-12 行中 3 个 boolean 型变量的值是由 Activity 中添加的按钮监听来控制的。
- 代码第 34-75 行为内部类场景渲染器类的定义，渲染场景的工作都是在这个类中进行的。

定义场景渲染器只需要实现 Renderer 接口和接口中的抽象方法。

- 代码第 62-75 行为绘制之前的初始化的代码，读者不必担心，这些初始化的工作将在后面章节中给出详细介绍，在这只通过注释了解其大意即可。
- 代码第 35 行创建了 Diamond 类的一个对象，Diamond 类就是主界面中的绘制的菱形，该类的具体实现将在后面小节中给出。
- 代码第 37-61 行是负责绘制的方法 onDrawFrame 的实现。其中第 38-55 行为 3 个 boolean 型的成员变量控制的 3 种绘制方式。
- 代码第 56 行是清除颜色缓存与深度缓存。OpenGL ES 保存了一系列缓存（buffers），即用于绘制图各方面的内存块。颜色缓存保存当前帧各像素的颜色。基本上就是用户在屏幕上看到的情况。深度缓存保存每个潜在像素离观察者距离的信息。使用此信息可以确定一个像素是否需要被绘制出来。这两个缓存是 OpenGL ES 中最常用的缓存。还有其他类型的一些缓存，如模板缓存和累积缓存等。
- 代码第 57-58 行为加载单位变换矩阵的代码。此工作将清除虚拟世界中的一切平衡、旋转、缩放或其他变化，并将观察都置于原点。
- 代码第 19-33 行为监听触控事件方法的实现。该方法中通过触控两次的位置之差来改变 diamond 对象的沿相应轴的旋转角度，使界面中的菱形旋转。

（4）Diamond 类的实现。该类是主界面中的绘制的菱形，在该类中才真正体现了用索引法绘制图形的方法。其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ SAMPLE_11_1 /src/com/bn/game/chap11/ex1 目录下的 Diamond.java。

```
1    package com.bn.game.chap11.ex1;                                  //声名包
2    import java.nio.ByteBuffer;                                      //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import javax.microedition.khronos.opengles.GL10;                 //引入相关类
5    public class Diamond {
6         private FloatBuffer   vertexBuffer;                         //顶点坐标数据缓冲
7       private FloatBuffer   colorBuffer;                            //顶点着色数据缓冲
8       private ByteBuffer  indexBuffer;                              //顶点构建索引数据缓冲
9       int vCount=0;                                                 //顶点个数
10      int iCount=0;
11      float size;                                                   //尺寸
12      float xAngle=0;                                               //绕 x 轴旋转的角度
13      float yAngle=0;                                               //绕 y 轴旋转的角度
14      float zAngle=0;                                               //绕 z 轴旋转的角度
15        public Diamond(float scale,float a, float b) {
16             size=Constant.UNIT_SIZE*scale;
17             vCount=3;
18             //坐标数据初始化
19             float[] vertices=new float[]{                          //存放顶点坐标的数组
20                  -a*size, 0*size, 0*size,
21                  0*size, -b*size, 0*size,
22                  a*size, 0*size, 0*size,
23                  0*size, b*size, 0*size,
24             };
25             ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                  //创建顶点坐标数据缓冲
26          vbb.order(ByteOrder.nativeOrder());                       //设置字节顺序
27          vertexBuffer = vbb.asFloatBuffer();                       //转换为 float 型缓冲
```

```
28          vertexBuffer.put(vertices);                //向缓冲区中放入顶点坐标数据
29          vertexBuffer.position(0);                                   //设置缓冲区起始位置
30          //着色数据的初始化
31          float[] colors=new float[]{                //顶点颜色值数组，每个顶点 4 个色彩值 RGBA
32                  1,0,0,0,
33                  0,0,1,0,
34                  0,1,0,0,
35                  1,1,1,0,
36          };
37          ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length*4);
                                                       //创建顶点着色数据缓冲
38          cbb.order(ByteOrder.nativeOrder());                         //设置字节顺序
39          colorBuffer = cbb.asFloatBuffer();                          //转换为 float 型缓冲
40          colorBuffer.put(colors);                   //向缓冲区中放入顶点着色数据
41          colorBuffer.position(0);                                    //设置缓冲区起始位置
42          //三角形构造索引数据初始化
43          iCount=6;
44          byte indexes[]=new byte[]{                                  //三角形索引数组
45              0,2,3,                                                  //逆时针
46              0,2,1,                                                  //顺时针
47          };
48          indexBuffer = ByteBuffer.allocateDirect(indexes.length);
                                                       //创建三角形构造索引数据缓冲
49          indexBuffer.put(indexes);                  //向缓冲区中放入三角形构造索引数据
50          indexBuffer.position(0);                                    //设置缓冲区起始位置
51        }
52      public void drawSelf(GL10 gl){
53        gl.glPushMatrix();                                            //将当前矩阵入栈
54          gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);               //启用顶点坐标数组
55          gl.glEnableClientState(GL10.GL_COLOR_ARRAY);                //启用顶点颜色数组
56          gl.glRotatef(xAngle, 1, 0, 0);                              //绕轴旋转
57          gl.glRotatef(yAngle, 0, 1, 0);
58          gl.glRotatef(zAngle, 0, 0, 1);
59          gl.glVertexPointer                         (//为画笔指定顶点坐标数据
60                  3,                                 //每个顶点的坐标数量为 3  xyz
61                  GL10.GL_FLOAT,                     //顶点坐标值的类型为 GL_FLOAT
62                  0,                                 //连续顶点坐标数据之间的间隔
63                  vertexBuffer                                        //顶点坐标数据
64          );
65          gl.glColorPointer(                         //为画笔指定顶点着色数据
66                  4,                                 //设置颜色的组成成分，必须为 4—RGBA
67                  GL10.GL_FLOAT,                     //顶点颜色值的类型为 GL_FLOAT
68                  0,                                 //连续顶点着色数据之间的间隔
69                  colorBuffer                                         //顶点着色数据
70          );
71          gl.glDrawElements(                                          //绘制图形
72                  GL10.GL_TRIANGLES,                                  //以三角形方式填充
73                  iCount,                                             //顶点个数
74                  GL10.GL_UNSIGNED_BYTE,                              //索引值的尺寸
75                  indexBuffer                                         //索引值数据
76          );
77          gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);              //禁用顶点坐标数组
78          gl.glDisableClientState(GL10.GL_COLOR_ARRAY);               //禁用顶点颜色数组
79          gl.glPopMatrix();                                           //恢复当前矩阵
80    }}
```

- 代码第 6-14 行为该类中成员变量的定义。其中包括顶点、颜色和索引的数据缓冲，以及顶点数等其他属性值。

- 代码第 19-29 行为坐标数据的初始化。其中顶点数组中存放的是菱形各个顶点的坐标值，在索引法中，有多少个顶点，相应就有多少个坐标与之相对应。
- 代码第 31-41 行为着色数据的初始化，这里不注重讲颜色，只是为每个顶点随便设置了一种颜色，读者也不必刻意去研究。
- 代码第 43-50 行为三角形构造索引数据的初始化，在其中索引数组中，两个三角形的顶点索引 0,2,3 和 0,2,1 一个为逆时针旋转绕行，一个为顺时针旋转绕行，所以，当开户背面剪裁功能时，只能看见其中一个三角形。
- 代码第 52-80 行为绘制菱形的 drawSelf 方法的实现。该方法为绘制图形指定了顶点、颜色和索引数据缓冲，在 72 行绘制图形时指定填充方式为以三角形方式填充(GL10.GL_TRIANGLES)，并指定其索引值数据，便实现了索引法绘制图形。

提示 在本章的案例中，由于篇幅所限，只详细讲解本节中该案例，其他案例只给出了讲解时所需的核心代码，其他没有讲到的代码，读者可自行查看随书光盘的源代码。

11.1.5 不使用索引数据绘制

前面的案例在进行图形渲染时采用了灵活性较强的顶点索引方式，其实不采用顶点索引直接采用顶点坐标数据也可以进行绘制。下面给出了上述案例的另一个版本，其效果和上述案例一样。其部分核心代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_2 /src/com/bn/game/chap11/ex2 目录下的 Diamond.java。

```
1    public class Diamond {
2         private FloatBuffer  vertexBuffer;                    //顶点坐标数据缓冲
3       private FloatBuffer  colorBuffer;                       //顶点着色数据缓冲
4       int vCount=0;                                           //顶点个数
5       int iCount=0;
6       float size;                                             //尺寸
7       float xAngle=0;                                         //绕 x 轴旋转的角度
8       float yAngle=0;                                         //绕 y 轴旋转的角度
9       float zAngle=0;                                         //绕 z 轴旋转的角度
10        public Diamond(float scale,float a, float b) {
11             size=Constant.UNIT_SIZE*scale;
12             vCount=6;
13             float[] vertices=new float[]{                    //坐标数据初始化
14                  -a*size, 0*size, 0*size,                    //0
15                  a*size, 0*size, 0*size,                     //2
16                  0*size, b*size, 0*size,                     //3
17                  -a*size, 0*size, 0*size,                    //0
18                  a*size, 0*size, 0*size,                     //2
19                  0*size, -b*size, 0*size,                    //1
20             };
21             ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                  //创建顶点坐标数据缓冲
22          vbb.order(ByteOrder.nativeOrder());                 //设置字节顺序
23          vertexBuffer = vbb.asFloatBuffer();                 //转换为 float 型缓冲
24          vertexBuffer.put(vertices);           //向缓冲区中放入顶点坐标数据
25          vertexBuffer.position(0);                           //设置缓冲区起始位置
```

```
26          //着色数据的初始化
27          float[] colors=new float[]{            //顶点颜色值数组，每个顶点 4 个色彩值 RGBA
28                  1,0,0,0,                                                //0
29                  0,1,0,0,                                                //2
30                  1,1,1,0,                                                //3
31                  1,0,0,0,                                                //0
32                  0,1,0,0,                                                //2
33                  0,0,1,0,                                                //1
34          };
35          ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length*4);
                                                   //创建顶点着色数据缓冲
36          cbb.order(ByteOrder.nativeOrder());                             //设置字节顺序
37          colorBuffer = cbb.asFloatBuffer();                              //转换为 float 型缓冲
38          colorBuffer.put(colors);               //向缓冲区中放入顶点着色数据
39          colorBuffer.position(0);                                        //设置缓冲区起始位置
40        }
41      public void drawSelf(GL10 gl){
42        gl.glPushMatrix();
43          gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);                   //启用顶点坐标数组
44          gl.glEnableClientState(GL10.GL_COLOR_ARRAY);                    //启用顶点颜色数组
45          gl.glRotatef(xAngle, 1, 0, 0);                                  //绕轴旋转
46          gl.glRotatef(yAngle, 0, 1, 0);
47          gl.glRotatef(zAngle, 0, 0, 1);
48          gl.glVertexPointer(                    //为画笔指定顶点坐标数据
49                  3,                             //每个顶点的坐标数量为 3  xyz
50                  GL10.GL_FLOAT,                 //顶点坐标值的类型为 GL_FLOAT
51                  0,                             //连续顶点坐标数据之间的间隔
52                  vertexBuffer                                            //顶点坐标数据
53          );
54          gl.glColorPointer(                     //为画笔指定顶点着色数据
55                  4,                             //设置颜色的组成成分，必须为 4—RGBA
56                  GL10.GL_FLOAT,                 //顶点颜色值的类型为 GL_FLOAT
57                  0,                             //连续顶点着色数据之间的间隔
58                  colorBuffer                                             //顶点着色数据
59          );
60          gl.glDrawArrays(                                                //绘制图形
61                  GL10.GL_TRIANGLES,                                      //以三角形方式填充
62                  0,                                                      //开始点编号
63                  vCount                                                  //顶点的数量
64          );
65          gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);                  //禁用顶点坐标数组
66          gl.glDisableClientState(GL10.GL_COLOR_ARRAY);                   //禁用顶点颜色数组
67          gl.glPopMatrix();
68    }}
```

- 代码第 2-9 行成员变量的定义中没有索引数据缓冲，因为直接采用顶点坐标数据绘制图形时不需要顶点的索引，旋转绕行顺序是由给出顶点的顺序决定的。
- 代码第 12-20 行中对坐标数据初始化时，直接给出顶点坐标和其旋转绕行顺序，因此，在同一位置的顶点有时会有必要的重复。
- 代码第 60-64 行中采用 glDrawArrays 方法绘制图形，参数 GL10.GL_TRIANGLES 表明仍以三角形方式填充。

11.2 正交投影和透视投影

经过前面的介绍，读者应该已经知道所谓三维系统最终还是要在二维的显示设备上显示，而将

三维的模型在二维的设备上显示就需要进行投影。本节将介绍两种投影方式：正交投影和透视投影。

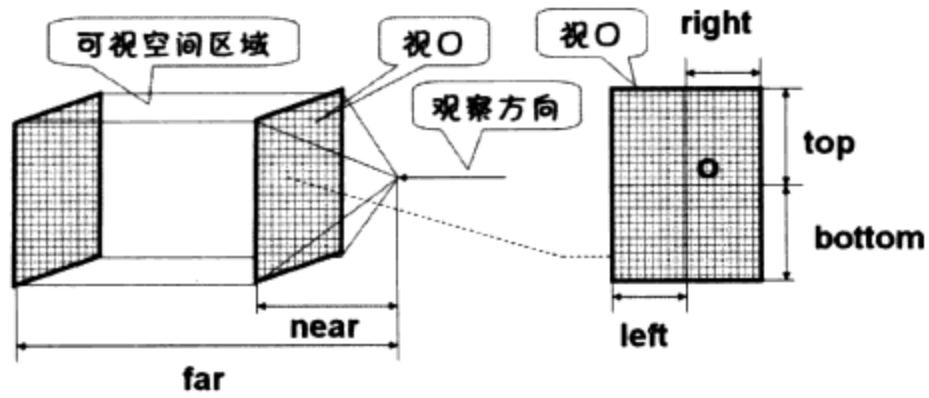
在设置投影模式之前首先要加载一下投影矩阵，代码如下。

```
1          gl.glMatrixMode(GL10.GL_PROJECTION);        //设置当前矩阵为投影矩阵
2          gl.glLoadIdentity();                        //设置当前矩阵为单位矩阵
```

下面将对这两种投影进行详细介绍。

11.2.1 正交投影

正交投影是平等投影的一种，观察者的视线是平行的，不产生真实世界近大远小的透视效果。正交投影中，可视的空间区域为长方体，图 11-16 所示为正交投影的示意图。



▲图 11-16 正交投影示意图

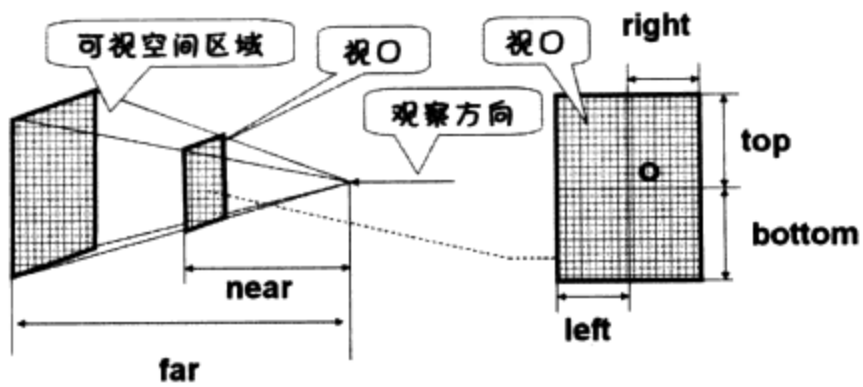
设置正交投影的语句为：

```
1          gl.glOrthof(left, right, bottom, top, near, far);        //设置为正交投影
```

其中 Left 与 Right 为视口左右侧对应的 x 坐标，bottom 与 top 为视口上下侧对应的 y 坐标，near 和 far 为可视空间区域最近端与最远端的 z 坐标。要注意的是，OpenGL ES 不支持无限可视区域。视口可以被看作手机屏幕上的指定矩形区域。

11.2.2 透视投影

透视投影属于非平行投影，观察者的视线在远处是相交的，视线相交处即为灭点。通过透视投影，可以产生现实世界中近大远小的效果。因此，使用透视投影可以得到更加真实的 3D 体验，游戏中较多采用的就是透视投影。透视投影的示意图如图 11-17 所示。



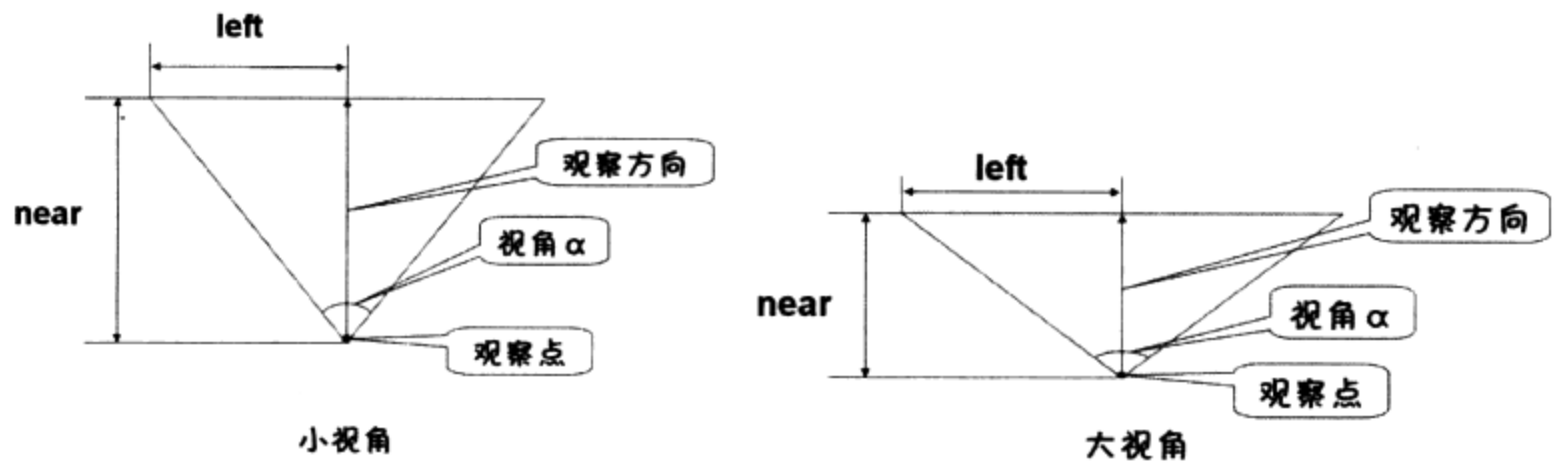
▲图 11-17 透视投影示意图

设置透视投影的语句为：

```
1          gl.glFrustumf(left, right, bottom, top, near, far);      //设置为透视投影
```

其中 Left 与 Right 为视口左右侧对应的 x 坐标，bottom 与 top 为视口上下侧对应的 y 坐标，near 和 far 为可视空间区域最近端与最远端的 z 坐标。要注意的是，OpenGL ES 不支持无限可视区域。视口可以被看作手机屏幕上的指定矩形区域。

同时还存在视角的问题，这就和照相机的镜头分广角的、普通和长焦的一样。大视角情况下，在同样的距离上可以看到更广阔范围内的内容，如图 11-18 所示。



▲图 11-18　视角问题示意图

从上面两幅图中可以看出，左边的情况下视角小，右边情况下视角大。视角的计算公式如下：

α =2arctg(left/near)。

上述公式为水平方向的视角，垂直方向的视角公式为：

α =2arctg(top/near)。

提示　上述两个计算公式是在左右或上下对称的情况下推导出的，若左右或上下不对称，则两个半角要分别计算。

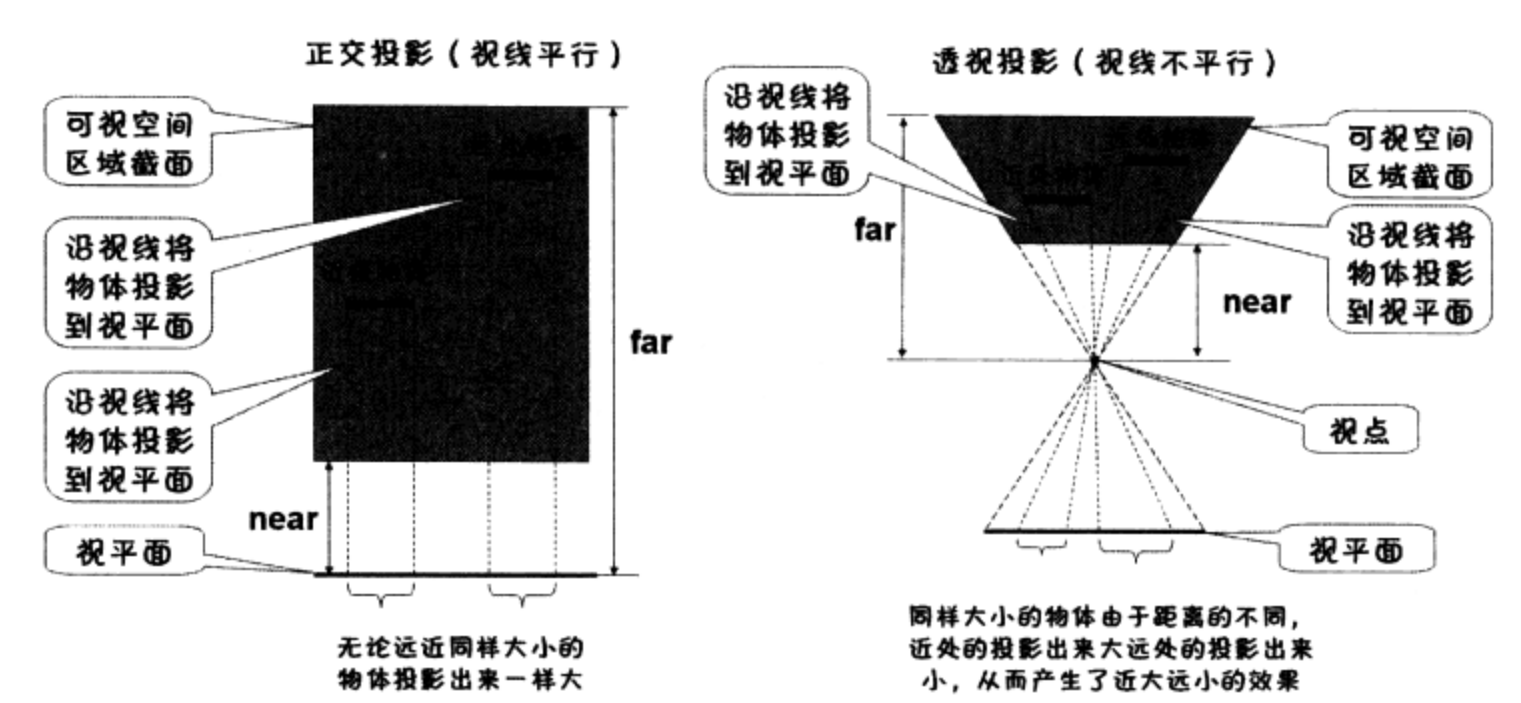
可以从上面的情况总结出如下的简单规律，在 left、right、top、bottom 值不变的情况下，near 值越小，视角越大。在实际开发中，对 left、right、bottom、top、near、far 6 个值的调整是非常重要的，合适的值组可以产生很好的效果，不合适的值组甚至有可能影响内容的正确显示。

11.2.3　两种投影方式的原理及视口

前面小节中已经提到过正交投影不产生近大远小的效果，而透视投影会产生类似人通过肉眼观察现实世界近大远小的效果，其原理可用图 11-19 来说明。

说明　图 11-19 中远处物体和近处物体的大小是完全一样的，只有保证大小一样，才能说明投影是否能产生近大远小的效果。

前面小节的介绍中一直用到了视口的概念，也知道了视口就可以看作手机屏幕上的指定矩形区域。一般情况下，在游戏娱乐应用程序中，视口往往占用整个屏幕。

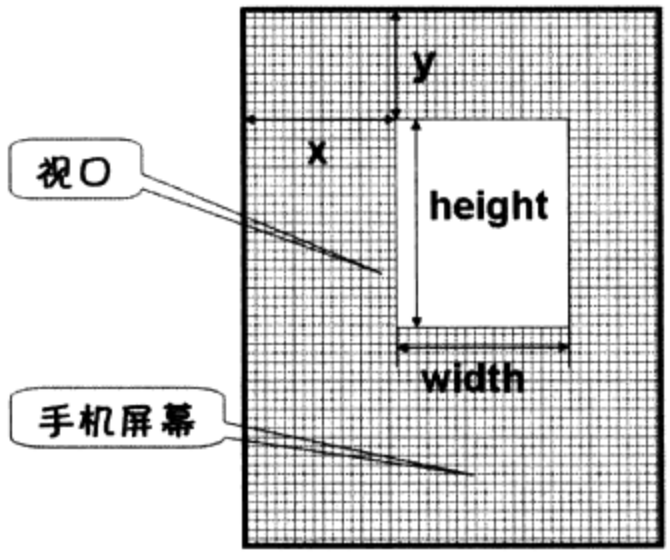


▲图 11-19　两种投影原理图

OpenGL ES 中视口的设置非常简单，使用如下语句：

```
1    Gl.glViewPort(x, y, wiidth, height);                    //设置视口
```

如图 11-20 所示，x 和 y 为视口矩形在屏幕上左上侧点的坐标，width 和 height 为视口的宽度与高度。

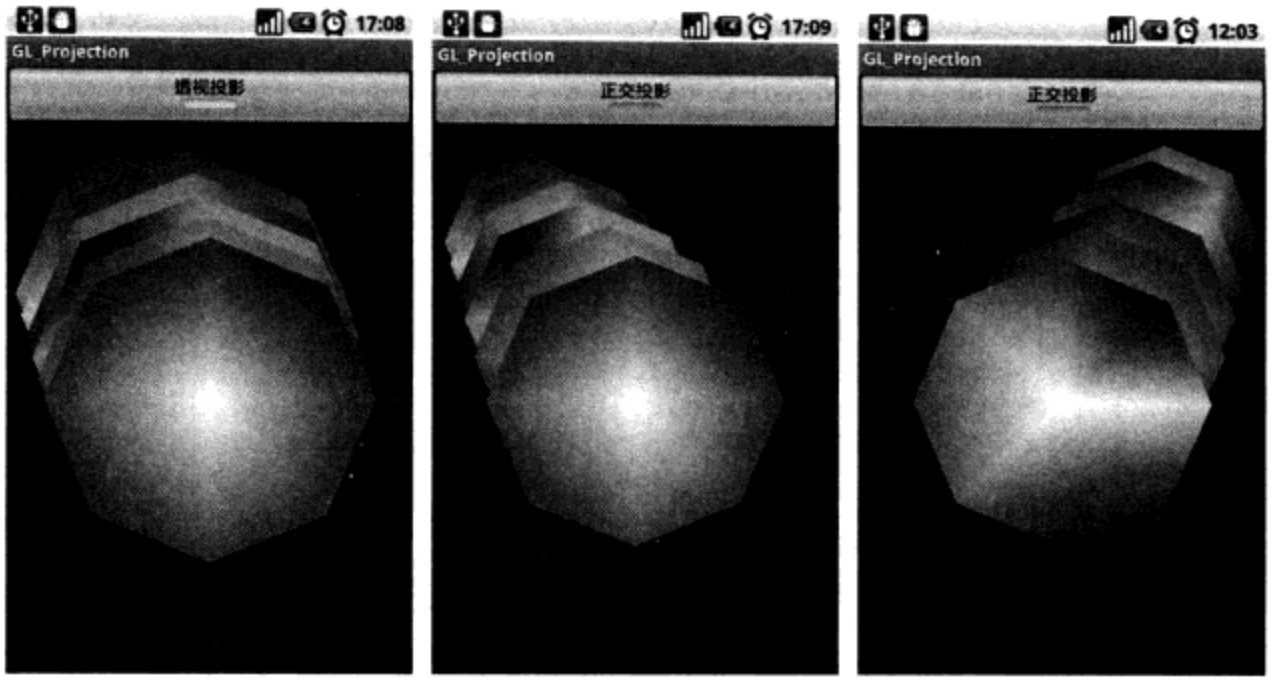


▲图 11-20　视口示意图

11.2.4　两种投影方式的案例

前面小节中详细介绍了两种投影模式，为了使读者对投影模式的理解更加深刻，本小节将给出一个使用了两种不同投影模式的案例 GL_Projection，案例的运行效果如图 11-21 所示。

提示　图 11-21 左面的一幅图为正交投影的效果图，右边的两幅图为透视投影的效果图，上侧的按钮是两种投影模式的切换按钮。



▲图 11-21　案例 GL_Projection 的运行效果图

接下来介绍实现本案例的代码。在这里只讲解有关两种投影模式的核心类 MyGLView，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_3 /src/com/bn/game/chap11/ex3 目录下的 MyGLView.java。

```
1    package com.bn.game.chap11.ex3;                                    //声名包
2    import javax.microedition.khronos.egl.EGLConfig;                   //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.view.MotionEvent;                                   //引入相关类
5    public class MyGLView extends GLSurfaceView{
6         private final float TOUCH_SCALE_FACTOR = 30.0f/480;           //角度缩放比例
7         private SceneRenderer renderer;                               //场景渲染器
8         private float previousX;                                      //上次的触控位置
9         private float previousY;
10        private boolean perspectiveFlag=false;                        //透视投影标志位
11        float ratio;                                                  //屏幕宽高比
12        float xAngle;
13        float yAngle;
14        public MyGLView(Context context) {                            //构造器
15             super(context);
16             renderer=new SceneRenderer();                            //创建渲染器
17             this.setRenderer(renderer);                              //设置渲染器
18             this.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);//设置主动渲染
19        }
20       public boolean onTouchEvent(MotionEvent e) {                   //触摸事件回调方法
21        ……//该处省略了 onTouchEvent 的方法体，读者可自行查看随书光盘的源代码
22       }
23        private class SceneRenderer implements GLSurfaceView.Renderer {
24             Circle[] circles=new Circle[6];                          //所有八边形的数组
25             public SceneRenderer(){}
26             public void onDrawFrame(GL10 gl) {
27             gl.glMatrixMode(GL10.GL_PROJECTION);              //设置当前矩阵为投影矩阵
28             gl.glLoadIdentity();                              //设置当前矩阵为单位矩阵
29                  if(perspectiveFlag){
30                gl.glFrustumf(-ratio, ratio, -1, 1, 13.5f, 100);//产生透视投影矩阵
31             }
```

```
32              else{
33               gl.glOrthof(-ratio, ratio, -1, 1, 1.5f, 100);  //产生正交投影矩阵
34               }
35               gl.glEnable(GL10.GL_CULL_FACE);                //设置为打开背面剪裁
36               gl.glShadeModel(GL10.GL_SMOOTH);               //设置着色模型为平滑着色
37               gl.glFrontFace(GL10.GL_CCW);       //设置为默认旋转绕行顺序——逆时针为正面
38               gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
39              gl.glMatrixMode(GL10.GL_MODELVIEW);             //设置当前矩阵为模式矩阵
40              gl.glLoadIdentity();                            //设置当前矩阵为单位矩阵
41              gl.glTranslatef(0, 0f, -15f);
42              gl.glRotatef(xAngle, 1, 0, 0);
43              gl.glRotatef(yAngle, 0, 1, 0);
44              for(Circle c:renderer.circles){                 //绘制所有正八边形
45               c.drawSelf(gl);
46               }
47               }
48               public void onSurfaceChanged(GL10 gl, int width, int height) {
49               gl.glViewport(0, 0, width, height);            //设置视窗大小及位置
50              gl.glMatrixMode(GL10.GL_PROJECTION);            //设置当前矩阵为投影矩阵
51              gl.glLoadIdentity();                            //设置当前矩阵为单位矩阵
52              ratio = (float) width / height;                 //计算透视投影的比例
53               }
54               public void onSurfaceCreated(GL10 gl, EGLConfig config) {
55               gl.glDisable(GL10.GL_DITHER);                  //关闭抗抖动
56              gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
57              gl.glClearColor(0,0,0,0);                       //设置屏幕背景色黑色 RGBA
58              gl.glEnable(GL10.GL_DEPTH_TEST);                //启用深度测试
59              for(int i=0;i<circles.length;i++){              //创建所有圆
60               circles[i]=new Circle(0.3f,1,8,-i*3f);
61               }
62           }
63       }
64       ……//该处省略了部分成员方法，读者可自行查看随书光盘的源代码
65   }
```

- 代码第 6-13 行为该类中成员变量的定义，其中 boolean 型的变量 perspectiveFlag 是控制两种投影模式切换的标志位。
- 代码第 24 行在 SceneRenderer 类中定义了 Circle 类的数组，Circle 类是自定义的圆类，可以为其指定切分的边数，切分的边数越多，就越接近圆，本案例只给定边数为 8，所以绘制出来是一个正八边形。
- 代码第 27-34 行加载了投影矩阵后，根据变量 perspectiveFlag 的值调用相应设置投影矩阵的方法，故在按钮监听时改变变量 perspectiveFlag 的值即可改变矩阵的投影方式。

11.3 光照与材质

经过前面的介绍读者应该已经可以绘制出较简单的三维图形。本小节将介绍 OpenGL ES 系统中光照和材质的相关知识，通过本节的学习，读者可以开发更加绚丽的 3D 场景。

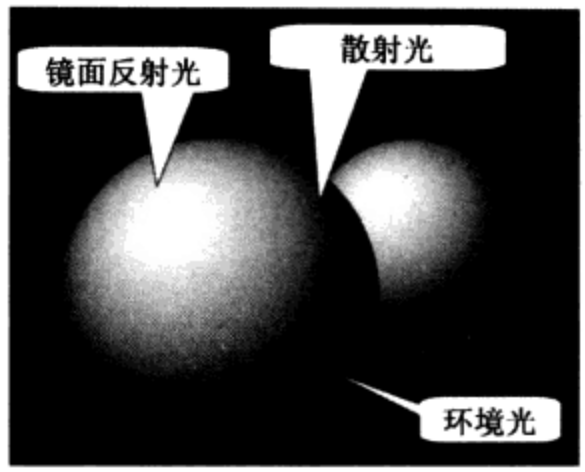
11.3.1 光照的 3 种组成元素

默认情况下，在 OpenGL ES 系统中灯光是关闭的，要想使用灯光必须首先允许灯光并打开相应编号的灯，代码如下。

```
1        gl.glEnable(GL10.GL_LIGHTING);                          //允许光照
2        gl.glEnable(GL10.GL_LIGHT0);                            //打开 0 号灯
```

提示　OpenGL ES 中最多允许 8 盏灯，id 从“GL10.GL_LIGHT0”到“GL10.GL_LIGHT7”。

OpenGL ES 中每盏灯的光分为 3 种组成元素：环境光（AMBIENT）、散射光（DIFFUSE）和镜面反射光（SPECULAR），如图 11-22 所示。



▲图 11-22　灯光的三种组成元素示意图

下面对三种光一一进行介绍。

（1）环境光是经过环境多次散射，来自四面八方的光。OpenGL ES 中的每盏灯也都支持环境光。用下列代码设置。

```
1        float[] ambientParams={0.1f,0.1f,0.1f,1.0f};            //光参数 RGBA
2        gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_AMBIENT, ambientParams,0);//设置环境光
```

提示　代码第 2 行的方法中，参数的含意为：GL10.GL_LIGHT0 是灯的 ID，GL10.GL_AMBIENT 是指设置的是环境光，ambientParams 数组给出的是 RGBA 4 个色彩通道的值，0 给出的是第一个色彩通道的值在数组中的偏移量。

（2）散射光来自于同一个方向，当其垂直地照射到物体表面时要比斜照时亮。然而散射光到达表面后将沿各个方向均匀的漫反射。因此，各个方向看到的反射光变化不大。OpenGL ES 中的每盏灯也都支持散射光。用下列代码设置。

```
1        float[] diffuseParams={1.0f,1.0f,1.0f,1.0f};            //光参数 RGBA
2        gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE, diffuseParams,0);    //设置散射光
```

提示　代码第 2 行的方法中参数的含意为：GL10.GL_LIGHT0 是灯的 ID，GL10.GL_DIFFUSE 是指设置的是散射光，diffuseParams 数组给出的是 RGBA 4 个色彩通道的值，0 给出的是第一个色彩通道的值在数组中的偏移量。

（3）镜面反射光来自于特定的方向，但也会被反射到特定的方向。反射光的强弱是由表面的反射率决定的。OpenGL ES 中的每盏灯也都支持镜面反射光。用下列代码设置。

```
1        float[] specularParams={1.0f,1.0f,1.0f,1.0f};           //光参数 RGBA
2        gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_SPECULAR, specularParams,0);
```

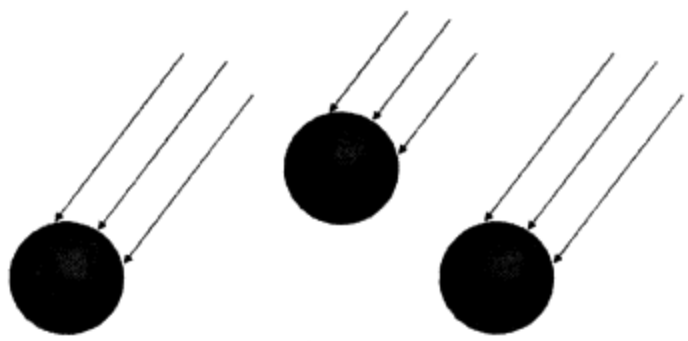
```
                                                                    //设置镜面反射光
```

提示 代码第 2 行的方法中参数的含意为：GL10.GL_LIGHT0 是灯的 ID，GL10.GL_SPECULAR 是指设置的是镜面反射光，specularParams 数组给出的是 RGBA 4 个色彩通道的值，0 给出的是第一个色彩通道的值在数组中的偏移量。

11.3.2 定向光与定位光

为光源设定好了 3 种光后就可以设置此光源的位置或光线方向信息了。请读者注意的是 OpenGL ES 中光源分为两种：定向光（directional）和定位光（positional）。下面对两种光一一进行介绍。

（1）定向光对应的是光源在无穷远处的光，例如，照射到地球上的太阳光就属于定向光，定向光在空间中所有的位置方向都是相同的，定向光的示意图如图 11-23 所示。



▲图 11-23 定向光示意图

指定定向光光源光线方向的基本代码如下：

```
1          float[] directionParams={0,offset,0,0};                    //定向光参数
2          gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, directionParams,0);
                                                                      //设定定向光的方向
```

- directionParams 数组中前 3 个值组成表示定向光方向的向量，光的方向为从向量点处向原点处照射，上述向量表示的是沿 y 轴负方向的光。最后的 0 表示此光源发出定向光。若向量点的位置为 [0,0,0]，则光方向向量的起点与终点重合，此时不能确定光线的方向，实际运行时表现为灯不亮。
- glLightfv 方法中各个参数含义为：GL10.GL_LIGHT0 是灯的 ID，GL10.GL_POSITION 表示是设置光源的位置或光线的方向，directionParams 给出光源的具体信息，0 给出的是第一个值在数组中的偏移量。

（2）定位光类似于现实生活中的打火机光源，其在固定的某个位置，发出的光向四周发散，定位光的示意图如图 11-24 所示。

指定定位光光源光线方向的基本代码如下：

```
1          float[] positionParams={0,0,1,1};                          //定位光参数
2          gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, positionParams,0);
                                                                      //设定定位光的位置
```

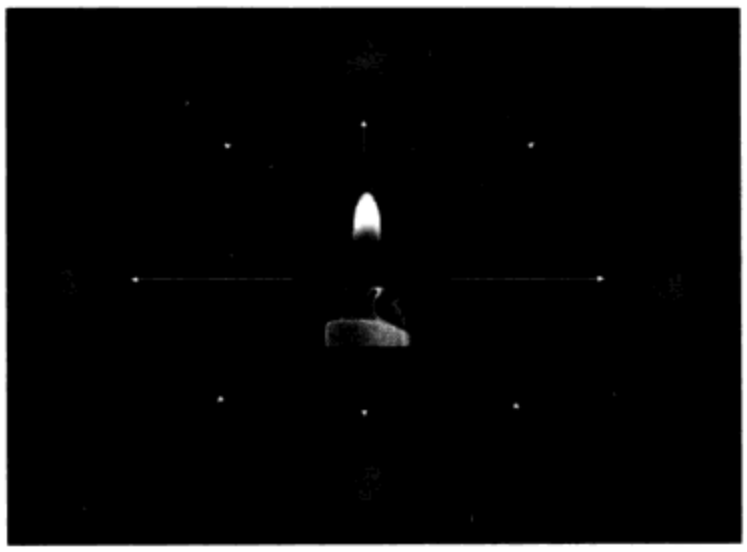
- positionParams 数组中前 3 个值组成表示定位光位置点的 x、y、z 坐标，光的方向从此点向四面八方照射。最后的 1 表示此光源为定位光。

- glLightfv 方法中各个参数含义为：GL10.GL_LIGHT0 是灯的 ID，GL10.GL_POSITION 表示是设置光源的位置或光线的方向，positionParams 给出光源的具体信息，0 给出的是第一个值在数组中的偏移量。

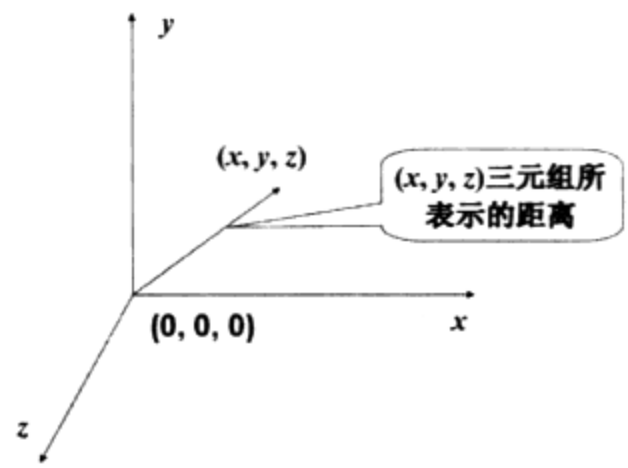
11.3.3　法向量

将光源的一系列参数设置完毕后，要想让光照起作用还需要为每个顶点指定法向量。每个顶点的法向量决定了光照时光的反射情况，如果没有为顶点设置法向量，光照系统也是不能正常工作的。

某个顶点的法向量是指此顶点处于三维物体表面处的法线方向，其用一个向量来表示。OpenGL ES 中向量用一个三元组来表示，格式为(x, y, z)。其表示的是从原点 O(0, 0, 0)出发到(x, y, z)点的一个向量，代表一个方向，如图 11-25 所示。



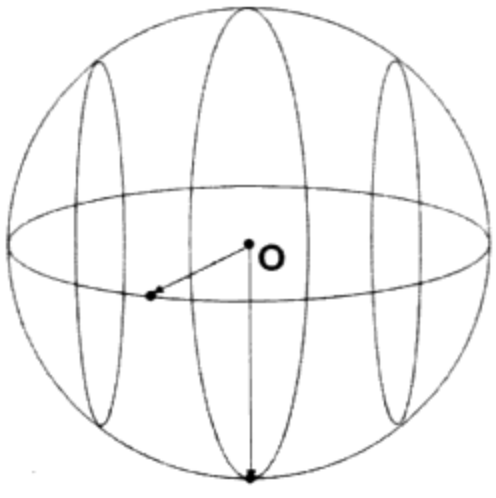
▲图 11-24　定位光示意图



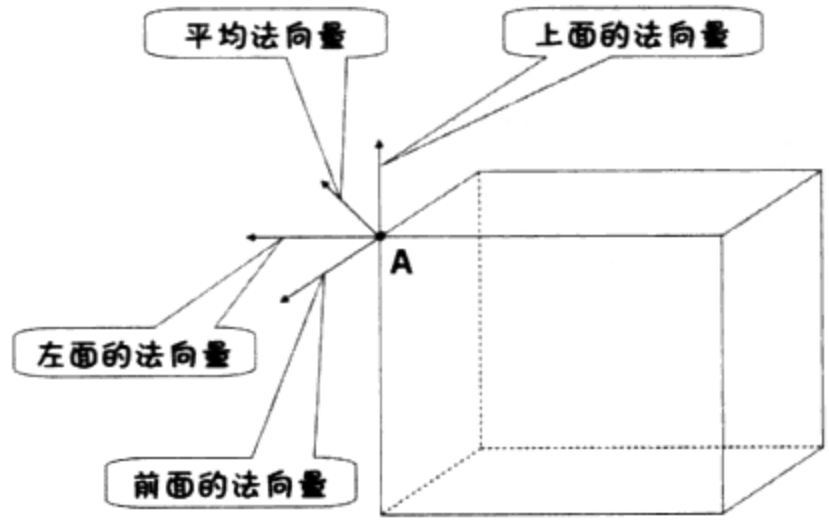
▲图 11-25　法向量示意图

提示　某个顶点的法向量三元组与其坐标三元组并没有必然的逻辑联系。开发中注意不要混淆法向量三元组与坐标三元组。

球面上某点的法向量就是球心与此点连线向量，若球心位于原点，则球面上点的法向量三元组与此点的三维坐标三元组是相同的，如图 11-26 所示。



▲图 11-26　球面的法向量示意图



▲图 11-27　长方体的法向量示意图

上述例子使用的是球面，球面处处连续可微，因此每个顶点都是可以方便地计算出法向量的。但很多三维物体的表面虽然是处处连续的，但某些特定的顶点处不可微，这时就不能直接计算出法向量了。

如图 11-27 所示，长方体的 A 顶点同时位于 3 个面（上面、左面、前面），因此，此顶点并不能直接计算出法向量。实际开发中有两种处理方式。

- 将 A 点处的 3 个法向量求平均值。
- 将 A 点处放置 3 个位于同一位置的顶点，每个顶点认为属于同一个面，单独给出各自的法向量。

另外要注意的是，给出顶点法向量时尽量使用规格化向量，也就是长度为单位长度的向量。如果顶点法向量给的不是规格化向量，系统在运算时还要执行规格化的操作，会在一定程度上对性能产生影响。

例如，向量(3, 3, 3)与(1, 1, 1)及(0.57735, 0.57735, 0.57735)表示的方向是一样的，但只有第 3 个向量的长度为 1，是单位向量。将向量规格化的算法非常简单，若对向量(x, y, z)进行规格化，假定规格的向量为(x', y', z')，规格化公式为：

$$\text{length}=\sqrt{x^2+y^2+z^2} \qquad x'=x/\text{length} \qquad y'=y/\text{length} \qquad z'=z/\text{length}$$

提示 编程开发时要注意，给出规格化法向量时要使用 float 类型，否则有可能会工作不正常。

有时不但需要物体能反射光，还需要物体能够发光，这时可以采用自发光效果，其核心代码如下。

```
1        float emission[] = {0.0f, 1f, 0f, 1.0f};                    //绿色自发光
2        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_EMISSION, emission,0);
```

提示 设置为自发光不会照亮其他物体。

11.3.4 材质

完成了光源设置与法向量的计算后，光照系统还是不能正常工作，最后一步就是为被光照的物体设置材质。设置材质的主要工作就是设置材质对于环境光、散射光、镜面反射光的反射能力，也就是说材质能反射什么颜色的光。例如，白色的材质能反射所有颜色的光，红色的材质只能反射红色光。

设置材质的主要代码如下：

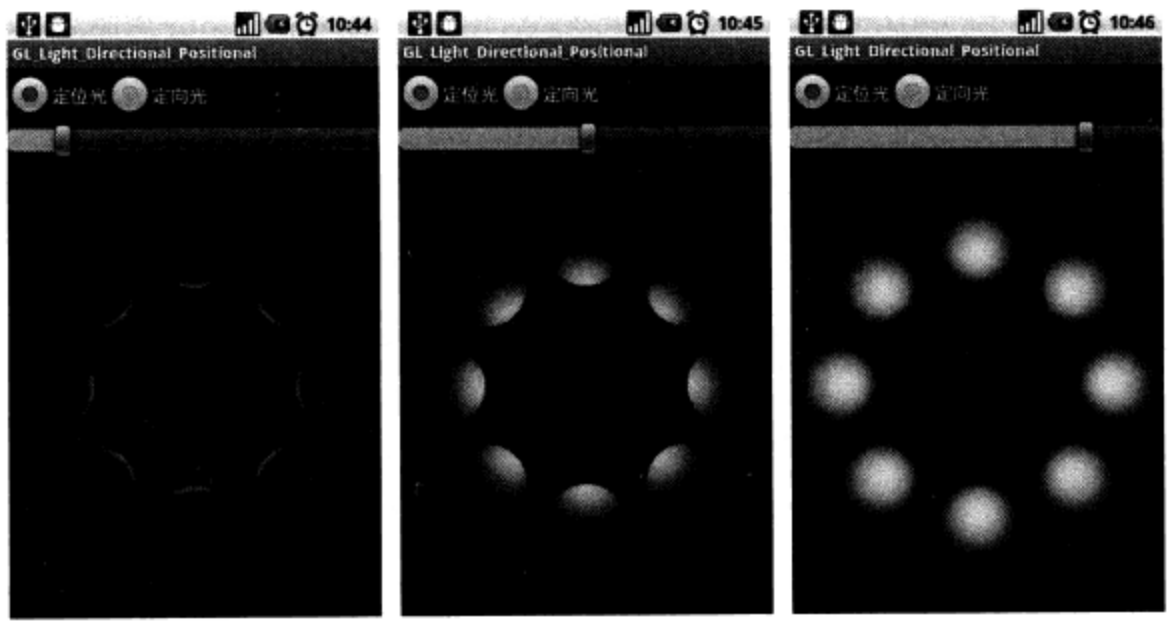
```
1        //环境光为白色材质
2        float ambientMaterial[] = {0.4f, 0.4f, 0.4f, 1.0f};
3        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, ambientMaterial,0);
4        //散射光为白色材质
5        float diffuseMaterial[] = {1.0f, 1.0f, 0.0f, 1.0f};
6        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, diffuseMaterial,0);
7        //高光材质为白色
8        float specularMaterial[] = {1.0f, 1.0f, 1.0f, 1.0f};
9        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR, specularMaterial,0);
```

```
10          //高光反射区域,数越大高亮区域越小、越暗
11          float shininessMaterial[] = {1.5f};
12          gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS, shininessMaterial,0);
```

提示　读者可能会觉得，为什么上述代码将材质都设置为了白色。这是因为白色的材质能很好地反射各种光源的光，为后面的例子打下基础。

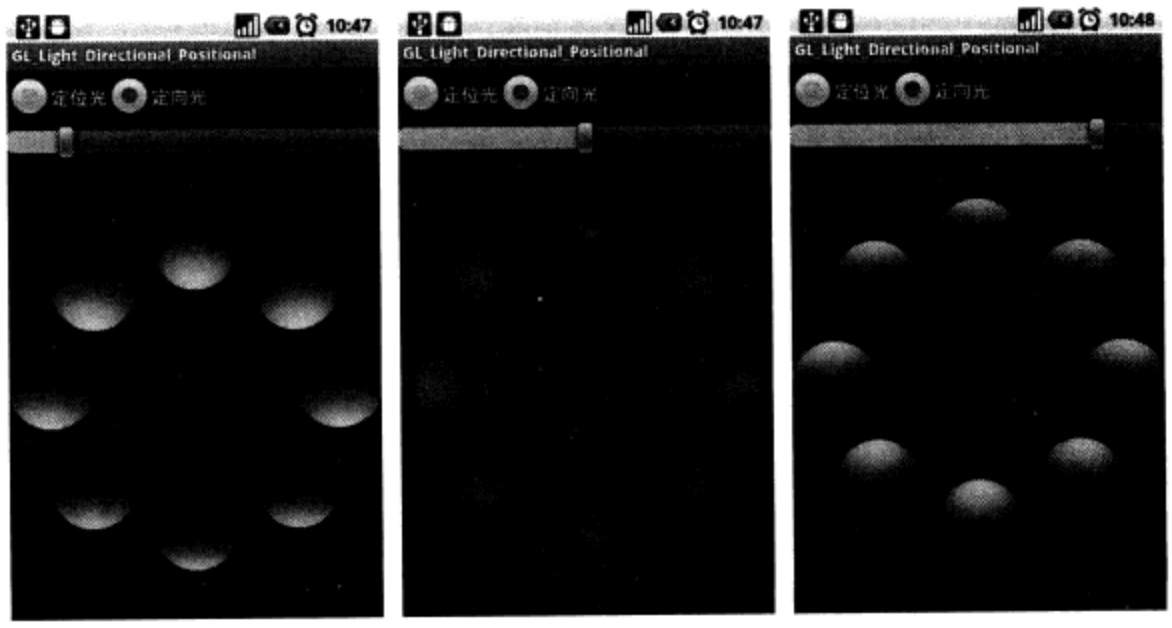
11.3.5　两种光源的案例

前面小节中详细介绍了定位光和定向光两种光源，本小节将给出一个使用了两种不同光源的案例 Sample_11_4，案例的运行效果如图 11-28、图 11-29 所示。



▲图 11-28　案例 Sample_11_4 的运行效果图 1

提示　图 11-28 中从左到右依次为当光源为定位光时，光源离视点距离比球远、光源离视点的距离与球相同，光源离视点距离比球近时的效果图。



▲图 11-29　案例 Sample_11_4 的运行效果图 2

提示 图 11-29 中从左到右依次为当光源为定向光时，光线方向从下往上、光线为(0, 0, 0)和光线方向从上往下时的效果图。

接下来介绍实现本案例的代码。在这里只讲解有关两种光源的核心类 MyGLView，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_4 /src/com/bn/game/chap11/ex4 目录下的 MyGLView.java。

```
1    package com.bn.game.chap11.ex4;                                //声名包
2    import javax.microedition.khronos.egl.EGLConfig;               //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.opengl.GLSurfaceView;                           //引入相关类
5    public class MyGLView extends GLSurfaceView{
6         private SceneRenderer renderer;                           //场景渲染器
7         private boolean isPosition=true;                          //是否为定位光的标志位
8         private float lightOffset=-4;                             //灯光的位置或方向的偏移量
9         public MyGLView(Context context) {
10             super(context);
11             renderer=new SceneRenderer();                        //创建渲染器
12             this.setRenderer(renderer);                          //设置渲染器
13             this.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
14        }
15        private class SceneRenderer implements GLSurfaceView.Renderer {
16             Ball ball=new Ball(1.0f,0.5f,80,80);
17             public SceneRenderer(){}
18             public void onDrawFrame(GL10 gl) {
19             gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
20             gl.glMatrixMode(GL10.GL_MODELVIEW);          //设置当前矩阵为模式矩阵
21             gl.glLoadIdentity();                         //设置当前矩阵为单位矩阵
22             gl.glTranslatef(0, 0f, -4f);
23             if(isPosition){                              //设定定位光的位置
24                  float[] positionParams={0,0,lightOffset,1};   //z 轴上的定位光
25                  gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, positionParams,0);
26             }
27             else{                                                //设定定向光的方向
28                  float[] directionParams={0,lightOffset,0,0};  //沿 y 轴某方向的定向光
29                  gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, directionParams,0);
30             }
31             float xyDis=1.7f;                                    //球与 z 轴的距离
32             float xyDis1=1.7f/1.414f;
33             gl.glPushMatrix();                                   //绘制右面的球
34             gl.glTranslatef(xyDis, 0f, 0);
35             ball.drawSelf(gl);
36             gl.glPopMatrix();
37             ……//该处省略了部分绘制球的代码，读者可自行查看随书光盘的源代码
38             gl.glPushMatrix();                                   //绘制左下角的球
39             gl.glTranslatef(-xyDis1, -xyDis1, 0);
40             ball.drawSelf(gl);
41             gl.glPopMatrix();
42             }
43             public void onSurfaceChanged(GL10 gl, int width, int height) {
44                  ……//该处省略了方法体，读者可自行查看随书光盘的源代码
45             }
46             public void onSurfaceCreated(GL10 gl, EGLConfig config) {
47             gl.glDisable(GL10.GL_DITHER);                   //关闭抗抖动
48             gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
49             gl.glClearColor(0,0,0,0);                       //设置屏幕背景色黑色 RGBA
50             gl.glShadeModel(GL10.GL_SMOOTH);                //设置着色模型为平滑着色
51             gl.glEnable(GL10.GL_DEPTH_TEST);                //启用深度测试
52             gl.glEnable(GL10.GL_CULL_FACE);                 //设置为打开背面剪裁
```

```
53              gl.glEnable(GL10.GL_LIGHTING);                    //允许光照
54              initGreenLight(gl);                               //初始化灯
55              initMaterial(gl);                                 //初始化材质
56              }
57          }//SceneRenderer
58          private void initGreenLight(GL10 gl){                 //初始化灯的方法
59            gl.glEnable(GL10.GL_LIGHT0);                        //打开 0 号灯
60            float[] ambientParams={0.1f,0.1f,0.1f,1.0f};        //光参数 RGBA
61            gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_AMBIENT, ambientParams,0); //环境光设置
62            float[] diffuseParams={1.0f,1.0f,1.0f,1.0f};//光参数 RGBA
63            gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE, diffuseParams,0); //散射光设置
64            float[] specularParams={1.0f,1.0f,1.0f,1.0f};//光参数 RGBA
65           gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_SPECULAR, specularParams,0);//反射光设置
66          }
67          private void initMaterial(GL10 gl){                       //初始化材质的方法
68            float ambientMaterial[] = {0.4f, 0.4f, 0.4f, 1.0f};     //环境光为白色材质
69           gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, ambientMaterial,0);
70            float diffuseMaterial[] = {1.0f, 1.0f, 0.0f, 1.0f};     //散射光为白色材质
71           gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, diffuseMaterial,0);
72            float specularMaterial[] = {1.0f, 1.0f, 1.0f, 1.0f};    //高光材质为白色
73           gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR, specularMaterial,0);
74            float shininessMaterial[] = {1.5f};                     //高光反射区域
75           gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS, shininessMaterial,0);
76          }
77          ……//该处省略了部分成员方法，读者可自行查看随书光盘的源代码
78      }
```

- 代码第 6-8 行为该类中成员变量的定义，其中 boolean 型的变量 isPosition 是控制两种光源切换的标志位，lightOffset 为两种光源的位置或方向的偏移量。
- 代码第 16 行创建了一个 Ball 类的成员变量，Ball 类表示一个球心在原点的球类，其也是由三角形卷绕而成的，只是相对于圆稍加复杂。注意 Ball 类中各点的法向量就是该点坐标本身。
- 代码第 46-56 行在 SceneRenderer 类中 onSurfaceCreated 方法中允许光照，并分别调用了 initGreenLight 方法和 initMaterial 方法来初始化灯和材质。
- 代码第 18-42 行加载了模式矩阵后，根据变量 isPosition 的值调用相应设置光源类型和参数的方法，故当拖拉条和单选按钮改变时主界面会发生相应的变化。

11.4 纹理及纹理映射

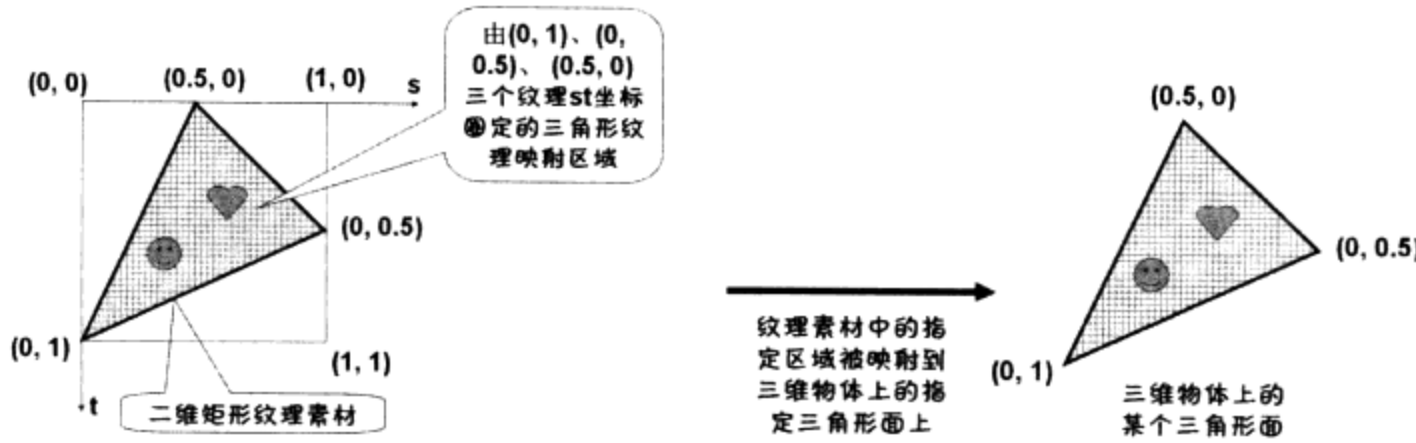
上一小节中已经介绍完了光照与材质，有了光照与材质后，画布效果丰富了很多，但是三维物体的 3 面只能是平滑的，还没有花纹等细节。要想看到更为真实的效果，还需要使用纹理映射。本小节将介绍有关纹理映射的相关知识。

11.4.1　纹理映射基本原理

纹理映射纹理素材映射到三维物体指定的多边形面上。要注意的是 OpenGL ES 中仅支持三角形，因此，指定纹理时也要注意这点。图 11-30 的示意图给出了纹理映射的基本原理。

提示

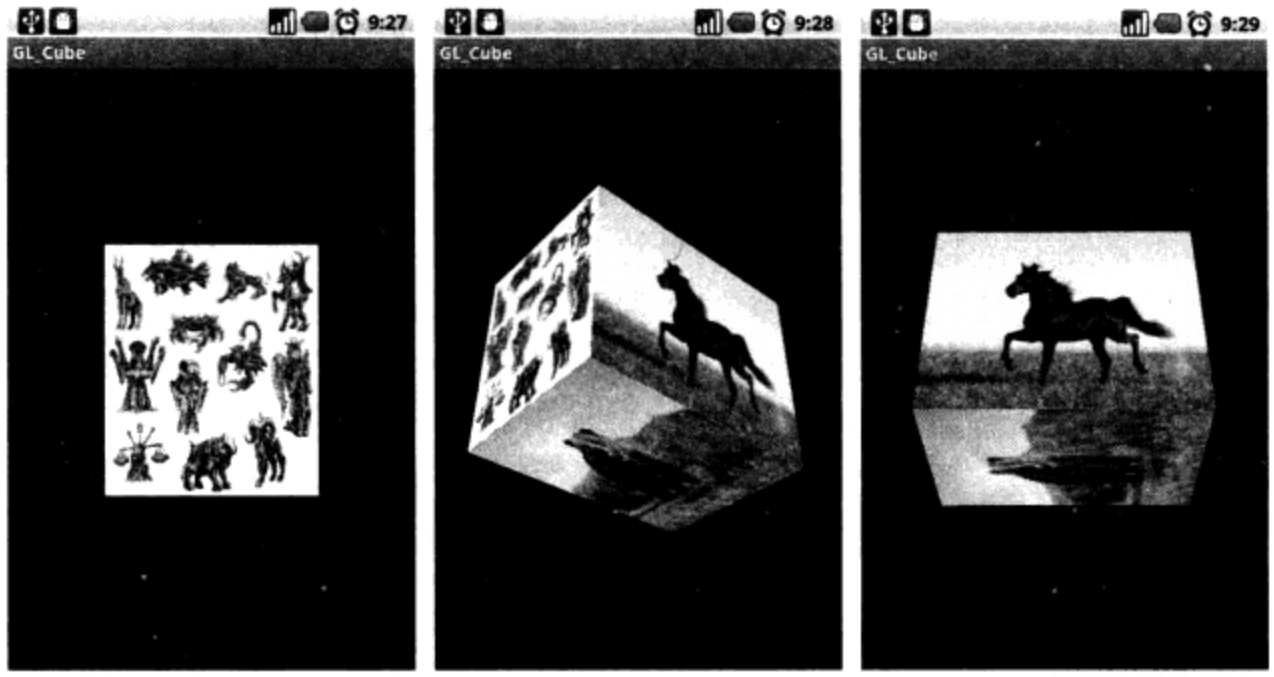
OpenGL ES 中对纹理映射时对纹理图片的尺寸是有规定的，要求纹理图片的宽度与高度必须为 2^n(2 的 n 次方)，即 32 × 32、256 × 512 等。



▲图 11-30　纹理映射的基本原理示意图

11.4.2　使用纹理映射的案例

前面小节中详细介绍了纹理映射，为了使读者对纹理映射的理解更加深刻，本小节将给出一个使用了纹理映射贴图的案例 Sample_11_5，案例的运行效果如图 11-31 所示。



▲图 11-31　案例 Sample_11_5 的运行效果图

提示　图 11-31 中从左到右为案例 Sample_11_5 运行时旋转不同角度的效果图。

接下来介绍实现本案例的代码。在这里讲解有关纹理贴图的核心类 TextureRect，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_5/src/com/bn/game/chap11/ex5 目录下的 TextureRect.java。

```
package com.bn.game.chap11.ex5;                                  //声名包
import java.nio.ByteBuffer;                                      //引入相关类
……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
import javax.microedition.khronos.opengles.GL10;                 //引入相关类
public class TextureRect {
     private FloatBuffer   vertexBuffer;                         //顶点坐标数据缓冲
    private FloatBuffer   textureBuffer;                         //顶点纹理数据缓冲
```

```
8        int vCount=0;                                                    //顶点个数
9        float size;                                                      //尺寸
10       float xAngle=0;                                                  //绕 x 轴旋转的角度
11       float yAngle=0;                                                  //绕 y 轴旋转的角度
12       float zAngle=0;                                                  //绕 z 轴旋转的角度
13       int textureId;                                                   //纹理 id
14         public TextureRect(float scale,float a, float b, int textureId) {
15              this.textureId=textureId;
16              size=Constant.UNIT_SIZE*scale;
17              a*=size;
18              b*=size;
19              float xOffset=a/2;                                  //离开原点的距离
20              float yOffset=b/2;
21              vCount=6;
22              //坐标数据初始化
23              float[] vertices=new float[]{
24                   -xOffset,-yOffset,0,
25                   xOffset,yOffset,0,
26                   -xOffset,yOffset,0,
27
28              -xOffset,-yOffset,0,
29              xOffset,-yOffset,0,
30              xOffset,yOffset,0
31              };
32              ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                                    //创建顶点坐标数据缓冲
33           vbb.order(ByteOrder.nativeOrder());                    //设置字节顺序
34           vertexBuffer = vbb.asFloatBuffer();                    //转换为 float 型缓冲
35          vertexBuffer.put(vertices);                             //向缓冲区中放入顶点坐标数据
36           vertexBuffer.position(0);                              //设置缓冲区起始位置
37           //纹理数据的初始化
38           float[] textures=new float[]{                          //顶点纹理 S、T 坐标值数组
39                   0,1,
40                   1,0,
41                   0,0,
42                   0,1,
43                   1,1,
44                   1,0,
45           };
46           ByteBuffer cbb = ByteBuffer.allocateDirect(textures.length*4);
                                                                    //创建顶点纹理数据缓冲
47           cbb.order(ByteOrder.nativeOrder());                    //设置字节顺序
48           textureBuffer = cbb.asFloatBuffer();                   //转换为 float 型缓冲
49          textureBuffer.put(textures);                            //向缓冲区中放入顶点纹理数据
50           textureBuffer.position(0);                             //设置缓冲区起始位置
51         }
52       public void drawSelf(GL10 gl){
53         gl.glPushMatrix();
54           gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);          //启用顶点坐标数组
55           gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);   //启用顶点纹理数组
56           //绕轴旋转
57           gl.glRotatef(xAngle, 1, 0, 0);
58           gl.glRotatef(yAngle, 0, 1, 0);
59           gl.glRotatef(zAngle, 0, 0, 1);
60                                                                  //为画笔指定顶点坐标数据
61           gl.glVertexPointer(
62                   3,                                             //每个顶点的坐标数量为 3  xyz
63                   GL10.GL_FLOAT,                                 //顶点坐标值的类型
64                   0,                                             //连续顶点坐标数据之间的间隔
65                   vertexBuffer                                   //顶点坐标数据
66           );
67           //为画笔指定纹理 ST 坐标缓冲
```

```
68          gl.glEnable(GL10.GL_TEXTURE_2D);                    //开启纹理
69          gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, textureBuffer);
                                                                //为画笔指定纹理 ST 坐标缓冲
70          gl.glBindTexture(GL10.GL_TEXTURE_2D, textureId);        //绑定当前纹理
71          //绘制图形
72          gl.glDrawArrays(
73                  GL10.GL_TRIANGLES,                              //以三角形方式填充
74                  0,                                              //开始点编号
75                  vCount                                          //顶点的数量
76          );
77          gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);          //禁用顶点坐标数组
78          gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);   //禁用顶点纹理数组
79          gl.glPopMatrix();
80    }}
```

- 代码第 6-13 行为该类中成员变量的定义，其中第 7 行声明了顶点纹理缓冲，第 13 行中 textureId 是纹理 id，该纹理 id 是用系统的方法自动生成的，代表一个图片资源的 id。
- 代码第 14-51 行为构造器的开发，在构造器中初始化某些成员变量之后，又初始化了纹理矩形的顶点坐标和纹理坐标。注意顶点坐标和纹理坐标的相对应。
- 代码第 52-80 行为绘制纹理矩形的方法，绘制前开户纹理数组、为画笔指定纹理 ST 坐标缓冲并绑定当前纹理，便实现了纹理贴图。

前面讲到了纹理 id 的生成是调用系统的方法实现的，下面将实现生成纹理 id 的方法做简单介绍，该方法给定图片资源的 id 会返回一个对应的纹理 id。其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_5/src/com/bn/game/chap11/ex5 目录下的 MyGLView.java。

```
1         //初始化纹理的方法
2         public int initTexture(GL10 gl,int drawableId){
3                                                               //生成纹理 ID
4             int[] textures = new int[1];
5             gl.glGenTextures(1, textures, 0);//生成一个纹理 id 放在 textures 数组中的 0 位置
6             int currTextureId=textures[0];                    //获取生成的纹理 id
7             gl.glBindTexture(GL10.GL_TEXTURE_2D, currTextureId);    //绑定该纹理 id
8             //设置 MIN_FILTER 与 MAG_FILTER 为 MIPMAP 纹理过滤方式
9             gl.glTexParameterf(GL10.GL_TEXTURE_2D,
10                    GL10.GL_TEXTURE_MIN_FILTER,GL10.GL_LINEAR_MIPMAP_NEAREST);
11            gl.glTexParameterf(GL10.GL_TEXTURE_2D,
12                    GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR_MIPMAP_LINEAR);
13                                                              //生成 MIPMAP 纹理
14            ((GL11)gl).glTexParameterf(GL10.GL_TEXTURE_2D,
15                    GL11.GL_GENERATE_MIPMAP, GL10.GL_TRUE);
16                                                              //设置纹理拉伸方式为 REPEAT
17        gl.glTexParameterf(GL10.GL_TEXTURE_2D,
18                GL10.GL_TEXTURE_WRAP_S,GL10.GL_CLAMP_TO_EDGE);
19        gl.glTexParameterf(GL10.GL_TEXTURE_2D,
20                GL10.GL_TEXTURE_WRAP_T,GL10.GL_CLAMP_TO_EDGE);
21        InputStream is = this.getResources().openRawResource(drawableId);
                                                                //获取图片资源的输入流
22        Bitmap bitmapTmp;
23        try{
24            bitmapTmp = BitmapFactory.decodeStream(is);//通过输入流生成位图
25        }
26        finally{
27            try {
28                is.close();                                   //关闭流
```

```
29              }catch(IOException e){
30                  e.printStackTrace();                    //打印异常
31              }
32          }
33          GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmapTmp, 0);
                                                            //自动设置图片的格式和类型
34          bitmapTmp.recycle();                            //回收图片资源
35          return currTextureId;
36      }
```

- 代码第 4-7 行由系统的方法 glGenTextures 自动生成一个纹理 id，放在 textures 数组中的 0 位置，再调用 glBindTexture 方法绑定该纹理 id 后，后面的操作都是针对该 id 的。
- 代码第 8-20 设置了纹理 id 的过滤方式和纹理拉伸方式，后面章节还要进行详细介绍，在这里读者只需大概浏览一下注释，了解大意即可。
- 代码第 21-34 行中为绑定的纹理设置了图片资源，通过输入流生成一个新的位图资源，最后再将资源回收，否则会常贮内存，造成空间的浪费。

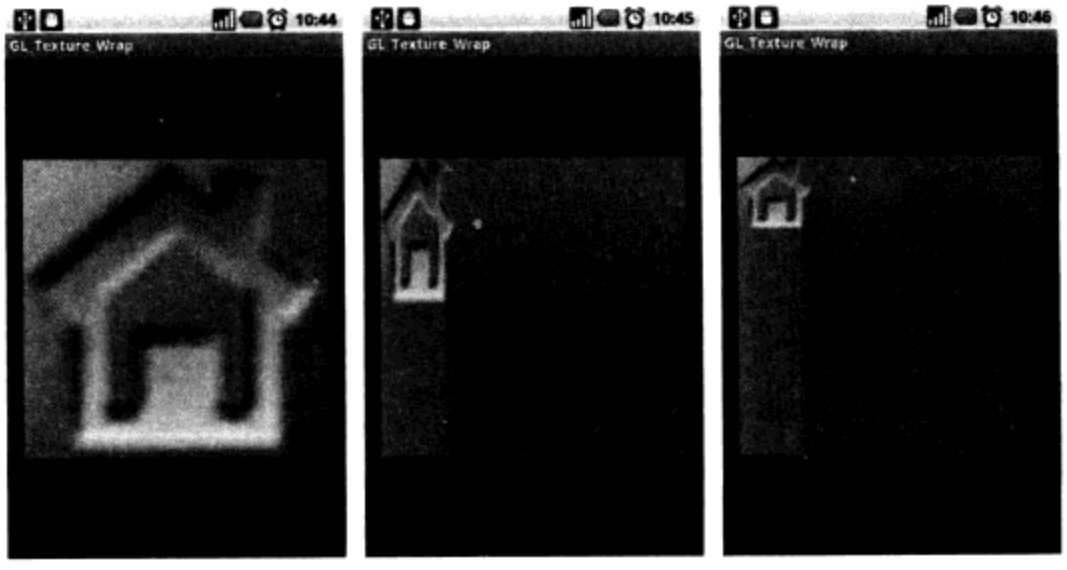
11.4.3　几种纹理拉伸方式

前面介绍的案例中都是把一幅纹理图拉伸或缩小贴到目标面上，有时目标面很大，纹理被拉大后就显得非常不清楚。此时希望纹理能够在目标上自动重复，这其实很容易实现，只需要做两方向的工作。

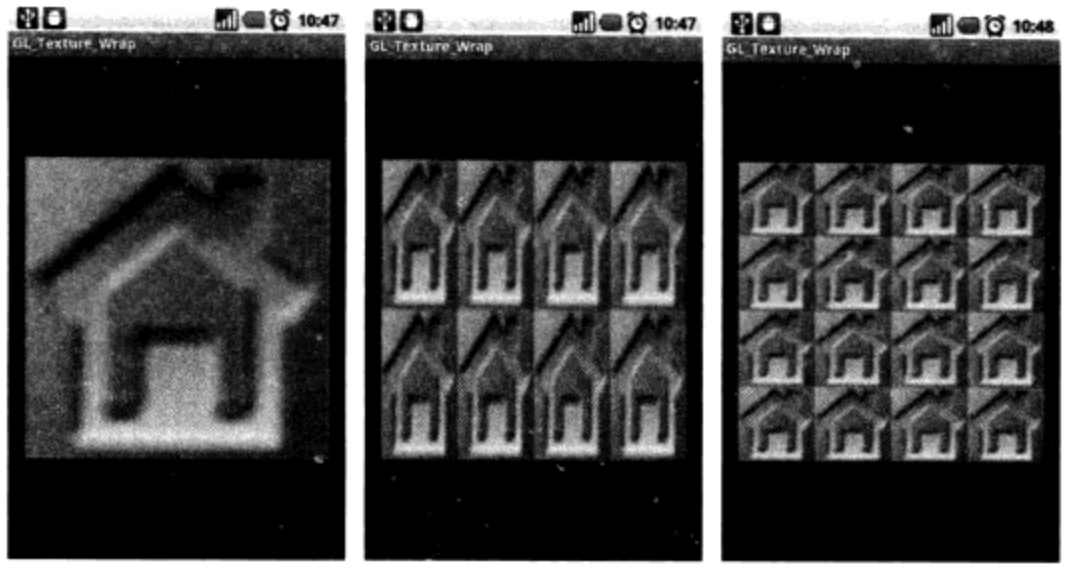
- 将纹理的 GL10.GL_TEXTURE_WRAP_S 与 GL10.GL_TEXTURE_WRAP_T 属性设置为 GL10.GL_REPEAT 而不是 GL10.GL_CLAMP_TO_EDGE。
- 设置纹理坐标时取值范围不再是 0～1，而是 0～n，其中 n 为期望纹理重复的次数。

下面给出一个使用了两种纹理拉伸的案例 Sample_11_6。由于篇幅所限，这里只给出案例的运行效果图，读者可自行查看随书光盘的源代码。案例的运行效果如图 11-32、图 11-33 所示。

提示　图 11-32 中为当 GL10.GL_TEXTURE_WRAP_S 与 GL10.GL_TEXTURE_WRAP_T 属性设置为 GL10.GL_CLAMP_TO_EDGE 时，纹理坐标为 1×1、4×2、4×4 时的效果图。



▲图 11-32　案例 Sample_11_6 的运行效果图 1



▲图 11-33　案例 Sample_11_6 的运行效果图 2

提示

图 11-33 中为当 GL10.GL_TEXTURE_WRAP_S 与 GL10.GL_TEXTURE_WRAP_T 属性设置为 GL10.GL_REPEAT 时，纹理坐标为 1×1、4×2、4×4 时的效果图。

11.4.4　几种纹理过滤方式

前面的案例中将纹理图贴到三维物体表面的过程就是采用的纹理过滤，也就是说纹理映射的过程就是不断执行纹理过滤的过程。

可以想象出，进行纹理过滤时，正在使用的纹理通常也正在被进行放大或缩小。换句话说，这个纹理将被映射到一个比它大或小的图元上的图像上。纹理的放大会导致一个纹理像素映射到许多实际像素上，图形呈现出来就会发虚。纹理的缩小会导致许多纹理像素被映射到一个实际像素上，也会产生不好的视觉效果。

因此，开发时根据实际的情况选择合适的纹理过滤方式就非常重要了。选择合适可以取得事半功倍的效果，选择不合适就很难满足用户的需要了。实际开发中，有两方面的纹理过滤需要设置。

- GL_TEXTURE_MIN_FILTER。其在图像绘制时小于贴图的原始尺寸时采用。
- GL_TEXTURE_MAG_FILTER。其在图像绘制时大于贴图的原始尺寸时采用。

OpenGL ES 中常用的纹理过滤方式有：最近点采样、线性纹理过滤和 MIPMAP 纹理过滤。下面对 3 种纹理过滤方式一一进行介绍。

（1）最近点采样是最简单的纹理过滤方式，前面案例中的 MIN_FILTER 采用的就是最近点采样，其代码如下。

```
1    gl.glTexParameterf(                                    //最近点采样
2         GL10.GL_TEXTURE_2D,
3         GL10.GL_TEXTURE_MIN_FILTER,
4         GL10.GL_NEAREST);
```

提示

所谓最近点采样就是，贴图时根据映射目标图像中像素的纹理坐标折算取整从纹理图中提取像素。当纹理图的大小与映射目标图像的大小差不多时，这种方法非常有效和快捷。如果大小不同，纹理就需要进行放大或缩小，这样，结果就会变得矮胖、变形或模糊。

（2）线性纹理过滤比最近点采样稍微复杂一点。前面例子中的 MAG_FILTER 采用的就是线性纹理过滤，其基本代码如下。

```
1    gl.glTexParameterf(                                    //线性纹理过滤
2         GL10.GL_TEXTURE_2D,
3         GL10.GL_TEXTURE_MAG_FILTER,
4         GL10.GL_LINEAR);
```

提示　线性纹理过滤也是要计算目标图像中像素的纹理坐标再取整，从纹理图中提取像素，但采用的计算方法比最近点采样复杂，不过可以取得更好的效果。

（3）MIPMAP 纹理过滤技术比前面两种复杂，其可以用来降低场景渲染的时间消耗。同时也提高了场景的真实感。但它的缺点是要占用大量的内存空间，在内存受限的场合不适合。

一个 MipMap 就是一系列的纹理图，每一幅纹理图都与前一幅是相同的图样，但是分辨率都要比前一幅有所降低。MipMap 中的每一幅或者每一级图像的高和宽都比前一级小二分之一。要注意的是，MipMap 并不一定是正方形。

高分辨率时的 MipMap 图像用于接近观察者的物体。当物体逐渐远离观察者时，使用低分辨率的图像 MipMap 可以提高场景渲染的质量，但是它的内存消耗却很大。这是因为此方法能够模拟纹理的透视效果并能减少处理时的计算量，与将一幅纹理用于不同的分辨率相比，这种方法更加快速。

要采用 MipMap 纹理过滤，基本代码如下。

```
1    gl.glTexParameterf(                                    //MipMap 纹理过滤
2         GL10.GL_TEXTURE_2D,
3         GL10.GL_TEXTURE_MIN_FILTER,
4         GL10.GL_LINEAR_MIPMAP_NEAREST);
5    gl.glTexParameterf(                                    //MipMap 纹理过滤
6         GL10.GL_TEXTURE_2D,
7         GL10.GL_TEXTURE_MAG_FILTER,
8         GL10.GL_LINEAR_MIPMAP_LINEAR);
```

提示　从代码中可以看出 MipMap 也分 NEAREST（最近点采样）与 LINEAR（线性纹理过滤）。

除了前面设置纹理过滤方式的代码，还需要生成 MipMap 系列纹理，其代码如下。

```
1    ((GL11)gl).glTexParameterf(                            //生成 MipMap 系列纹理
2         GL10.GL_TEXTURE_2D,
3         GL11.GL_GENNERATE_MIPMAP,
4         GL10.GL_TRUE);
```

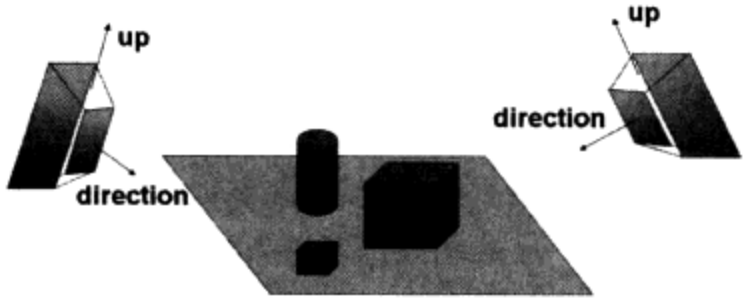
提示　由于篇幅所限，关于 MipMap 纹理技术的使用，这里不再举例。

11.5 摄像机和雾特效

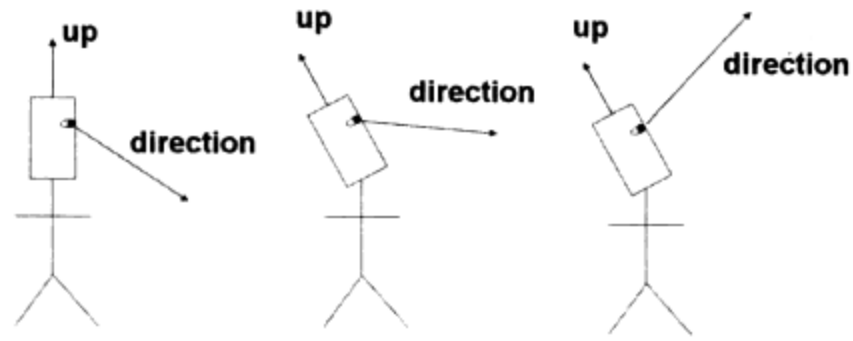
上一节中已经介绍了纹理及纹理映射的相关知识，本节将介绍最后两个知识点：摄像机和雾气。通过本节的学习读者将掌握摄像机的设置和雾气的开发，之后就可以开发更加丰富多样的 3D 场景了。

11.5.1 摄像机的设置

默认情况下，摄像机是位于原点，观察的方向是沿 z 轴的负方向。但在很多实际应用中都需要根据程序运行的情况修改摄像机的位置、朝向及 UP 方向。像机的和人观察物体的示意图如图 11-34、图图 11-35 所示。



▲图 11-34　摄像机观察物体示意图



▲图 11-35　人观察物体的示意图

- 摄像机的位置很好理解，指的是摄像机的 X，Y，Z 轴坐标，也就是观察者眼睛的位置。
- 摄像机的朝向可以理解为观察者目光的方向。
- 摄像机的 up 方向可以理解为观察者头顶法线的指向。
- 摄像机的位置、朝向、up 方向可以有很多种组合，例如，同样的位置可以有不同的朝向、up 方向，这与现实世界中人观察世界的情况是非常类似的。

设置观察点使用 GLU 类的 gluLookAt 方法，其代码如下。

```
1     GLU.gluLookAt(                                          //设置 camera（摄像机）位置
2                   arg0,
3                   arg1,                                     //人眼位置的 X
4                   arg2,                                     //人眼位置的 Y
5                   arg3,                                     //人眼位置的 Z
6                   arg4,                                     //人眼看的点 X
7                   arg5,                                     //人眼看的点 Y
8                   arg6,                                     //人眼看的点 Z
9                   arg7,                                     //up 朝向 x
10                  arg8,                                     //up 朝向 y
11                  arg9                                      //up 朝向 z
12            );
```

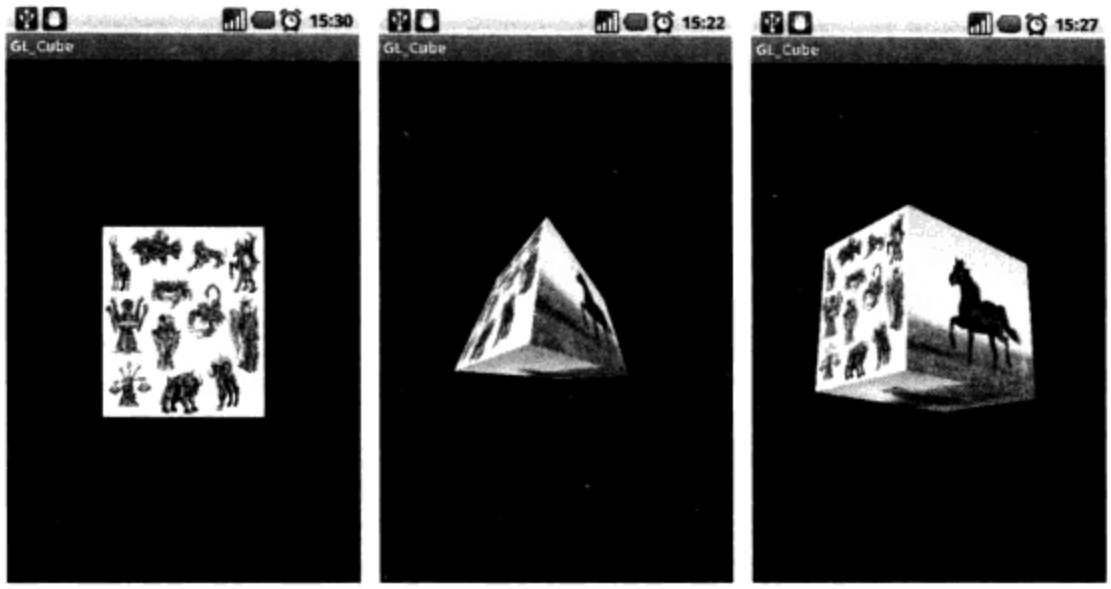
提示　方法中的各参数含意为，arg0 表示画笔，arg1 –arg3 依次表示观察点位置的 X、Y、Z 坐标，arg4 –arg6 依次表示目光目标点的 X、Y、Z 坐标，arg7 –arg9 依次表示 up 方向向量的 X、Y、Z 分量。

11.5.2 设置合理的视角

前面章节在介绍两种投影模式时介绍过视角的问题。实际开发中，若使用的是正交投影，由于其产生近大远小的效果，因此视角的大小只是会影响在同样的距离上看到空间的多少，不会引起物体的变形。

如果采用的是透视投影，则不但视角大小会影响看到空间的多少，或视角设置不合理还可能会

引起物体的变形。为了使读者对视角的理解更加深刻，本小节将给出一个使用了设置合理视角的案例 Sample_11_7，案例的运行效果如图 11-36 所示。



▲图 11-36　案例 Sample_11_7 的运行效果图

说明　图 11-36 中最左边的图为合理视角和不合理视角开始进行时的效果图，中间的图为不合理视角转过一定角度后的效果图，最右面的图为合理视角转过相同角度后的效果图。

从图 11-36 中可以明显看出，视角如果不合理，会引起物体的变形。本案例中物体变形是非常严重的。有关物体视角的代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_7 /src/com/bn/game/chap11/ex7 目录下的 MyGLView.java。

```
1    public void onDrawFrame(GL10 gl) {
2    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
3    gl.glMatrixMode(GL10.GL_MODELVIEW);          //设置当前矩阵为模式矩阵
4    gl.glLoadIdentity();                         //设置当前矩阵为单位矩阵
5    gl.glTranslatef(0, 0f, -1.9f);               //不合理视角
6       // gl.glTranslatef(0, 0f, -6f);            //合理视角
7    cube.drawSelf(gl);                           //绘制立方体
8    }
9    @Override
10   public void onSurfaceChanged(GL10 gl, int width, int height) {
11
12   gl.glMatrixMode(GL10.GL_PROJECTION);         //设置当前矩阵为投影矩阵
13   gl.glLoadIdentity();                         //设置当前矩阵为单位矩阵
14   float ratio = (float) width / height;        //计算透视投影的比例
15   gl.glFrustumf(-ratio, ratio, -1, 1, 0.4f, 10);//调用此方法计算产生透视投影
                                                   //矩阵，不合理的视角
16      // gl.glFrustumf(-ratio, ratio, -1, 1, 2.0f, 10);//调用此方法计算产生透
                                                   //视投影矩阵，合理的视角
17   }
```

- 代码第 5 行和第 15 行都是不合理视角的代码，可以看出摄像机离物体很近，但视角开阔，也可以看全整个立方体。

- 代码第 6 行和第 16 行都被注释掉了，两句都是合理视角时的代码。此时摄像机离物体的位置较远，而视角较小。
- 两种视角相比较，不合理的视角使物体变形，合理的视角不会使物体变形。而两种视角都能看清整个立方体，并且看到的物体大小都一样，原理如图 11-37 所示。

▲图 11-37　视角原理图

> **提示**　图 11-37 表明，同一个程序修改了视角（一般为改小视角），可能会觉得看到的物体变大了，此时只要将摄像机远离物体即可。

11.5.3　雾特效的开发

计算机图像很多情况下轮廓过于鲜明，显得不够真实。在进行 3D 应用开发时可以加入雾效果，将物体融入到背景中，使整个图像更为自然。

雾是一个通用术语，描述了一种类似大气效果的各种形态，其可以用于模拟薄雾和污染。雾效果启用后，离视点较远的物体将融入到雾的颜色中。可以控制雾的浓度，其决定了物体的淡化速度和雾的颜色。

> **提示**　大多数仿真程序中，使用雾效果可以提高性能。因为启用雾效果后无需绘制由于雾效果的影响而看不见的物体。

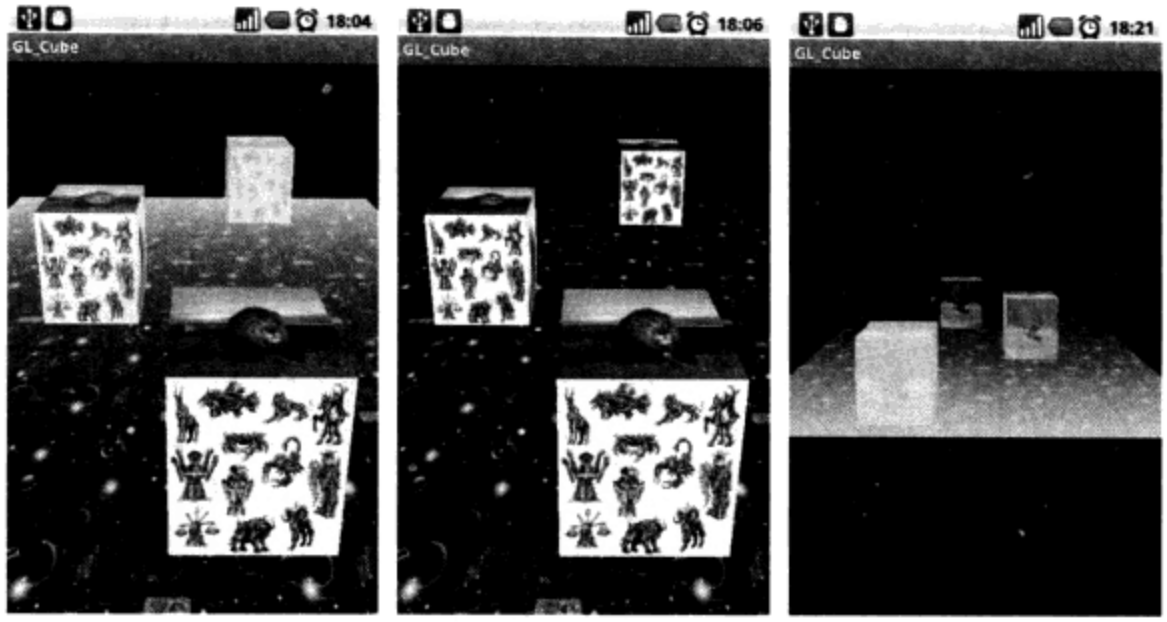
在 OpenGL ES 中使用雾效果非常简单，其核心代码如下。

```
1        gl.glEnable(GL10.GL_FOG);                          //允许雾
2        float[] fogColor={1,0.91765f,0.66667f,0};          //雾的颜色
3        gl.glFogfv(GL10.GL_FOG_COLOR, fogColor, 0);        //设置雾的颜色
4        gl.glFogx(GL10.GL_FOG_MODE, GL10.GL_EXP2);         //设置雾的模式
5        gl.glFogf(GL10.GL_FOG_DENSITY, 0.13f);             //设置雾的浓度
6        gl.glFogf(GL10.GL_FOG_START, 0.5f);                //设置雾的开始距离
7        gl.glFogf(GL10.GL_FOG_END, 100.0f);                //设置雾的结束距离
```

下面给出一个使用雾特效场景的案例 Sample_11_8。案例的运行效果如图 11-38 所示。

> **说明**　图 11-38 中从左到右分别为启用雾特效的场景、未启用雾特效的场景、启用雾特效后视角转到后面时的场景图。

接下来介绍实现本案例的代码。在这里只讲解有关实现雾气特效的核心类中部分源代码，其代码如下。



▲图 11-38　案例 Sample_11_8 的运行效果图

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_8 /src/com/bn/game/chap11/ex8 目录下的 MyGLView.java。

```
1          public void onSurfaceCreated(GL10 gl, EGLConfig config) {
2               gl.glDisable(GL10.GL_DITHER);                          //关闭抗抖动
3               gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
4               gl.glClearColor(0,0,0,0);                              //设置屏幕背景色为黑色 RGBA
5                gl.glEnable(GL10.GL_CULL_FACE);                       //设置为打开背面剪裁
6                gl.glEnable(GL10.GL_DEPTH_TEST);                      //启用深度测试
7                //初始化纹理
8               textureIds=new int[]{
9                    ……//该处省略了初始化纹理的代码，读者可自行查看随书光盘的源代码
10               };
11              desertTexId=initTexture(gl,R.drawable.desert);      //创建底下的纹理矩形
12              cube=new Cube(1.4f,new float[]{1,1.2f,1.4f},textureIds);//创建立方体
13              desertRect=new TextureRect(6,1.5f,2.4f,desertTexId,4,5);
                                                                    //创建底下的纹理矩形
14               gl.glEnable(GL10.GL_FOG);//允许雾
15              initFog(gl);//初始化雾
16         }
```

- 本段代码为 MyGLView 类中的内部类 SceneRenderer 中 onSurfaceCreated 方法。
- 代码第 14-15 行为开启雾效果的核心代码，在界面被创建时开启的雾效果，其中 initFog 方法就是把设置雾效果的代码封装了起来。

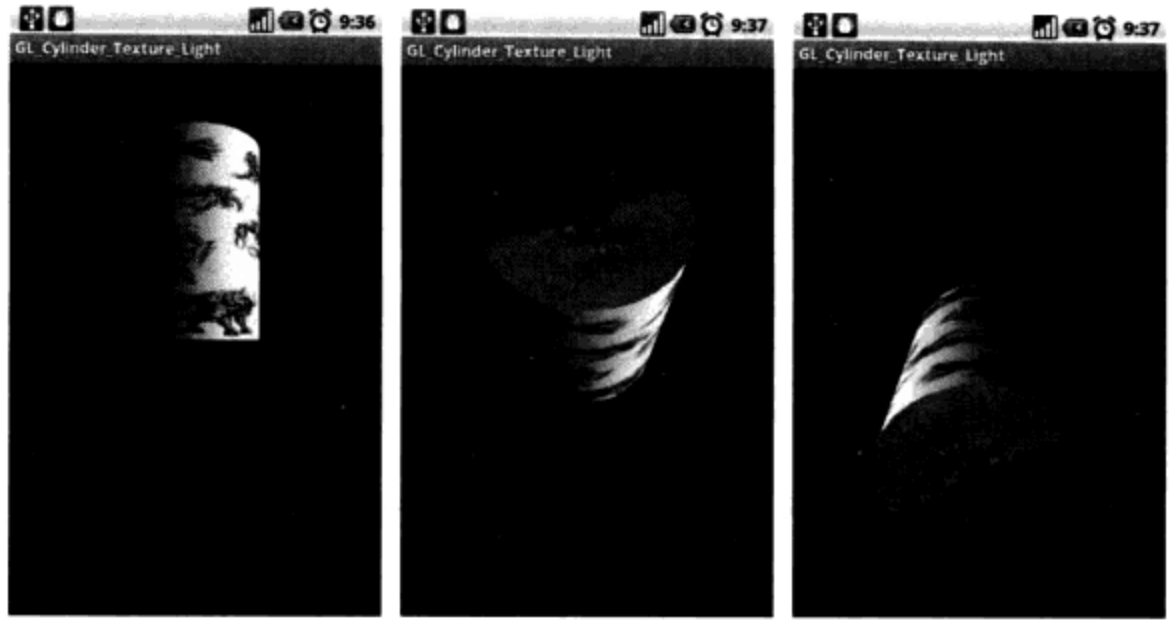
提示　关于场景绘制的代码，由于篇幅所限，这里不再介绍，读者可自行查看随书光盘中的源代码。

11.6 典型几何体的开发

前面几节中详细介绍了 OpenGL ES 应用开发的基本知识，本节将介绍几种典型基本几何体的开发，有了基本几何体，读者就可以开发简单的 3D 应用了。

11.6.1 圆柱

圆柱的开发比较简单，只有一个圆柱面和两个圆面组成，纹理坐标和法向量的计算也相对容易。下面介绍本小节的案例 Sample_11_9，其运行效果如图 11-39 所示。



▲图 11-39　案例 Sample_11_9 的运行效果图

接下来介绍实现本案例的代码。在这里讲解有关几何体组成的几个核心类 Circle、CylinderSide 及 Cylinder，下面对 3 个类一一进行讲解。

（1）Circle 类是圆类，是圆柱的两个圆面，有顶点、纹理和法向量，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_9/src/com/bn/game/chap11/ex9 目录下的 Circle.java。

```
1     ……//该处省略了声明包和相关类的引入代码，读者可自行查看随书光盘的源代码
2     public class Circle {
3          private FloatBuffer   vertexBuffer;                          //顶点坐标数据缓冲
4          private FloatBuffer   normalBuffer;                          //顶点法向量数据缓冲
5          private FloatBuffer   textureBuffer;                         //顶点纹理数据缓冲
6        int vCount=0;                                                  //顶点个数
7        float size;                                                    //尺寸
8        float angdegSpan;                                              //每个三角形顶角
9        float xAngle=0;                                                //绕 z 轴旋转的角度
10       float yAngle=0;                                                //绕 y 轴旋转的角度
11       float zAngle=0;                                                //绕 z 轴旋转的角度
12       int textureId;                                                 //纹理 id
13         public Circle(float scale,float r, int n, int textureId) {//大小、半径、边数
14              this.textureId=textureId;
15              size=Constant.UNIT_SIZE*scale;                   //改变尺寸
16              r*=size;
17              angdegSpan=360.0f/n;
18              vCount=3*n;                  //顶点个数，共有 n 个三角形，每个三角形都有 3 个顶点
19              float[] vertices=new float[vCount*3];            //坐标数据
20              float[] textures=new float[vCount*2];            //顶点纹理 S、T 坐标值数组
21              //坐标数据初始化
22              int count=0;
23              int stCount=0;
24              for(float angdeg=0;Math.ceil(angdeg)<360;angdeg+=angdegSpan){
25                   double angrad=Math.toRadians(angdeg);                //当前弧度
```

```
26                  double angradNext=Math.toRadians(angdeg+angdegSpan);   //下一弧度
27                  //中心点
28                  vertices[count++]=0;                                   //顶点坐标
29                  vertices[count++]=0;
30                  vertices[count++]=0;
31                  textures[stCount++]=0.5f;                              //st 坐标
32                  textures[stCount++]=0.5f;
33                  //当前点
34                  vertices[count++]=(float) (-r*Math.sin(angrad));       //顶点坐标
35                  vertices[count++]=(float) (r*Math.cos(angrad));
36                  vertices[count++]=0;
37                  textures[stCount++]=(float) (0.5f-0.5f*Math.sin(angrad));//st 坐标
38                  textures[stCount++]=(float) (0.5f-0.5f*Math.cos(angrad));
39                  //下一点
40                  vertices[count++]=(float) (-r*Math.sin(angradNext));   //顶点坐标
41                  vertices[count++]=(float) (r*Math.cos(angradNext));
42                  vertices[count++]=0;
43                  textures[stCount++]=(float) (0.5f-0.5f*Math.sin(angradNext));//st 坐标
44                  textures[stCount++]=(float) (0.5f-0.5f*Math.cos(angradNext));
45              }
46              ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                        //创建顶点坐标数据缓冲
47          vbb.order(ByteOrder.nativeOrder());         //设置字节顺序
48          vertexBuffer = vbb.asFloatBuffer();         //转换为 float 型缓冲
49          vertexBuffer.put(vertices);                 //向缓冲区中放入顶点坐标数据
50          vertexBuffer.position(0);                   //设置缓冲区起始位置
51          //法向量数据初始化
52          float[] normals=new float[vertices.length];
53          for(int i=0;i<normals.length;i+=3){
54              normals[i]=0;
55              normals[i+1]=0;
56              normals[i+2]=1;
57          }
58          ByteBuffer nbb = ByteBuffer.allocateDirect(normals.length*4);
                                                        //创建顶点法向量数据缓冲
59          nbb.order(ByteOrder.nativeOrder());         //设置字节顺序
60          normalBuffer = nbb.asFloatBuffer();         //转换为 float 型缓冲
61          normalBuffer.put(normals);                  //向缓冲区中放入顶点法向量数据
62          normalBuffer.position(0);                   //设置缓冲区起始位置
63          //st 坐标数据初始化
64          ByteBuffer cbb = ByteBuffer.allocateDirect(textures.length*4);
                                                        //创建顶点纹理数据缓冲
65          cbb.order(ByteOrder.nativeOrder());         //设置字节顺序
66          textureBuffer = cbb.asFloatBuffer();        //转换为 float 型缓冲
67          textureBuffer.put(textures);                //向缓冲区中放入顶点纹理数据
68          textureBuffer.position(0);                  //设置缓冲区起始位置
69        }
70      public void drawSelf(GL10 gl){
71        gl.glPushMatrix();
72        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);           //启用顶点坐标数组
73          gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);         //启用顶点法向量数组
74          gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);  //启用顶点纹理数组
75          //绕轴旋转
76          gl.glRotatef(xAngle, 1, 0, 0);
77          gl.glRotatef(yAngle, 0, 1, 0);
78          gl.glRotatef(zAngle, 0, 0, 1);
79              //为画笔指定顶点坐标数据
80          gl.glVertexPointer(
81                  3,                                      //每个顶点的坐标数量为 3  xyz
82                  GL10.GL_FLOAT,                          //顶点坐标值的类型为 GL_FLOAT
83                  0,                                      //连续顶点坐标数据之间的间隔
84                  vertexBuffer                            //顶点坐标数据
```

```
85          );
86          //为画笔指定顶点法向量数据
87          gl.glNormalPointer(GL10.GL_FLOAT, 0, normalBuffer);
88          //为画笔指定纹理 ST 坐标缓冲
89          gl.glEnable(GL10.GL_TEXTURE_2D);                    //开启纹理
90          gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, textureBuffer);
                                                                //为画笔指定纹理 ST 坐标缓冲
91          gl.glBindTexture(GL10.GL_TEXTURE_2D, textureId);    //绑定当前纹理
92          //绘制图形
93          gl.glDrawArrays(
94                  GL10.GL_TRIANGLES,                          //以三角形方式填充
95                  0,                                          //开始点编号
96                  vCount                                      //顶点的数量
97          );
98          gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);      //禁用顶点坐标数组
99          gl.glDisableClientState(GL10.GL_NORMAL_ARRAY);      //禁用顶点法向量数组
100         gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY); //禁用顶点纹理数组
101         gl.glPopMatrix();
102   }}
```

- 代码第 3-12 行为该类中成员变量的定义，其中有顶点、纹理及法向量数据缓冲，第 12 行中 textureId 是纹理 id，该纹理 id 是用系统的方法自动生成的，代表一个图片资源的 id。
- 代码第 13-69 行为构造器的开发，在构造器中初始化某些成员变量之后，又初始化了纹理矩形的顶点坐标，和纹理坐标和法向量。注意顶点坐标、纹理坐标和法向量都是相对应的。
- 代码第 70-102 行为绘制纹理矩形的方法，绘制前开启纹理数组、为画笔指定纹理 ST 坐标缓冲并绑定当前纹理，便实现了纹理贴图。

提示 由于圆的绘制比较简单，顶点、纹理及法向量很容易算出，因此请读者自己分析，由于篇幅所限，这里不再赘述。后面小节中若有类似圆的基本类，将只给出其构造器。

（2）CylinderSide 类是圆柱的柱面，该类也有顶点、纹理和法向量，其构造器的代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_9/src/com/bn/game/chap11/ex9 目录下的 CylinderSide.java。

```
1       public CylinderSide(float scale,float r, float h, int n, int textureId) {
                                                        //大小、半径、高度、边数、纹理 id
2           this.textureId=textureId;
3           size=Constant.UNIT_SIZE*scale;              //改变尺寸
4           r*=size;
5           h*=size;
6           angdegSpan=360.0f/n;
7           vCount=3*n*4;          //顶点个数，共有 3*n*4 个三角形，每个三角形都有 3 个顶点
8           //坐标数据初始化
9           float[] vertices=new float[vCount*3];
10          float[] textures=new float[vCount*2];       //顶点纹理 S、T 坐标值数组
11          //坐标数据初始化
12          int count=0;
13          int stCount=0;
14          for(float angdeg=0;Math.ceil(angdeg)<360;angdeg+=angdegSpan){
15              double angrad=Math.toRadians(angdeg);           //当前弧度
16              double angradNext=Math.toRadians(angdeg+angdegSpan);   //下一弧度
17                                                              //底圆当前点---0
18              vertices[count++]=(float) (-r*Math.sin(angrad));
19              vertices[count++]=0;
20              vertices[count++]=(float) (-r*Math.cos(angrad));
```

```
21                  textures[stCount++]=(float) (angrad/(2*Math.PI)); //st 坐标
22                  textures[stCount++]=1;
23                                                          //顶圆下一点---3
24                  vertices[count++]=(float) (-r*Math.sin(angradNext));
25                  vertices[count++]=h;
26                  vertices[count++]=(float) (-r*Math.cos(angradNext));
27                  textures[stCount++]=(float) (angradNext/(2*Math.PI)); //st 坐标
28                  textures[stCount++]=0;
29                                                          //顶圆当前点---2
30                  vertices[count++]=(float) (-r*Math.sin(angrad));
31                  vertices[count++]=h;
32                  vertices[count++]=(float) (-r*Math.cos(angrad));
33                  textures[stCount++]=(float) (angrad/(2*Math.PI)); //st 坐标
34                  textures[stCount++]=0;
35                                                          //底圆当前点---0
36                  vertices[count++]=(float) (-r*Math.sin(angrad));
37                  vertices[count++]=0;
38                  vertices[count++]=(float) (-r*Math.cos(angrad));
39                  textures[stCount++]=(float) (angrad/(2*Math.PI)); //st 坐标
40                  textures[stCount++]=1;
41                                                          //底圆下一点---1
42                  vertices[count++]=(float) (-r*Math.sin(angradNext));
43                  vertices[count++]=0;
44                  vertices[count++]=(float) (-r*Math.cos(angradNext));
45                  textures[stCount++]=(float) (angradNext/(2*Math.PI)); //st 坐标
46                  textures[stCount++]=1;
47                                                          //顶圆下一点---3
48                  vertices[count++]=(float) (-r*Math.sin(angradNext));
49                  vertices[count++]=h;
50                  vertices[count++]=(float) (-r*Math.cos(angradNext));
51                  textures[stCount++]=(float) (angradNext/(2*Math.PI)); //st 坐标
52                  textures[stCount++]=0;
53              }
54              ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                            //创建顶点坐标数据缓冲
55          vbb.order(ByteOrder.nativeOrder());             //设置字节顺序
56          vertexBuffer = vbb.asFloatBuffer();             //转换为 float 型缓冲
57          vertexBuffer.put(vertices);                     //向缓冲区中放入顶点坐标数据
58          vertexBuffer.position(0);                       //设置缓冲区起始位置
59          //法向量数据初始化
60          for(int i=0;i<vertices.length;i++){
61              if(vertices[i]==h){
62                  vertices[i]=0;                          //将所有顶点的 y 坐标换成 0
63              }
64          }
65          ByteBuffer nbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                            //创建顶点法向量数据缓冲
66          nbb.order(ByteOrder.nativeOrder());             //设置字节顺序
67          normalBuffer = nbb.asFloatBuffer();             //转换为 float 型缓冲
68          normalBuffer.put(vertices);                     //向缓冲区中放入顶点法向量数据
69          normalBuffer.position(0);                       //设置缓冲区起始位置
70          //st 坐标数据初始化
71          ByteBuffer cbb = ByteBuffer.allocateDirect(textures.length*4);
                                                            //创建顶点纹理数据缓冲
72          cbb.order(ByteOrder.nativeOrder());             //设置字节顺序
73          textureBuffer = cbb.asFloatBuffer();            //转换为 float 型缓冲
74          textureBuffer.put(textures);                    //向缓冲区中放入顶点纹理数据
75          textureBuffer.position(0);                      //设置缓冲区起始位置
76      }
```

- 初始化顶点的思想为，从上下两圆中各取两个点组成一个矩形，再将该矩形分成两个三角

形，卷绕两个三角形后，再处理下一个矩形，做同样的操作即可。

- 初始化纹理坐标数据很简单，即把一个纹理矩形图片切分成小块三角形，依次对应柱面的各个三角形，注意其中纹理坐标是通过角度的比例来确定的。
- 初始化法向量坐标的思想为，将顶点坐标数据中的数据保留 x、z 坐标的值，将 y 坐标全部置为 0，这是由于柱面的法向量就是垂直于该点并且 y 为 0 的点的坐标。

（3）Cylinder 类是由上述两个类的对象组成的，一个圆柱体由一个侧面和两个圆面组成，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_9/src/com/bn/game/chap11/ex9 目录下的 Cylinder.java。

```
1    package com.bn.game.chap11.ex9;                                  //声名包
2    import javax.microedition.khronos.opengles.GL10;                 //引入相关类
3    public class Cylinder{
4         Circle bottomCircle;                                        //底圆
5         Circle topCircle;                                           //顶圆
6         CylinderSide cylinderSide;                                  //侧面
7         float xAngle=0;                                             //绕 x 轴旋转的角度
8       float yAngle=0;                                               //绕 y 轴旋转的角度
9       float zAngle=0;                                               //绕 z 轴旋转的角度
10      float size;                                                   //尺寸
11      float h;
12        public Cylinder(float scale,float r, float h, int n,
13                  int topTexId, int BottomTexId, int sideTexId){
14             topCircle=new Circle(scale,r,n, topTexId);                //顶圆
15             bottomCircle=new Circle(scale,r,n, BottomTexId);          //底圆
16             cylinderSide=new CylinderSide(scale, r,  h, n, sideTexId);  //侧面
17             // 初始化完成后再改变各量的值
18             size=Constant.UNIT_SIZE*scale;
19             this.h=h*size;
20        }
21        public void drawSelf(GL10 gl){
22             gl.glRotatef(xAngle, 1, 0, 0);                            //旋转
23         gl.glRotatef(yAngle, 0, 1, 0);
24         gl.glRotatef(zAngle, 0, 0, 1);
25                                                                       //顶面
26             gl.glPushMatrix();
27             gl.glTranslatef(0, h, 0);
28             gl.glRotatef(-90, 1, 0, 0);
29             topCircle.drawSelf(gl);                                   //绘制
30             gl.glPopMatrix();
31                                                                       //底面
32             gl.glPushMatrix();
33             gl.glRotatef(90, 1, 0, 0);
34             gl.glRotatef(180, 0, 0, 1);
35             bottomCircle.drawSelf(gl);                                //绘制
36             gl.glPopMatrix();
37                                                                       //侧面
38             gl.glPushMatrix();
```

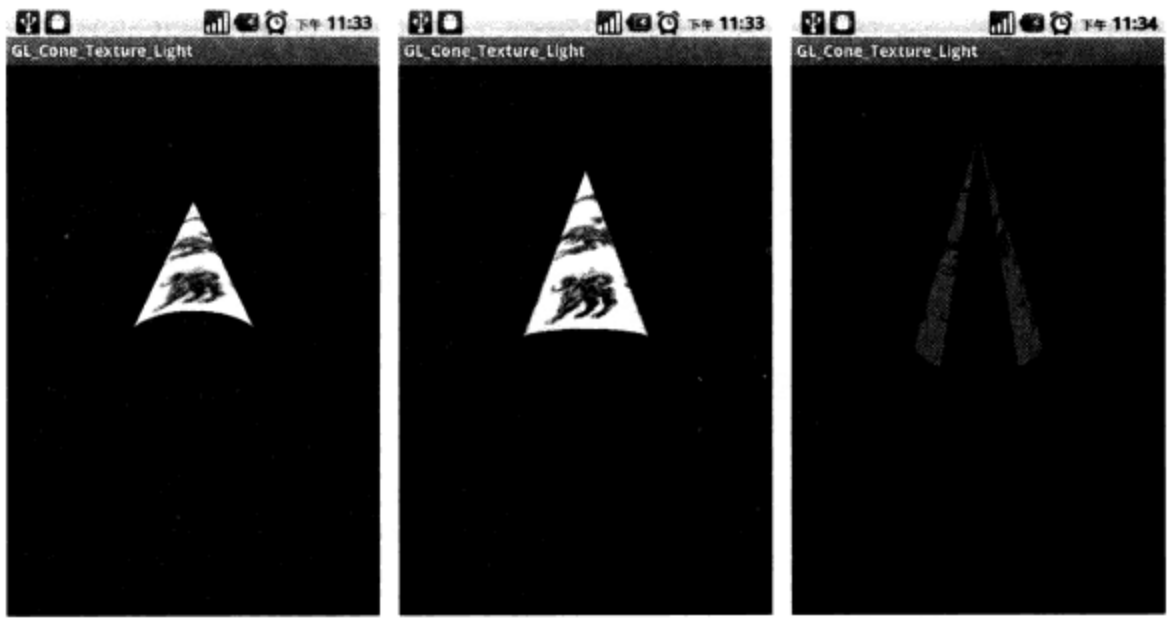
```
39                cylinderSide.drawSelf(gl);                          //绘制
40                gl.glPopMatrix();
41    }}
```

● 代码第 4-20 行为该类的成员变量及构造器，从中可以看出，圆柱是由一个顶面、一个底面和一个侧面组成，这充分体现了面向对象的思想。

● 代码第 21-41 行是绘制组成圆柱的各个组成部分的代码，在绘制每一部分前，要先将各个部分平移并旋转特定的位置，以保证组成整个圆柱。

11.6.2　圆锥

相对于圆柱而言，圆锥的开发相对复杂，其有一个圆锥面和一个圆形底面组成，而其法向量涉及数学知识，不易理解，这里不为其加光照，只有顶点和纹理。下面介绍本小节的案例 Sample_11_10，其进行效果如图 11-40 所示。



▲图 11-40　案例 Sample_11_10 的运行效果图

接下来介绍实现本案例的代码。在这里讲解有关关几何体组成的几个核心类 ConeSide、Cone，下面对两个类一一进行讲解。

（1）ConeSide 类是圆锥侧面类，该类有顶点、纹理和法向量，其构造器的代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_10/src/com/bn/game/chap11/ex10 目录下的 ConeSide.java。

```
1         public ConeSide(float scale,float r, float h, int n, int textureId) {
                                                          //大小、半径、高度、边数、纹理 id
2              this.textureId=textureId;
3                                                         //改变尺寸
4              size=Constant.UNIT_SIZE*scale;
5              r*=size;
6              h*=size;
7              angdegSpan=360.0f/n;
8              vCount=3*n*4;//顶点个数，共有 3*n*4 个三角形，每个三角形都有 3 个顶点
9                                                         //坐标数据初始化
10             float[] vertices=new float[vCount*3];
11             float[] textures=new float[vCount*2];      //顶点纹理 S、T 坐标值数组
```

```
12              float[] normals=new float[vertices.length];          //法向量数组
13              //坐标数据初始化
14              int count=0;
15              int stCount=0;
16              int norCount=0;
17              for(float angdeg=0;Math.ceil(angdeg)<360;angdeg+=angdegSpan)     //侧面
18              {
19                   double angrad=Math.toRadians(angdeg);//当前弧度
20                   double angradNext=Math.toRadians(angdeg+angdegSpan);   //下一弧度
21                   //中心点
22                   vertices[count++]=0;
23                   vertices[count++]=h;
24                   vertices[count++]=0;
25
26                   textures[stCount++]=0.5f;                              //st 坐标
27                   textures[stCount++]=0;
28                   //当前点
29                   vertices[count++]=(float) (-r*Math.sin(angrad));
30                   vertices[count++]=0;
31                   vertices[count++]=(float) (-r*Math.cos(angrad));
32
33                   textures[stCount++]=(float) (angrad/(2*Math.PI));      //st 坐标
34                   textures[stCount++]=1;
35                   //下一点
36                   vertices[count++]=(float) (-r*Math.sin(angradNext));
37                   vertices[count++]=0;
38                   vertices[count++]=(float) (-r*Math.cos(angradNext));
39
40                   textures[stCount++]=(float) (angradNext/(2*Math.PI));  //st 坐标
41                   textures[stCount++]=1;
42
43                   //法向量数据初始化
44                   float [] norXYZ=VectorUtil.calConeNormal( //通过 3 个顶点求出法向量
45                             0, 0, 0,                          //底面圆心坐标
46                             vertices[count-9], vertices[count-8], vertices[count-7],
47                             vertices[count-6], vertices[count-5], vertices[count-4]);
48                   for(int i=0;i<3;i++){                       //顶点和当前点的法向量
49                        normals[norCount++]=norXYZ[0];
50                        normals[norCount++]=norXYZ[1];
51                        normals[norCount++]=norXYZ[2];
52                   }
53                   for(int i=0;i<2;i++){                       //三角形 3 个点法向量都相同
54                        normals[norCount++]=norXYZ[0];
55                        normals[norCount++]=norXYZ[1];
56                        normals[norCount++]=norXYZ[2];
57                   }
58                   norXYZ=VectorUtil.calConeNormal(            //通过 3 个顶点求出法向量
59                             0, 0, 0,                          //底面圆心坐标
60                             vertices[count-9], vertices[count-8], vertices[count-7],
61                             vertices[count-3], vertices[count-2], vertices[count-1]);
62                   for(int i=0;i<1;i++){                       //下一点的法向量
63                        normals[norCount++]=norXYZ[0];
64                        normals[norCount++]=norXYZ[1];
65                        normals[norCount++]=norXYZ[2];
66                   }
67              }
68              ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                                //创建顶点坐标数据缓冲
69          vbb.order(ByteOrder.nativeOrder());                 //设置字节顺序
70          vertexBuffer = vbb.asFloatBuffer();                 //转换为 float 型缓冲
71          vertexBuffer.put(vertices);                 //向缓冲区中放入顶点坐标数据
72          vertexBuffer.position(0);                           //设置缓冲区起始位置
```

```
73          //法向量数据初始化
74          ByteBuffer nbb = ByteBuffer.allocateDirect(normals.length*4);
                                                        //创建顶点法向量数据缓冲
75          nbb.order(ByteOrder.nativeOrder());         //设置字节顺序
76          normalBuffer = nbb.asFloatBuffer();         //转换为 float 型缓冲
77          normalBuffer.put(normals);              //向缓冲区中放入顶点法向量数据
78          normalBuffer.position(0);                   //设置缓冲区起始位置
79          //st 坐标数据初始化
80          ByteBuffer cbb = ByteBuffer.allocateDirect(textures.length*4);
                                                        //创建顶点纹理数据缓冲
81          cbb.order(ByteOrder.nativeOrder());         //设置字节顺序
82          textureBuffer = cbb.asFloatBuffer();        //转换为 float 型缓冲
83          textureBuffer.put(textures);            //向缓冲区中放入顶点纹理数据
84          textureBuffer.position(0);                  //设置缓冲区起始位置
85        }
```

- 初始化顶点的思想和圆非常类似，只是把圆心的位置调节到了圆锥的顶点处。
- 初始化纹理坐标数据的思想也是把一个纹理矩形图片切分成小块三角形，依次对应柱面的各个三角形，不过由于圆锥的特殊性，不能把所有纹理都贴上去，而只是贴上一个三角形区域。
- 圆锥面的法向量是通过自己 VectorUtil 类中 calConeNormal 方法算出的，该方法传递的参数为圆锥的底面圆心坐标、顶点坐标、圆上一点坐标。

（3）Cone 类是由圆类和圆锥侧面两个类的对象组成的，一个圆柱体由一个圆形底面和一个圆锥侧面组成，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_10/src/com/bn/game/chap11/ex10 目录下的 Cone.java。

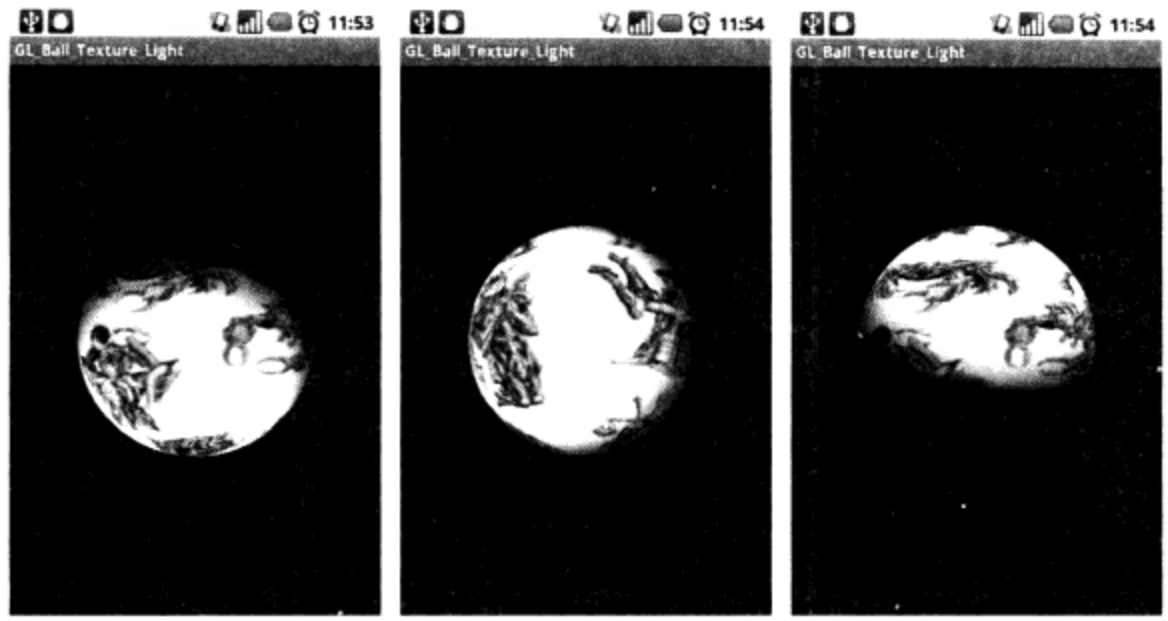
```
1     package com.bn.game.chap11.ex10;                  //声名包
2     import javax.microedition.khronos.opengles.GL10;  //引入相关类
3     public class Cone{
4           Circle bottomCircle;                        /底圆
5           ConeSide coneSide;                          //侧面
6           float xAngle=0;                             //绕 x 轴旋转的角度
7         float yAngle=0;                               //绕 y 轴旋转的角度
8         float zAngle=0;                               //绕 z 轴旋转的角度
9         float size;                                   //尺寸
10        float h;
11          public Cone(float scale,float r, float h, int n,
12                    int BottomTexId, int sideTexId){
13              bottomCircle=new Circle(scale,r,n, BottomTexId);
14              coneSide=new ConeSide(scale, r,  h, n, sideTexId);
15                                                  // 初始化完成后再改变各量的值
16              size=Constant.UNIT_SIZE*scale;
17              this.h=h*size;
18          }
19          public void drawSelf(GL10 gl){
20              gl.glRotatef(xAngle, 1, 0, 0);          //旋转
21            gl.glRotatef(yAngle, 0, 1, 0);
22            gl.glRotatef(zAngle, 0, 0, 1);
23                                                      //底面
24              gl.glPushMatrix();
25              gl.glRotatef(90, 1, 0, 0);
26              gl.glRotatef(180, 0, 0, 1);
27              bottomCircle.drawSelf(gl);              //绘制
28              gl.glPopMatrix();
29                                                      //侧面
30              gl.glPushMatrix();
```

```
31                  coneSide.drawSelf(gl);                          //绘制
32                  gl.glPopMatrix();
33      }}
```

- 代码第 4-18 行为该类的成员变量及构造器，从中可以看出，圆锥是由一个底面和一个侧面组成。
- 代码第 19-32 行是绘制组成圆锥的各个组成部分的代码，在绘制每一部分前要先将各个部分平移并旋转特定的位置，以保证组成整个圆锥。

11.6.3 球

球的代码虽然简短，但其原理比圆柱和圆锥都要复杂，下面介绍本小节的案例 Sample_11_11，其运行效果如图 11-41 所示。



▲图 11-41　案例 Sample_11_11 的运行效果图

接下来介绍实现本案例的代码。在这里讲解有关几何体组成的核心类 Ball，其构造器代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_11/src/com/bn/game/chap11/ex11 目录下的 Ball.java。

```
1     public Ball(float scale,float r, int nCol ,int nRow, int textureId) {
                                                          //大小、半径、高度、边数、纹理 id
2          this.textureId=textureId;
3          size=Constant.UNIT_SIZE*scale;                 //改变尺寸
4          r*=size;
5          angdegColSpan=360.0f/nCol;
6          angdegRowSpan=180.0f/nRow;
7          vCount=3*nCol*nRow*2;//顶点个数，共有 nColumn*nRow*2 个三角形，每个三角形都有 3 个顶点
8          //坐标数据初始化
9          float[] vertices=new float[vCount*3];
10         float[] textures=new float[vCount*2];          //顶点纹理 S、T 坐标值数组
11         //坐标数据初始化
12         int count=0;
13         int stCount=0;
14         for(float angdegCol=0;Math.ceil(angdegCol)<360;angdegCol+=angdegColSpan){
15              double angradCol=Math.toRadians(angdegCol);     //当前列弧度
16              double angradColNext=Math.toRadians(angdegCol+angdegColSpan);
                                                          //下一列弧度
```

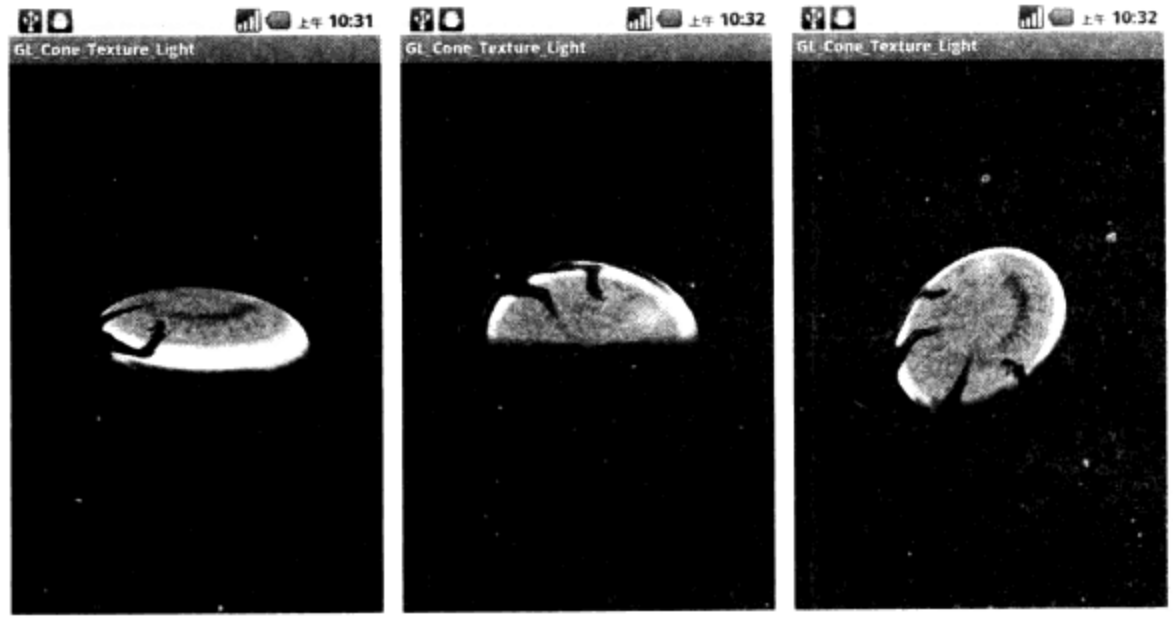
```
17              for(float angdegRow=0;Math.ceil(angdegRow)<180;angdegRow+=angdegRowSpan){
18                  double angradRow=Math.toRadians(angdegRow);          //当前行弧度
19                  double angradRowNext=Math.toRadians(angdegRow+angdegRowSpan);
                                                                         //下一行弧度
20                  float rCircle=(float) (r*Math.sin(angradRow)); //当前行上圆的半径
21                  float rCircleNext=(float) (r*Math.sin(angradRowNext));
                                                                     //下一行上圆的半径
22                  //当前行，当前列---0
23                  vertices[count++]=(float) (-rCircle*Math.sin(angradCol));
24                  vertices[count++]=(float) (r*Math.cos(angradRow));
25                  vertices[count++]=(float) (-rCircle*Math.cos(angradCol));
26                  textures[stCount++]=(float) (angradCol/(2*Math.PI));   //st 坐标
27                  textures[stCount++]=(float) (angradRow/Math.PI);
28                  //下一行，当前列---2
29                  vertices[count++]=(float) (-rCircleNext*Math.sin(angradCol));
30                  vertices[count++]=(float) (r*Math.cos(angradRowNext));
31                  vertices[count++]=(float) (-rCircleNext*Math.cos(angradCol));
32                  textures[stCount++]=(float) (angradCol/(2*Math.PI));   //st 坐标
33                  textures[stCount++]=(float) (angradRowNext/Math.PI);
34                  //下一行，下一列---3
35                  vertices[count++]=(float) (-rCircleNext*Math.sin(angradColNext));
36                  vertices[count++]=(float) (r*Math.cos(angradRowNext));
37                  vertices[count++]=(float) (-rCircleNext*Math.cos(angradColNext));
38                  textures[stCount++]=(float) (angradColNext/(2*Math.PI));//st 坐标
39                  textures[stCount++]=(float) (angradRowNext/Math.PI);
40                  //当前行，当前列---0
41                  vertices[count++]=(float) (-rCircle*Math.sin(angradCol));
42                  vertices[count++]=(float) (r*Math.cos(angradRow));
43                  vertices[count++]=(float) (-rCircle*Math.cos(angradCol));
44                  textures[stCount++]=(float) (angradCol/(2*Math.PI));   //st 坐标
45                  textures[stCount++]=(float) (angradRow/Math.PI);
46                  //下一行，下一列---3
47                  vertices[count++]=(float) (-rCircleNext*Math.sin(angradColNext));
48                  vertices[count++]=(float) (r*Math.cos(angradRowNext));
49                  vertices[count++]=(float) (-rCircleNext*Math.cos(angradColNext));
50                  textures[stCount++]=(float) (angradColNext/(2*Math.PI));//st 坐标
51                  textures[stCount++]=(float) (angradRowNext/Math.PI);
52                  //当前行，下一列---1
53                  vertices[count++]=(float) (-rCircle*Math.sin(angradColNext));
54                  vertices[count++]=(float) (r*Math.cos(angradRow));
55                  vertices[count++]=(float) (-rCircle*Math.cos(angradColNext));
56                  textures[stCount++]=(float) (angradColNext/(2*Math.PI));//st 坐标
57                  textures[stCount++]=(float) (angradRow/Math.PI);
58          }}
59          ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                                     //创建顶点坐标数据缓冲
60            vbb.order(ByteOrder.nativeOrder());                    //设置字节顺序
61            vertexBuffer = vbb.asFloatBuffer();                    //转换为 float 型缓冲
62            vertexBuffer.put(vertices);                     //向缓冲区中放入顶点坐标数据
63            vertexBuffer.position(0);                       //设置缓冲区起始位置
64            //法向量数据初始化
65            for(int i=0;i<vertices.length;i+=3){                //将顶点向量规格化
66                float[] vec=VectorUtil.normalizeVector(vertices[i], vertices[i+1],
                  vertices[i+2]);
67                vertices[i]=vec[0];
68                vertices[i+1]=vec[1];
69                vertices[i+2]=vec[2];
70            }
71            ByteBuffer nbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                                 //创建顶点法向量数据缓冲
72            nbb.order(ByteOrder.nativeOrder());                //设置字节顺序
73            normalBuffer = nbb.asFloatBuffer();                //转换为 float 型缓冲
```

```
74          normalBuffer.put(vertices);                    //向缓冲区中放入顶点法向量数据
75          normalBuffer.position(0);                      //设置缓冲区起始位置
76          //st 坐标数据初始化
77          ByteBuffer tbb = ByteBuffer.allocateDirect(textures.length*4);
                                                           //创建顶点纹理数据缓冲
78          tbb.order(ByteOrder.nativeOrder());            //设置字节顺序
79          textureBuffer = tbb.asFloatBuffer();           //转换为 float 型缓冲
80          textureBuffer.put(textures);                   //向缓冲区中放入顶点纹理数据
81          textureBuffer.position(0);                     //设置缓冲区起始位置
82      }
```

- 初始化顶点的思想为，将球按地球的经纬度来分割，分割后的小块为一个四边形，再将该四边形分解成两个三角形。
- 初始化纹理坐标数据的思想为，把一个纹理矩形图片横竖切分成小块三角形，依次对应球面的各个三角形，其中纹理坐标是通过经纬度角度的比例来确定的。
- 初始化法向量坐标非常简单，由于球体法向量的特殊，并且本案例中球心在原点，所以法向量的数据就是顶点坐标数据。

11.6.4 椭圆体

椭圆体和球类似但光照所用的法向量要比球法向量复杂，下面介绍本小节的案例 Sample_11_12，其运行效果如图 11-42 所示。



▲图 11-42　案例 Sample_11_12 的运行效果图

接下来介绍实现本案例的代码。在这里讲解有关几何体组成的核心类 Spheroid，其构造器代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_12/src/com/bn/game/chap11/ex12 目录下的 Spheroid.java。

```
1       public Spheroid(float scale, float a, float b, float c,
2                 int nCol ,int nRow, int textureId) {  //大小、半径、高度、边数、纹理 id
3           this.textureId=textureId;
4                                                        //改变尺寸
5           size=Constant.UNIT_SIZE*scale;
6           a*=size;
```

```
7            b*=size;
8            c*=size;
9            this.b=b;                                        //成员变量赋值
10           angdegColSpan=360.0f/nCol;
11           angdegRowSpan=180.0f/nRow;
12           vCount=3*nCol*nRow*2;//顶点个数，共有 nColumn*nRow*2 个三角形，每个三角形都有
                                  //3 个顶点
13           //坐标数据初始化
14           float[] vertices=new float[vCount*3];
15           float[] textures=new float[vCount*2];      //顶点纹理 S、T 坐标值数组
16           //坐标数据初始化
17           int count=0;
18           int stCount=0;
19           for(float angdegCol=0;Math.ceil(angdegCol)<360;angdegCol+=angdegColSpan)
20           {//---经度
21                double angradCol=Math.toRadians(angdegCol);        //当前列弧度
22                double angradColNext=Math.toRadians(angdegCol+angdegColSpan);
                                  //下一列弧度
23                for(float angdegRow=-90;Math.ceil(angdegRow)<90;angdegRow+
                  =angdegRowSpan)
24                {//---纬度
25                     double angradRow=Math.toRadians(angdegRow);   //当前行弧度
26                     double angradRowNext=Math.toRadians(angdegRow+
                       angdegRowSpan);//下一行弧度
27                     //当前行，当前列---0
28                     vertices[count++]=(float) (a*Math.cos(angradRow)*Math.
                       cos(angradCol));
29                     vertices[count++]=(float) (b*Math.sin(angradRow));
30                     vertices[count++]=(float) (c*Math.cos(angradRow)*Math.
                       sin(angradCol));
31
32                     textures[stCount++]=(float) (1-angradCol/(2*Math.PI));//st 坐标
33                     textures[stCount++]=(float) (1-(angradRow+Math.PI/2)/Math.PI);
34                     //下一行，当前列---2
35                     vertices[count++]=(float) (a*Math.cos(angradRowNext)*Math.
                       cos(angradCol));
36                     vertices[count++]=(float) (b*Math.sin(angradRowNext));
37                     vertices[count++]=(float) (c*Math.cos(angradRowNext)*Math.
                       sin(angradCol));
38
39                     textures[stCount++]=(float) (1-angradCol/(2*Math.PI));//st 坐标
40                     textures[stCount++]=(float) (1-(angradRowNext+Math.PI/2)
                       /Math.PI);
41                     //下一行，下一列---3
42                     vertices[count++]=(float) (a*Math.cos(angradRowNext)*Math.
                       cos(angradColNext));
43                     vertices[count++]=(float) (b*Math.sin(angradRowNext));
44                     vertices[count++]=(float) (c*Math.cos(angradRowNext)*Math.
                       sin(angradColNext));
45
46                     textures[stCount++]=(float) (1-angradColNext/(2*Math.PI));
                       //st 坐标
47                     textures[stCount++]=(float) (1-(angradRowNext+Math.PI/2)
                       /Math.PI);
48
49
50                     //当前行，当前列---0
51                     vertices[count++]=(float) (a*Math.cos(angradRow)*Math.cos
                       (angradCol));
52                     vertices[count++]=(float) (b*Math.sin(angradRow));
53                     vertices[count++]=(float) (c*Math.cos(angradRow)*Math.sin
                       (angradCol));
```

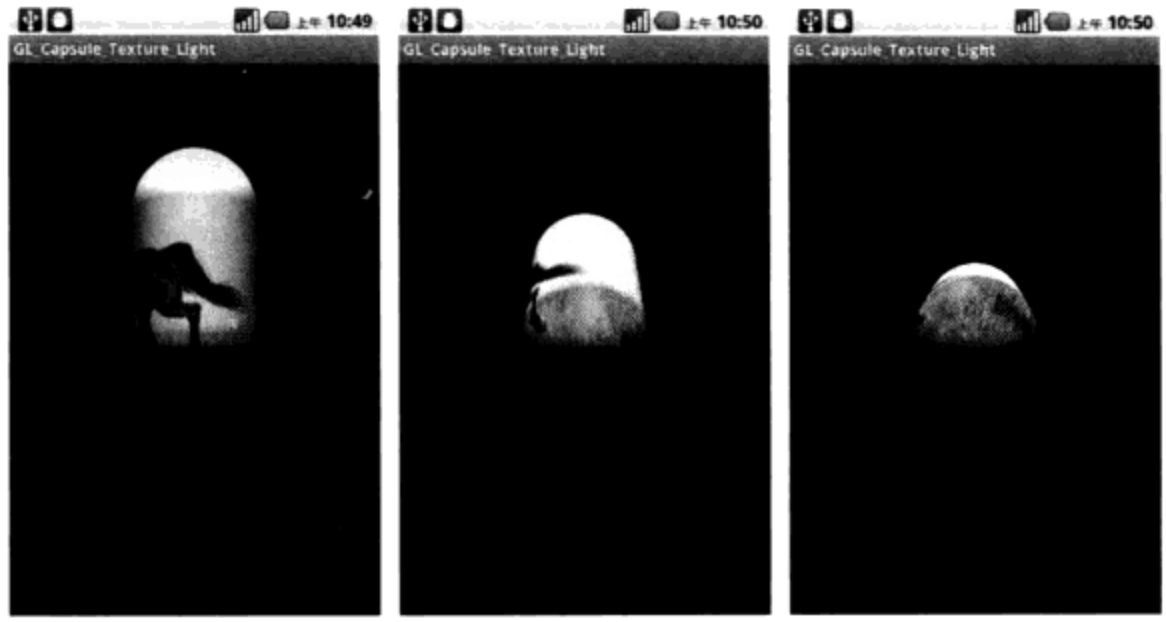
```
54
55                          textures[stCount++]=(float) (1-angradCol/(2*Math.PI));//st 坐标
56                          textures[stCount++]=(float) (1-(angradRow+Math.PI/2)/
                            Math.PI);
57                          //下一行，下一列---
58                          vertices[count++]=(float) (a*Math.cos(angradRowNext)*Math.
                            cos(angradColNext));
59                          vertices[count++]=(float) (b*Math.sin(angradRowNext));
60                          vertices[count++]=(float) (c*Math.cos(angradRowNext)*Math.
                            sin(angradColNext));
61
62                          textures[stCount++]=(float) (1-angradColNext/(2*Math.PI));
                            //st 坐标
63                          textures[stCount++]=(float) (1-(angradRowNext+Math.PI/2)
                            /Math.PI);
64                          //当前行，下一列---1
65                          vertices[count++]=(float) (a*Math.cos(angradRow)*Math.cos
                            (angradColNext));
66                          vertices[count++]=(float) (b*Math.sin(angradRow));
67                          vertices[count++]=(float) (c*Math.cos(angradRow)*Math.sin
                            (angradColNext));
68
69                          textures[stCount++]=(float) (1-angradColNext/(2*Math.PI));
                            //st 坐标
70                          textures[stCount++]=(float) (1-(angradRow+Math.PI/2)/Math.PI);
71                      }
72                  }
73                  ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                                //创建顶点坐标数据缓冲
74              vbb.order(ByteOrder.nativeOrder());             //设置字节顺序
75              vertexBuffer = vbb.asFloatBuffer();             //转换为 float 型缓冲
76              vertexBuffer.put(vertices);                     //向缓冲区中放入顶点坐标数据
77              vertexBuffer.position(0);                       //设置缓冲区起始位置
78                                                              //法向量数据初始化
79                  float[] normals=new float[vertices.length];//法向量数组
80                  int norCount=0;
81                  for(int i=0;i<vertices.length;i+=3){
82                      normals[norCount++]=vertices[i+0]/(a*a);
83                      normals[norCount++]=vertices[i+1]/(b*b);
84                      normals[norCount++]=vertices[i+2]/(c*c);
85                  }
86                  VectorUtil.normalizeAllVectors(vertices);  //规格化法向量
87
88              ByteBuffer nbb = ByteBuffer.allocateDirect(normals.length*4);
                                                                //创建顶点法向量数据缓冲
89              nbb.order(ByteOrder.nativeOrder());             //设置字节顺序
90              normalBuffer = nbb.asFloatBuffer();             //转换为 float 型缓冲
91              normalBuffer.put(normals);                      //向缓冲区中放入顶点法向量数据
92              normalBuffer.position(0);                       //设置缓冲区起始位置
93              //st 坐标数据初始化
94              ByteBuffer cbb = ByteBuffer.allocateDirect(textures.length*4);
                                                                //创建顶点纹理数据缓冲
95              cbb.order(ByteOrder.nativeOrder());             //设置字节顺序
96              textureBuffer = cbb.asFloatBuffer();            //转换为 float 型缓冲
97              textureBuffer.put(textures);                    //向缓冲区中放入顶点纹理数据
98              textureBuffer.position(0);                      //设置缓冲区起始位置
99          }
```

- 初始化顶点的思想和球的非常相似，也是将球按地球的经纬度来分割，分割后的小块为一个四边形，再将该四边形分解成两个三角形。

- 初始化纹理坐标数据的思想为，把一个纹理矩形图片横竖切分成小块三角形，依次对应球面的各个三角形，其中纹理坐标是通过经纬度角度的比例来确定的。
- 初始化法向量坐标与球的非常类似，根据椭圆体上的点的坐标即可求出法向量。若椭圆体球心在原点，椭圆体上一点的坐标为(x,y,z)，椭圆体 3 个半轴长为 a、b、c，则其法向量为(x/(a*a) ,y/(b*b), z/(c*c))。

11.6.5　胶囊体

胶囊体是由两个半椭圆体和一个柱面组成，其纹理是用一张纹理图贴成的，有一度的难度，下面介绍本小节的案例 Sample_11_13，其运行效果如图 11-43 所示。



▲图 11-43　案例 Sample_11_13 的运行效果图

接下来介绍实现本案例的代码。在这里讲解有关几何体组成的核心类 Capsule，其构造器代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_13/src/com/bn/game/chap11/ex13 目录下的 Capsule.java。

```
1        public Capsule(float scale, float r, float h, float bTop, float bBottom,
2                       int nCol ,int nRow, int textureId) {//大小，x、z 半轴长，y 半轴长，
                                                            //圆柱面高度，列数，行数
3          this.textureId=textureId;
4                                                            //改变尺寸
5              size=Constant.UNIT_SIZE*scale;
6              r*=size;
7              bTop*=size;
8              bBottom*=size;
9              h*=size;
10             this.bBottom=bBottom;
11                                                           //纹理图分割点
12             float hTotal=h+bTop+bBottom;
13             float topTexBegin=0f;
14             float topTexEnd=bTop/hTotal;
15             float centerTexBegin=topTexEnd;
```

```
16          float centerTexEnd=(bTop+h)/hTotal;
17          float bottomTexBegin=centerTexEnd;
18          float bottomTexEnd=1;
19          angdegColSpan=360.0f/nCol;
20          angdegRowSpan=90.0f/nRow;
21          vCount=3*(nCol*nRow*4+nCol*2);              //顶点个数
22                                                      //坐标数据初始化
23          float[] vertices=new float[vCount*3];
24          float[] textures=new float[vCount*2];       //顶点纹理 S、T 坐标值数组
25          float[] normals=new float[vertices.length];          //法向量数组
26          int count=0;
27          int stCount=0;
28          int norCount=0;
29          for(float  angdegCol=0;Math.ceil(angdegCol)<360;angdegCol+=angdegColSpan)
                                                                //上半椭圆体面
30          {//---经度
31               double angradCol=Math.toRadians(angdegCol);         //当前列弧度
32               double angradColNext=Math.toRadians(angdegCol+angdegColSpan);
                                                                //下一列弧度
33               for(float angdegRow=0;Math.ceil(angdegRow)<90;angdegRow+=
                 angdegRowSpan)
34               {//---纬度
35                    double angradRow=Math.toRadians(angdegRow);   //当前行弧度
36                   double angradRowNext=Math.toRadians(angdegRow+angdegRowSpan);
                      //下一行弧度
37                    //当前行，当前列---0
38                    vertices[count++]=(float) (r*Math.cos(angradRow)*Math.cos
                      (angradCol));
39                   vertices[count++]=(float) (bTop*Math.sin(angradRow))+h;
40                   vertices[count++]=(float) (r*Math.cos(angradRow)*Math.sin
                     (angradCol));
41                   textures[stCount++]=(float) (1-angradCol/(2*Math.PI));//st 坐标
42                   textures[stCount++]=topTexBegin+
43                   (float) ((Math.PI/2-angradRow)/((Math.PI/2)/(topTexEnd-top
                     TexBegin)));
44                   //下一行，当前列---2
45                   vertices[count++]=(float) (r*Math.cos(angradRowNext)*Math.
                     cos(angradCol));
46                   vertices[count++]=(float) (bTop*Math.sin(angradRowNext))+h;
47                   vertices[count++]=(float) (r*Math.cos(angradRowNext)*Math.
                     sin(angradCol));
48                   textures[stCount++]=(float) (1-angradCol/(2*Math.PI));
                     //st 坐标
49                   textures[stCount++]=topTexBegin+
50                   (float) ((Math.PI/2-angradRowNext)/((Math.PI/2)/(topTexEnd-
                     topTexBegin)));
51                   //下一行，下一列---3
52                   vertices[count++]=(float) (r*Math.cos(angradRowNext)*Math.
                     cos(angradColNext));
53                   vertices[count++]=(float) (bTop*Math.sin(angradRowNext))+h;
54                   vertices[count++]=(float) (r*Math.cos(angradRowNext)*Math.
                     sin(angradColNext));
55                   textures[stCount++]=(float) (1-angradColNext/(2*Math.PI));
                     //st 坐标
```

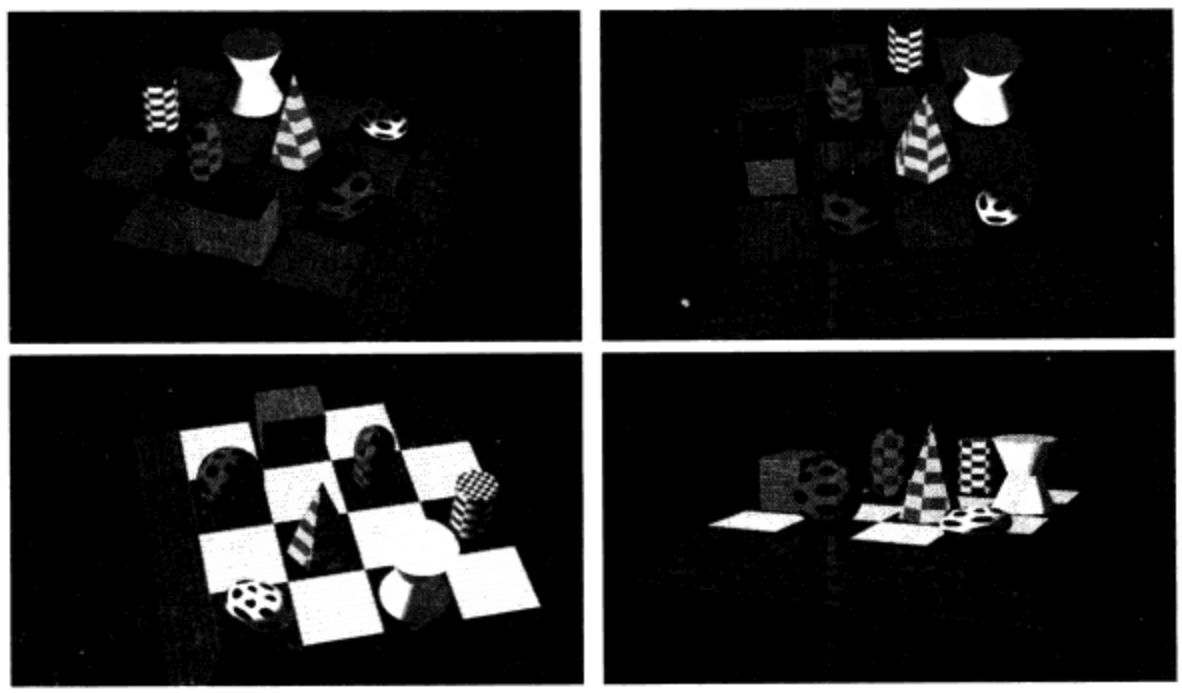
```
56                      textures[stCount++]=topTexBegin+
57                      (float) ((Math.PI/2-angradRowNext)/((Math.PI/2)/(topTexEnd-
                        topTexBegin)));
58                      //当前行，当前列---0
59                      vertices[count++]=(float) (r*Math.cos(angradRow)*Math.cos
                        (angradCol));
60                      vertices[count++]=(float) (bTop*Math.sin(angradRow))+h;
61                      vertices[count++]=(float) (r*Math.cos(angradRow)*Math.sin
                        (angradCol));
62                      textures[stCount++]=(float) (1-angradCol/(2*Math.PI));//st 坐标
63                      textures[stCount++]=topTexBegin+
64                      (float) ((Math.PI/2-angradRow)/((Math.PI/2)/(topTexEnd-top
                        TexBegin)));
65                      //下一行，下一列---3
66                      vertices[count++]=(float) (r*Math.cos(angradRowNext)*Math.
                        cos(angradColNext));
67                      vertices[count++]=(float) (bTop*Math.sin(angradRowNext))+h;
68                      vertices[count++]=(float) (r*Math.cos(angradRowNext)*Math.
                        sin(angradColNext));
69                      textures[stCount++]=(float) (1-angradColNext/(2*Math.PI));
                        //st 坐标
70                      textures[stCount++]=topTexBegin+
71                      (float) ((Math.PI/2-angradRowNext)/((Math.PI/2)/(topTexEnd-
                        topTexBegin)));
72                      //当前行，下一列---1
73                      vertices[count++]=(float) (r*Math.cos(angradRow)*Math.cos
                        (angradColNext));
74                      vertices[count++]=(float) (bTop*Math.sin(angradRow))+h;
75                      vertices[count++]=(float) (r*Math.cos(angradRow)*Math.sin
                        (angradColNext));
76                      textures[stCount++]=(float)
77                      (1-angradColNext/(2*Math.PI));            //st 坐标
78                      textures[stCount++]=topTexBegin+
79                      (float) ((Math.PI/2-angradRow)/((Math.PI/2)/(topTexEnd-
                        topTexBegin)));
80                  }}
81                  for(int i=norCount;i<count;i+=3){                   //上半球法向量初始化
82                      normals[norCount++]=vertices[i+0]/(r*r);
83                      normals[norCount++]=(vertices[i+1]-h)/(bTop*bTop);
84                      normals[norCount++]=vertices[i+2]/(r*r);
85                  }
86                  ……//该处省略了下半椭圆体面和侧面数据初始化的代码，读者可自行查看随书光盘的源代码
87                  ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                                    //创建顶点坐标数据缓冲
88              vbb.order(ByteOrder.nativeOrder());                 //设置字节顺序
89              vertexBuffer = vbb.asFloatBuffer();                 //转换为 float 型缓冲
90              vertexBuffer.put(vertices);                     //向缓冲区中放入顶点坐标数据
91              vertexBuffer.position(0);                           //设置缓冲区起始位置
92              //法向量数据初始化
93                  VectorUtil.normalizeAllVectors(normals);        //规格化法向量
94              ByteBuffer nbb = ByteBuffer.allocateDirect(normals.length*4);
                                                            //创建顶点法向量数据缓冲
95              nbb.order(ByteOrder.nativeOrder());                 //设置字节顺序
96              normalBuffer = nbb.asFloatBuffer();                 //转换为 float 型缓冲
```

```
97          normalBuffer.put(normals);                          //向缓冲区中放入顶点法向量数据
98          normalBuffer.position(0);                                   //设置缓冲区起始位置
99          //st 坐标数据初始化
100         ByteBuffer cbb = ByteBuffer.allocateDirect(textures.length*4);
                                                                //创建顶点纹理数据缓冲
101         cbb.order(ByteOrder.nativeOrder());                         //设置字节顺序
102         textureBuffer = cbb.asFloatBuffer();                //转换为 float 型缓冲
103         textureBuffer.put(textures);                        //向缓冲区中放入顶点纹理数据
104         textureBuffer.position(0);                          //设置缓冲区起始位置
105       }
```

- 代码第 12-17 行根据胶囊体两个半椭圆体 y 半轴的长度和柱面 h 计算出顶半球、柱面和底半球的纹理分割点，以便在后面把一个图分块贴在 3 个组成部分上。
- 初始化顶点和法向量的思想和椭圆体、圆柱面的法向量类似，这里不再赘述。
- 初始化纹理坐标数据在该类的实现中是最难的，其基本思想为，根据绘制到的角度值和纹理坐标的线性比例关系，求出一个计算出对应的纹理坐标的公式，然后依次套用该公式来计算出相应的纹理坐标。关键在于每一个基本形状纹理坐标的公式都不相同，甚至连椭圆体的上下半球的纹理坐标公式都不同，请读者耐心计算各个形状的纹理坐标公式。

11.6.6 几何体大集合

前面小节介绍了几种典型几何体的开发，但都是单独的一个物体。本小节将所有几何体放在一起，构成一个大的场景。下面介绍本小节的案例 Sample_11_14，其运行效果如图 11-44 所示。



▲图 11-44　案例 Sample_11_14 的运行效果图

接下来介绍实现本案例的代码。在这里只讲解主界面类 MyGLView 的绘制部分，其代码如下。

代码位置：见随书光盘中源代码/第 11 章/ Sample_11_14/src/com/bn/game/chap11/ex14 目录下的 MyGLView.java。

```
1           public void onDrawFrame(GL10 gl) {
2           gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
```

```
3            gl.glMatrixMode(GL10.GL_MODELVIEW);                    //设置当前矩阵为模式矩阵
4            gl.glLoadIdentity();                                   //设置当前矩阵为单位矩阵
5            double angradElevation=Math.toRadians(angdegElevation); //仰角（弧度）
6             double angradAzimuth=Math.toRadians(angdegAzimuth);     //方位角
7            cameraX=(float) (targetX+sightDis*Math.cos(angradElevation)*Math.cos
             (angradAzimuth));
8            cameraY=(float) (targetY+sightDis*Math.sin(angradElevation));
9            cameraZ=(float) (targetZ+sightDis*Math.cos(angradElevation)*Math.sin
             (angradAzimuth));
10           GLU.gluLookAt(                                         //设置 camera 位置
11                 gl,
12                 cameraX,                                         //人眼位置的 X
13                 cameraY,                                         //人眼位置的 Y
14                 cameraZ,                                         //人眼位置的 Z
15                 targetX,                                         //人眼看的点 X
16                 targetY,                                         //人眼看的点 Y
17                 targetZ,                                         //人眼看的点 Z
18                 0,                                               //头的朝向
19                 1,
20                 0
21           );
22                                                                  //设置定位光
23           double angradLight=Math.toRadians(angdegLight);
24           double angradZ=Math.toRadians(angdegZ);
25           float[] positionParams={
26                 (float) (-rLight*Math.cos(angradLight)*Math.sin(angradZ)),
27                 (float) (rLight*Math.sin(angradLight)),
28                 (float) (rLight*Math.cos(angradLight)*Math.cos(angradZ)),
29                 1};                                              //设置定位光
30            gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, positionParams,0);
31           gl.glTranslatef(0, 0f, -6f);
32                                                                  //立方体
33           gl.glPushMatrix();
34           gl.glTranslatef(-4.4f, cube.b/2, 1.4f);
35           cube.drawSelf(gl);
36           gl.glPopMatrix();
37                                                                  //圆柱
38           gl.glPushMatrix();
39           gl.glTranslatef(1.5f,0, -4.5f);
40           cylinder.drawSelf(gl);
41           gl.glPopMatrix();
42                                                                  //圆锥
43           gl.glPushMatrix();
44           gl.glTranslatef(4.4f,0, -1.5f);
45           cone.drawSelf(gl);
46           gl.glTranslatef(0,cone.h, 0);                          //推移
47           gl.glRotatef(180, 1, 0, 0);                            //旋转
48           cone.drawSelf(gl);                                     //再绘制一次圆锥
49           gl.glPopMatrix();
50                                                                  //球
51           gl.glPushMatrix();
52           gl.glTranslatef(-1.5f,ball.r, 4.3f);
53           ball.drawSelf(gl);
54           gl.glPopMatrix();
55                                                                  //椭圆体
56           gl.glPushMatrix();
57           gl.glTranslatef(4.2f,spheroid.b, 4.2f);
58           gl.glRotatef(45, 0, 1, 0);
59           spheroid.drawSelf(gl);
60           gl.glPopMatrix();
61                                                                  //胶囊体
62           gl.glPushMatrix();
```

```
63              gl.glTranslatef(-1.3f,capsule.bBottom,-1.3f);
64              capsule.drawSelf(gl);
65              gl.glPopMatrix();
66                                                          //梯台
67              gl.glPushMatrix();
68              gl.glTranslatef(0,-platform.h,0);
69              platform.drawSelf(gl);
70              gl.glPopMatrix();
71                                                          //正棱锥
72              gl.glPushMatrix();
73              gl.glTranslatef(1.5f,0,1.4f);
74              regularPyramid.drawSelf(gl);
75              gl.glPopMatrix();
76              }
```

- 代码第 5-21 行为设置摄像机位置的代码，摄像机的位置是通过触控屏幕事件来改变的。因此，可以用手指触控屏幕从不同角度观察场景。
- 代码第 22-30 行为设置光源位置的代码，光源的位置是通过一个线程来改变的，因此，可以看到光从不同角度照射到场景中的效果。
- 代码第 31-76 行为绘制各个基本几何体的代码，只要将各个基本几何体移到相应位置、旋转一定角度，再调用各自的 drawSelf 方法即可完成几何体的绘制。

11.7 小结

本章深入介绍了 OpenGL ES 应用开发的 3D 基本知识和典型几何体的开发，通过典型案例的讲解，读者应该很容易理解本章所涉及的基本知识。通过本章的学习，读者应具备用 OpenGL ES 开发基本 3D 场景的能力。

第二篇

Android 游戏开发实战综合案例

第 12 章　滚屏动作游戏——太空保卫战

第 13 章　棋牌游戏——中国象棋人机对弈

第 14 章　物理传感器游戏——小球快跑

第 15 章　塔防游戏——精灵塔防

第 16 章　策略游戏——回到战国

第 17 章　体育游戏——2D 迷你桌球

第 18 章　益智游戏——3D 版推箱子

第 19 章　物理引擎游戏——盛怒的老鼠

第 12 章　滚屏动作游戏——太空保卫战

滚屏动作游戏主要考验玩家的反应能力和手眼配合能力，游戏的情景不是重点，而游戏火爆的场面，快速、激烈的游戏节奏，良好的操作以及强烈的震撼感才是最重要的。

太空保卫战便是一款滚屏动作游戏，玩家通过控制角色使用各种武器消灭敌人来过关，其操作简单，易于上手，且紧张激烈的游戏节奏会使玩家产生浓厚的兴趣。

本章将通过“太空保卫战”游戏在 Android 平台上的设计与实现，使读者学会该类游戏的开发，掌握滚屏游戏的开发技巧。

12.1 游戏的背景及功能概述

本节主要对太空保卫战游戏的背景以及功能进行简单介绍，使读者对该游戏有个整体了解，为之后游戏的开发做好准备。

12.1.1　背景概述

滚屏动作类游戏是最传统的游戏类型之一，早期动作游戏的画面一般比较简单，没有复杂的游戏特效，主要是通过各种操作技巧来进行游戏。而现在的动作类游戏则融入了很多新的元素，使其逐渐成为所有游戏类型里面种类最多，发展最快的一种。

太空保卫战便是滚屏动作类游戏的一种，其中融入了很多动作类游戏特有的技术，画面流畅、声音震撼。

12.1.2　功能简介

太空保卫战游戏主要包括欢迎界面、帮助界面，以及游戏界面。接下来对该游戏的部分界面及运行效果进行简单介绍。

（1）运行该游戏，通过 Ctrl+F12 组合键将模拟器调成横屏，进入加载界面，如图 12-1 所示。

（2）之后将进入欢迎动画界面，效果如图 12-2、图 12-3 和图 12-4 所示。

（3）欢迎动画播放完毕将进入选择界面，菜单界面包括“开始游戏”、“帮助”、“关闭声音”以及“退出”按钮，如图 12-5 所示。

（4）单击“帮助”按钮将进入“帮助”界面，如图 12-6 所示。

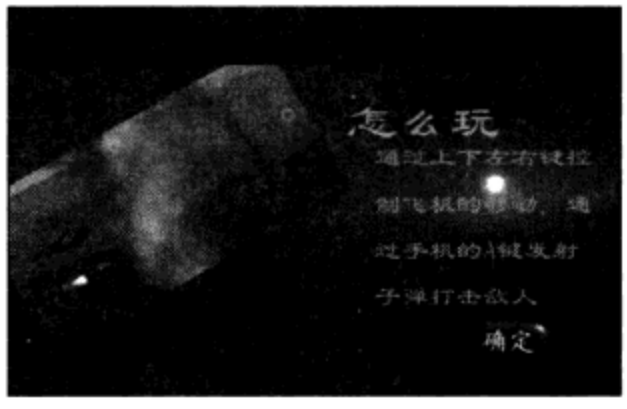
▲图 12-1　加载界面

▲图 12-2　欢迎动画界面 1

▲图 12-3　欢迎动画界面 2

▲图 12-4　欢迎动画界面 3

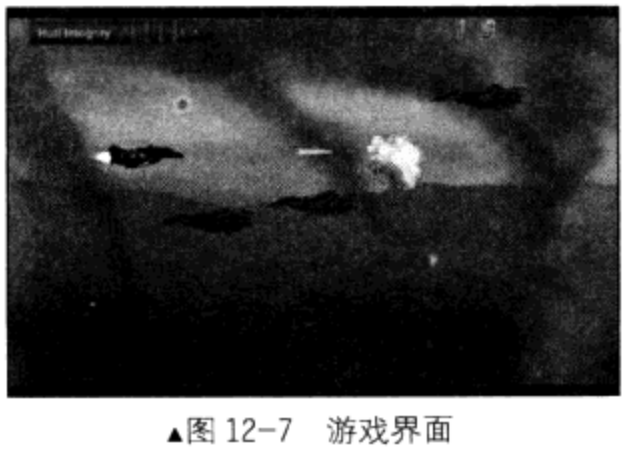
▲图 12-5　选择界面



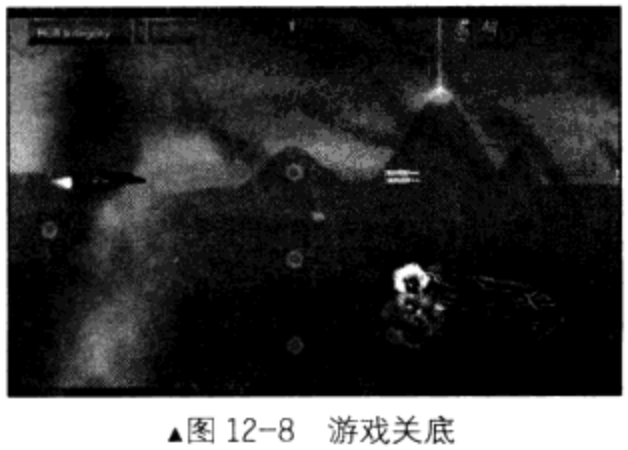
▲图 12-6　帮助界面

（5）单击“开始游戏”按钮将进入游戏界面，在游戏界面，玩家可通过上下左右键控制飞机移动，然后通过 A 键发射子弹打击敌人，如图 12-7 所示。

（6）如果玩家飞机坚持足够的时间没有爆炸的话，将看到关底，如图 12-8 所示。



▲图 12-7　游戏界面



▲图 12-8　游戏关底

（7）当顺利过关时，会出现恭喜过关界面，如图 12-9 所示；而当游戏失败时，则会出现失败界面，如图 12-10 所示。

▲图 12-9　恭喜过关界面

▲图 12-10　游戏失败界面

提示　感性认识对于学习游戏开发是非常重要的，所以请读者在学习本游戏开发步骤前先运行一下光盘中的项目。

12.2 游戏的策划及准备工作

本节主要介绍太空保卫战游戏的策划以及开发前的准备工作，搜集游戏开发过程中用到的资源文件并将其放到指定的位置。

12.2.1　游戏的策划

下面对本游戏的策划进行简单介绍，在真实的游戏开发中其涉及的方面会很多，此处只列出其中的一部分。

- 游戏情节

本游戏为动作类游戏，没有过多的情景设计，所以，主要的工作是主战飞机生命的设计（血值或生命值）、道具的设计（换枪的道具及补血的道具）和关卡的设计等。

- 运行的目标平台

目标平台为 Android 2.1。

- 采用的呈现技术

本案例采用的是 2D 呈现技术，而表现形式采用的是横向滚屏技术，且在贴图时采用多层贴图，增加游戏界面的层次感。

- 操作方式

本游戏拟采用手机键盘进行操作，通过上下左右键控制飞机的移动，A 键发射子弹。

- 音效设计

为了增加玩家的体验，本案例根据界面的效果添加了适当的音效，例如，背景音效、发射炮弹时的音效以及爆炸音效等。

12.2.2　Android 平台下游戏的准备工作

游戏开发前的准备工作是不能省略的，包括制作或搜集所用到的图片、声音等。本案例所用到的图片文件如表 12-1 所示。

表 12-1　　图片清单

图 片 名	大小（KB）	像素（w×h）	用　　途	图 片 名	大小（KB）	像素（w×h）	用　　途
background.jpg	7.52	480×391	欢迎界面背景	background2.jpg	17.2	480×385	欢迎界面背景
battleback.jpg	34.5	2000×300	游戏界面背景	bullet1.png	0.37	30×4	角色的子弹 1
bullet2.png	3.38	15×15	敌人的子弹	bullet3.png	0.48	30×12	角色的子弹 2
bullet4.png	0.5	25×25	角色的子弹 3	bullet5.png	0.53	25×25	角色的子弹 4
closesound.png	4.97	80×30	关闭声音按钮	changebullet.png	1.84	30×25	改变子弹的道具
cloud.png	56.0	160×300	云彩	exit2.png	4.97	90×21	退出游戏按钮 2
exit.png	5.08	80×30	退出游戏按钮 1	explode1～6.png	1.03	50×50	爆炸效果
goon.png	5.01	90×20	重新游戏按钮	fialbackground.jpg	10.5	480×320	游戏失败背景
help2.png	28.9	300×252	帮助界面背景 2	help3.png	6.91	200×200	帮助界面的文字
help.png	4.49	80×30	帮助按钮	hull.png	0.25	7×18	玩家的生命
icon.png	2.51	48×48	游戏的图标	hullbackground.png	4.28	150×33	表示生命处的背景
sound1.png	4.4	200×80	声音开按钮	image1～7.png	5.11	480×216	欢迎动画帧
life.png	1.87	30×25	补生命的道具	number0～9png	3.44	12×17	数字图片
ok.png	2.29	64×32	确定按钮	opensound.png	4.81	80×30	打开声音按钮
plane1～3.png	6.05	100×29	主战飞机	plane4～7.png	3.72	80×23	敌方飞机
progress.png	8.16	300×20	进度条	processbackground.jpg	10.4	516×291	加载界面背景
startgame2.png	2.57	100×44	开始游戏按钮 2	processman.png	24.7	224×150	加载界面效果
startgame.png	4.88	80×30	开始游戏按钮	welcomeborder.png	9.85	500×300	欢迎动画外框

提示　本程序中同样需要将所有的图片资源都存储在 res\drawable-mdpi 文件夹下。

本游戏用到的声音文件如表 12-2 所示。

表 12-2　　声音清单

声音文件名	大小（KB）	格式	用　　途	声音文件名	大小（KB）	格式	用　　途
bulletsound1	6.65	ogg	飞机打子弹的声音	dead	31.7	ogg	游戏失败的声音
explode	5.68	ogg	爆炸效果音乐	gamestart	66.8	ogg	欢迎动画背景音乐
welcome1	91.4	ogg	欢迎动画背景音乐 2				

说明	同样将上述声音文件放到该游戏项目中 res/raw 文件夹下，.ogg 格式更适合在游戏中使用，所以建议读者将搜集到的声音文件通过转换器转换成.ogg 格式再使用。

12.3 游戏的架构

本节将对太空保卫战游戏的架构进行介绍，通过本节的学习，可以使读者对该游戏的设计思路以及整体架构有一定的了解。

12.3.1 各个类的简要介绍

首先对该项目中各个类的功能进行简单介绍，为了让读者更加清晰地理解，下面将这些类分成 4 部分进行介绍。

1. 共有类

- Activity 的实现类 PlaneActivity

该类是通过扩展基类 Activity 实现的，是整个游戏程序的入口。

- 键盘监听线程 KeyThread

该类为手机键盘的监听线程，每隔一段时间读取一次键盘的状态值，并根据当前键盘状态做出相应的操作。

- 常量类 ConstantUtil

将游戏中用到的所有常量全部封装到该类中，便于管理与维护。

2. 辅助界面相关类

- 欢迎界面 WelcomeView

该类为游戏欢迎界面的实现类，主要负责欢迎界面的绘制工作。

- 欢迎界面的动画实现类 WelcomeViewThread

该类是为 WelcomeView 服务的线程类，主要负责欢迎界面中动画的生成。

- 帮助界面 HelpView

该类非常简单，主要负责绘制帮助界面的相关图片。

- 加载界面 ProcessView

为了使游戏更加人性化，给玩家更好的体验，在长时间加载某些资源时，需要给出加载界面以提示玩家后台加载的进度。

- 胜利界面 WinView

该类为玩家顺利过关后的界面。

- 失败界面 FailView

该类为主战飞机死亡后显示的失败界面，在该界面中玩家可以选择重新玩游戏或退出游戏。

3. 游戏界面相关类

- 游戏主界面类 GameView

该类为游戏菜单界面的实现类，负责界面绘制工作。

- 背景滚动线程类 GameViewBackGroundThread

该类主要负责背景滚动、敌机以及各种道具的出现。

- 地图信息类 Maps

该类包含着地图的相关信息，例如，敌机所有的移动路线、各个关卡中补血道具、换枪道具等。

- 物体移动线程类 MoveThread

该线程负责除了我方主战飞机外所有可移动物的移动。

4. 情景相关类

- 主战飞机 Plane

该类为主战飞机的封装类，可由玩家控制其方向及移动状态。

- 敌机 EnemyPlane

该类为敌机的封装类。

- 子弹类 Bullet

子弹的封装类，游戏中所有的子弹都为该类的对象。

- 爆炸效果类 Explode

该类为爆炸效果的渲染类，负责在指定位置绘制爆炸效果。

- 爆炸换帧线程 ExplodeThread

该类是为爆炸效果类服务的，当爆炸存在时，定时为每个爆炸效果换帧。

- 补血道具类 Life

该类为补血道具类，当主战飞机与该类的对象碰撞时，会为主战飞机增加一格血。

- 换枪道具类 ChangeBullet

该类为换枪道具类，当主战飞机与该类的对象碰撞时，会增强主战飞机所携带武器。

12.3.2　游戏的框架简介

本小节将对该游戏的框架进行介绍，使读者从整体上对该游戏有一定的了解，其框架如图 12-11 所示。

接下来将通过游戏的运行过程介绍各个类在游戏中的作用，具体如下。

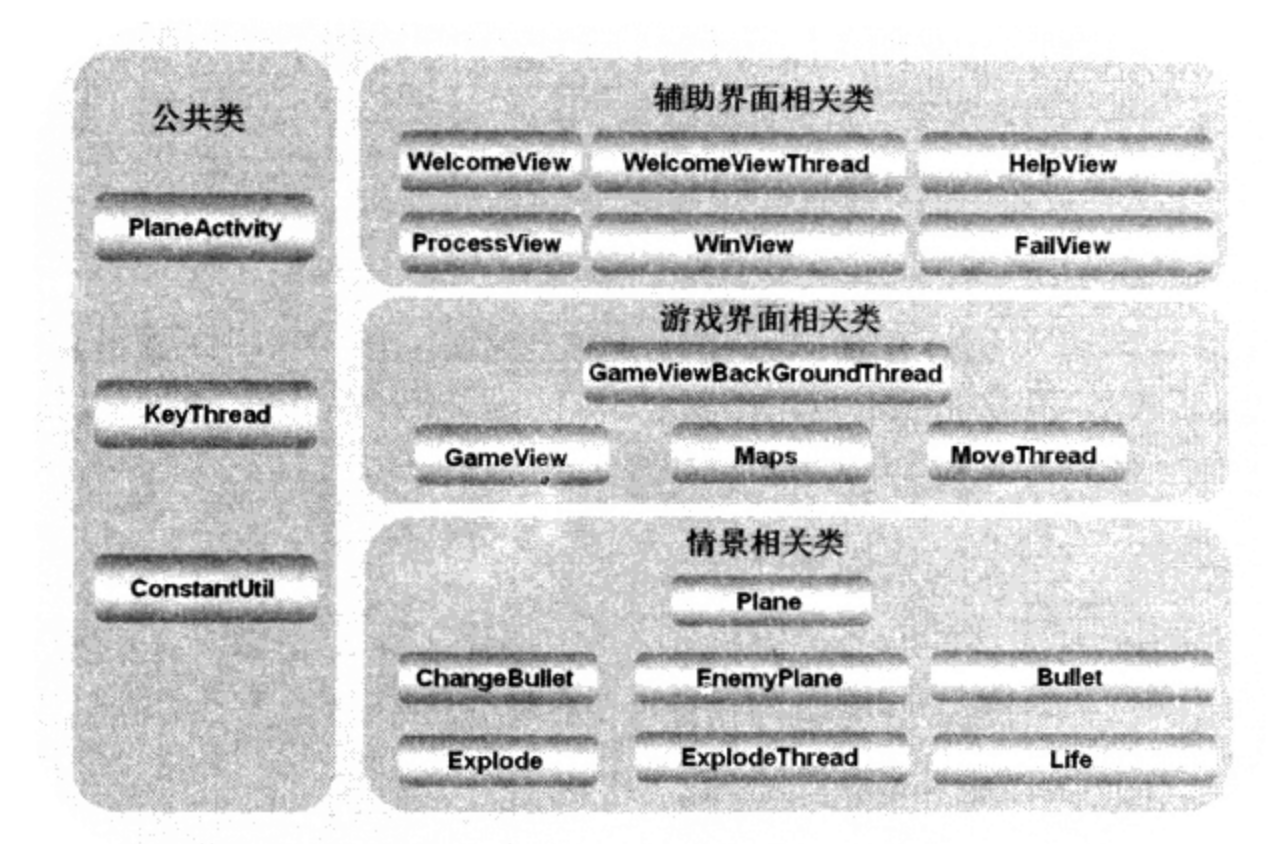
（1）启动游戏，在 PlaneActivity 中会先进入加载界面 ProcessView，然后后台加载欢迎界面 WelcomeView，当加载完成后将屏幕切换到欢迎界面，并启动 WelcomeViewThread 线程生成欢迎界面的动画效果。

（2）当欢迎动画播放完毕后，屏幕会停留在选择状态等待玩家操作。

（3）当玩家单击“帮助”按钮时，将初始化帮助界面 HelpView 并将其显示出来。

（4）当玩家单击“开始游戏”按钮时，屏幕将显示加载界面。后台将初始化游戏界面 GameView，

在地图信息类 Maps 中初始化此关卡中的所有敌机、变枪的道具、补血的道具等。



▲图 12-11　太空保卫战游戏类框架图

（5）初始化游戏界面的同时启动 GameViewBackGroundThread 线程，以滚动游戏界面的背景图片。

（6）当玩家发射炮弹或敌机发射炮弹时会创建炮弹类 Bullet 的对象，并将其添加到已知的容器中。

（7）在运行游戏过程中如果需要显示爆炸效果，会创建爆炸效果类 Explode 及 ExplodeThread 类来渲染爆炸效果。

（8）游戏中所有类中使用的常量全部封装到常量类 ConstantUtil 中。

12.4 辅助界面相关类的实现

接下来将正式进入游戏的开发，为了让读者了解游戏的开发过程，本节仍然按照实际开发的步骤进行介绍，当然，首先应该从欢迎界面的开发开始。

12.4.1　主控制类——PlaneActivity 的实现

在开发欢迎界面之前，应该先介绍 Activity 的实现，因为在游戏的运行过程中随时需要 PlaneActivity 的管理，其框架代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 PlaneActivity.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    ……//该处省略了部分包的引入，读者可自行查看随书光盘源代码的该处
```

```
4    import android.view.WindowManager;                          //引入相关类
5    public class PlaneActivity extends Activity{
6         int action = 0;                                        //键盘的状态,二进制表示从左
                                                                 //往右表示上下左右
7         GameView gameView;                                     //GameView 的引用
8         WelcomeView welcomeView;                               //WelcomeView 的引用
9         FailView failView;                                     //游戏失败界面的引用
10        HelpView helpView;                                     //HelpView 的引用
11        WinView winView;                                       //欢迎界面的引用
12        ProcessView processView;                               //进度条界面的引用
13        boolean isSound = true;                                //是否播放声音
14        Handler myHandler = new Handler(){                     //用来更新 UI 线程中的控件
15             public void handleMessage(Message msg) {
16             ……//该处省略了根据发来的消息做相应操作的相关代码，读者可自行查看随书光盘的源代码
17             }
18        };
19        public void onCreate(Bundle savedInstanceState) {      //Activity 被创建被调用
20             super.onCreate(savedInstanceState);
21             //全屏
22             requestWindowFeature(Window.FEATURE_NO_TITLE);
23             getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN ,
24                        WindowManager.LayoutParams.FLAG_FULLSCREEN);
25             processView = new ProcessView(this,1);             //初始化进度条并切换到进度
                                                                 //条 View
26             this.setContentView(processView);                  //设置加载界面
27             new Thread(){                                      //线程
28                  public void run(){
29                       Looper.prepare();
30                       welcomeView = new WelcomeView(PlaneActivity.this);
                                                                 //初始化 WelcomeView
31                  }
32             }.start();                                         //启动线程
33        }
34        public void toWelcomeView(){                           //切换到欢迎界面
35             this.setContentView(welcomeView);
36        }
37        public void toGameView(){                              //切换到游戏界面
38             this.setContentView(gameView);
39        }
40        public void initHelpView(){                            //初始帮助界面
41             ……//该处省略了帮助界面的初始化工作，读者可自行查看随书光盘的源代码
42        }
43        public void initFailView(){                            //初始游戏失败界面
44             ……//该处省略了失败界面的初始化工作，读者可自行查看随书光盘的源代码
45        }
46        public void initWinView(){                             //初始胜利界面
47             ……//该处省略了胜利界面的初始化工作，读者可自行查看随书光盘的源代码
48        }
49        public boolean onKeyUp(int keyCode, KeyEvent event) {        //键盘抬起
50             ……//该处省略了键盘按键抬起的事件响应代码，读者可自行查看随书光盘的源代码
51        }
52        public boolean onKeyDown(int keyCode, KeyEvent event){       //键盘按下监听
53             ……//该处省略了键盘按键按下的事件响应代码，读者可自行查看随书光盘的源代码
54        }}
```

- 第 6 行用一个整形变量表示当前键盘的状态，计算时使用二进制计算，用后 6 位表示各个键的当前状态，从左往右各位依次表示上下左右及 A、B 按键是否被按下，其中 A 键表示发射子弹，B 键为备用键，为以后游戏的扩展提供方便。
- 第 7～12 行声明各个界面的引用，第 16 行省略了根据收到的消息调用不同的方法进行切换

屏幕的代码，其实现与前面章节介绍过的完全相同，此处不再赘述。

- 第 14～18 行初始化用于接收各个界面发送消息的 Handler。
- 第 19～33 行重写 onCreate 方法，在方法中先将手机设置成全屏模式，然后切换到加载界面并在后台初始化欢迎动画界面。
- 第 34～48 行为各个界面的初始化方法，此处只给出这些方法的空实现，其详细代码将在后面相继给出。
- 第 49～54 行重写了手机键盘的监听方法，其作用是根据当前键盘的按键状态改变成员变量 action 的值，其实现与前面章节的内容很相似，读者有需要可以自行查阅光盘中的相关代码。

12.4.2 欢迎界面 WelcomeView 类

接下来正式进入欢迎动画界面的介绍。

1. 欢迎界面的框架介绍

首先对欢迎界面 WelcomeView 的框架进行介绍，其代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 WelcomeView.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    import java.util.HashMap;                                           //引入相关类
3    ……//该处省略了部分包的引入，读者可自行查看随书光盘源代码的该处
4    import android.view.SurfaceView;                                    //引入相关类
5    public class WelcomeView extends SurfaceView implements SurfaceHolder.Callback {
6         PlaneActivity activity;                                        //Activity 的引用
7         private TutorialThread thread;                                 //刷帧的线程
8         private WelcomeViewThread welcomeThread;                       //绘制线程
9         int status = 1;                                                //当前的状态值
10        int k = 0;                                                     //状态为 2 时用到的切换图片索引
11        int alpha = 255;                                               //透明度
12        SoundPool soundPool;                                           //声音
13        HashMap<Integer, Integer> soundPoolMap;
14        Bitmap welcomeborder;                                          //背景外框图片
15        Bitmap background;                                             //背景图片
16        Bitmap background2;                                                //背景图片 2
17        Bitmap image1,image2,image3,image4,image5,image6,image7;           //动画帧
18        Bitmap startGame;                                                  //开始游戏菜单
19        Bitmap help;                                                       //帮助菜单
20        Bitmap openSound;                                                  //打开声音菜单
21        Bitmap closeSound;                                                 //关闭声音菜单
22        Bitmap exit;                                                       //退出菜单
23        int backgroundY = -200;                                            //背景的坐标
24        int background2Y = 40;
25        Paint paint;                                                       //用于改变透明度
26        Paint paint2;                                                      //正常绘制
27        public WelcomeView(PlaneActivity activity) {                       //构造器
28             super(activity);
29             this.activity = activity;                                     //得到 Activity 的引用
30             if(activity.processView != null){                             //加载界面的进度条
31                  activity.processView.process += 10;
32             }
33             getHolder().addCallback(this);
34        this.thread = new TutorialThread(getHolder(), this);               //初始化刷帧线程
35             this.welcomeThread = new WelcomeViewThread(this);             //初始化动画线程
36        initSounds();                                                      //初始化声音
```

```
37              initBitmap();                                          //初始化图片资源
38              playSound(1);                                          //播放声音
39          }
40          public void initSounds(){                                  //初始化声音的方法
41              ……//该处省略了声音初始化的代码，将在之后给出
42          }
43          public void playSound(int sound) {                         //播放声音的方法
44              ……//该处省略了播放的代码，将在之后给出
45          }
46          public void initBitmap(){                                  //初始化图片资源的方法
47              ……//该处省略了初始化图片资源的代码，读者可自行查看随书光盘的源代码
48          }
49          public void onDraw(Canvas canvas){                         //自己写的绘制方法
50              ……//该处省略了屏幕绘制的代码，将在之后给出
51          }
52          public void surfaceChanged(SurfaceHolder holder, int format, int width, int
            height) {
53          }
54          public void surfaceCreated(SurfaceHolder holder) {         //创建时被调用
55              ……//该处省略了绘制线程和动画线程的初始化和启动工作，读者可自行查看随书光盘的源代码
56          }
57          public void surfaceDestroyed(SurfaceHolder holder) {       //摧毁时被调用
58              ……//该处省略了停止绘制线程的代码，读者可自行查看随书光盘的源代码
59          }
60          public boolean onTouchEvent(MotionEvent event) {           //屏幕监听
61              ……//该处省略了屏幕的监听方法，读者可自行查看随书光盘的源代码
62          }
63          class TutorialThread extends Thread{                       //刷帧线程
64              private int span = 100;                                //睡眠的毫秒数
65              private SurfaceHolder surfaceHolder;
66              private WelcomeView welcomeView;                   //欢迎界面的引用
67              private boolean flag = false;                      //循环标记
68              public TutorialThread(SurfaceHolder surfaceHolder, WelcomeView welcomeView) {
                                                                   //构造器
69                  this.surfaceHolder = surfaceHolder;            //SurfaceHolder 的引用
70                  this.welcomeView = welcomeView;                //欢迎界面的引用
71              }
72              public void setFlag(boolean flag) {                //设置标志位
73                  this.flag = flag;
74              }
75              public void run() {                                //重写 run 方法
76              ……//该处省略了定时调用屏幕的绘制方法的代码，与前面章节中的相同，读者可自行查看源代码
77              }}}
```

- 第 6～26 为成员变量的声明，各个变量的含义如注释所示。
- 第 27～38 行为该类的构造器，第 30～32 行将加载界面中表示进度的值加 10，第 34～35 行初始化刷帧线程以及动画线程，第 36～38 行则初始化声音及图片资源并播放欢迎界面的背景音乐。
- 第 40～59 行为各个方法的空实现，方法的作用如注释所示，各个方法的代码将在之后给出。
- 第 60～62 行的方法为屏幕的监听方法，在该方法中根据单击位置的坐标判断出玩家单击的是什么菜单，并根据单击的菜单向 Activity 的 Handler 发送消息，其代码比较简单，与之前例子的屏幕监听方法基本相同。
- 第 33～77 行为刷帧线程，在该类的 run 方法中每隔一段时间便调用一次 WelcomeView 中的 onDraw 方法绘制界面。

2. 欢迎界面声音相关方法的完善

（1）首先对初始化声音的方法进行完善，本游戏中使用 SoundPool 类来播放声音，其初始化的代码如下。

```
1    public void initSounds(){                                    //初始化声音的方法
2         soundPool = new SoundPool(4, AudioManager.STREAM_MUSIC, 100);
                                                                   //初始化 SoundPool
3         soundPoolMap = new HashMap<Integer, Integer>();          //初始化 HashMap
4         soundPoolMap.put(1, soundPool.load(getContext(), R.raw.welcome1, 1));
                                                                   //初始化第一个声音
5    }
```

说明 用该方法替换欢迎界面框架中第 41 行的方法，在该方法中先初始化 SoundPool，然后创建一个 HashMap 来协助管理需要播放的声音。

（2）对播放声音的方法进行完善，用下列代码替换游戏界面框架中第 44 行的方法。

```
1    public void playSound(int sound) {                           //播放声音的方法
2         AudioManager mgr = (AudioManager)getContext().getSystemService(Context.AUDIO_SERVICE);
3         float streamVolumeCurrent = mgr.getStreamVolume(AudioManager.STREAM_MUSIC);
4         float streamVolumeMax = mgr.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
                                                                   //最大音量
5         float volume = streamVolumeCurrent/streamVolumeMax;      //设备的音量
6         soundPool.play(soundPoolMap.get(sound), volume, volume, 1, 0, 1f); //播放
7    }
```

说明 该方法为播放声音的方法，首先对播放声音的设备和声音的大小进行设置，然后根据需要播放声音的 ID 播放声音。

3. 绘制方法 onDraw 的完善

接下来对欢迎界面的绘制方法进行介绍，同样用下列代码替换框架中第 50 行的方法。

```
1    public void onDraw(Canvas canvas){                           //自己写的绘制方法
2         //画的内容是 z 轴的，后画的会覆盖前面画的
3         canvas.drawColor(Color.WHITE);                          //背景色
4         if(this.status == 1 || this.status == 2){
5              canvas.drawBitmap(background, 0, backgroundY, paint);//绘制背景
6              if(k == 0){                                        //k 值不同时的绘制
7                   canvas.drawBitmap(image1, 0, 48, paint);      //绘制图片
8              }else if(k > 0 && k<=1){                           //k 值不同时的绘制
9                   canvas.drawBitmap(image2, 0, 48, paint);
10             }
11             ……//该处省略了 k 的不同值的绘制方法
12             }else if(k>8 && k<=10){                            //k 值不同时的绘制
13                  canvas.drawBitmap(image2, 0, 48, paint);      //绘制图片
14             }
15             canvas.drawRect(0, 0, 480, 48, paint);             //绘制上下的黑框
16             canvas.drawRect(0, 270, 480, 320, paint);          //绘制矩形
17             canvas.drawBitmap(welcomeborder, -14, 10, paint);  //绘制外框
18        }
19        else if(status == 3){
20             paint.setAlpha(alpha);                             //设置透明度
21             canvas.drawBitmap(background2, 0, background2Y, paint);
```

```
                                                                    //绘制背景图
22              canvas.drawRect(0, 0, 480, 48, paint2);             //绘制上下的黑框
23              canvas.drawRect(0, 270, 480, 320, paint2);          //绘制矩形框
24          }
25          else if(status == 4){                                   //菜单状态
26              canvas.drawBitmap(background2, 0, background2Y, paint);
                                                                    //绘制背景图
27              canvas.drawBitmap(startGame, 10, 70, paint2);       //绘制开始游戏按钮
28              canvas.drawBitmap(help, 390, 60, paint2);           //绘制帮助按钮
29              canvas.drawBitmap(exit, 380, 230, paint2);          //绘制退出按钮
30              if(activity.isSound){
31                  canvas.drawBitmap(closeSound, 10, 230, paint2); //绘制关闭声音菜单
32              }else{
33                  canvas.drawBitmap(openSound, 10, 230, paint2);//绘制打开声音
34              }
35              canvas.drawRect(0, 0, 480, 48, paint2);             //绘制上下的黑框
36              canvas.drawRect(0, 270, 480, 320, paint2);
37          }}
```

说明　该方法只是根据当前状态在指定位置绘制相应的图片，在绘制前，可通过对画笔的设置调整所绘图片的透明度。

4. 动画线程 WelcomeViewThread 类的介绍

WelcomeViewThread 线程是用来生成欢迎界面动画。在该类的 run 方法中，会根据 WelcomeView 中的状态值改变图片的坐标或者更改需要绘制的图片帧，其 run 方法代码如下。

```
1     public void run(){                                            //重写 run 方法
2         while(flag){                                              //循环
3             if(welcomeView.status == 1){                          //当状态为 1 时
4                 welcomeView.backgroundY += 8;
5                 if(welcomeView.backgroundY>80){                   //当背景图片到位后
6                     welcomeView.status = 2;                       //改变状态值
7                 }
8             }else if(welcomeView.status == 2){                    //当状态为 2 时
9                 welcomeView.k++;                                  //更换帧
10                if(welcomeView.k == 11){
11                    welcomeView.status = 3;                       //改变状态
12                }
13            }else if(welcomeView.status == 3){                    //状态值为 3 时
14                welcomeView.background2Y -= 2;                    //移动图片
15                welcomeView.alpha -= 2;                           //改变透明度
16                if(welcomeView.alpha < 125){                      //当透明度太低时
17                    welcomeView.alpha = 125;                      //将透明度设为 125
18                }
19                if(welcomeView.background2Y < -90){               //当背景 2 图片到位后
20                    welcomeView.status = 4;                       //更改状态
21                }
22            }
23            try{
24                Thread.sleep(span);                               //睡眠制定毫秒数
25            }catch(Exception e){                                  //捕获异常
26                e.printStackTrace();                              //打印异常信息
27            }}}
```

- 第 3～7 行用于移动最后面的大背景图片。
- 第 8～12 行当状态位为 2 时更改显示图片的帧，达到动画的效果。

- 第 13～22 行当状态为 3 时，移动图片的同时改变图片的透明度，当透明度过低时将透明度设置成半透明。

12.4.3 其他辅助界面的介绍

前面已经对欢迎界面进行了介绍，下面对该游戏的一些其他辅助界面进行简单介绍。

1. 加载界面 ProcessView 类

加载界面在游戏中会用到多次，当需要长时间加载某些资源时，会先将屏幕切到加载界面，将耗时的加载工作放到后台的线程中完成，加载界面的代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 ProcessView.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.graphics.Bitmap;                           //引入相关类
3    ……//该处省略了部分包的引入，读者可自行查看随书光盘源代码的该处
4    import android.view.SurfaceView;                          //引入相关类
5    public class ProcessView extends SurfaceView implements SurfaceHolder.Callback {
6         PlaneActivity activity;                              //Activity 的引用
7         private TutorialThread thread;                       //刷帧的线程
8         Paint paint;                                         //画笔
9         Bitmap processBitmap;                                //加载
10        Bitmap processBackground;                            //背景图片
11        Bitmap processMan;                                   //人的图片
12        int process = 0;                                     //用 0～100 表示后台的加载进度
13        int startX = 90;
14        int startY = 150;
15        int type;                                            //当前加载的是哪个 View
16        int k = 0;                                           //加载中的点
17        public ProcessView(PlaneActivity activity, int type) {   //构造器
18             ……//该处省略了构造器中的部分代码，读者可自行查看随书光盘的源代码
19        }
20        public void initBitmap(){                            //初始化图片资源的方法
21             ……//该处省略了图片资源的初始化代码，读者可自行查看随书光盘的源代码
22        }
23        public void onDraw(Canvas canvas){                   //自己写的绘制方法
24             //画的内容是 z 轴的，后画的会覆盖前面画的
25             canvas.drawBitmap(processBackground, -10, 10, paint);   //绘制背景图片
26             canvas.drawBitmap(processMan, 10, 130, paint);          //绘制人的图片
27             canvas.drawBitmap(processBitmap, startX, startY, paint);//绘制进度条
28             canvas.drawRect(startX+process*(processBitmap.getWidth()/100), startY,
29                  startX+processBitmap.getWidth(), startY+processBitmap.getHeight(), paint);
30             if(k == 0){                                     //当 k 等于 0 时
31                  canvas.drawText("加载中", startX+(processBitmap.getWidth()/2)-20,
32                            startY+processBitmap.getHeight()+20, paint);
                                                               //绘制加载中 3 个字
33             ……//该处省略了 k 等于其他值的代码，读者可自行查看随书光盘的源代码
34             }else{
35                  canvas.drawText("加载中...", startX+(processBitmap.getWidth()/2)-20,
36                            startY+processBitmap.getHeight()+20, paint);
37             }
38             canvas.drawRect(0, 0, 480, 48, paint);          //绘制上下的黑框
39             canvas.drawRect(0, 270, 480, 320, paint);       //绘制外框
40             k = (k+1)%4;                                    //k 循环自加
41        }
42        public void surfaceChanged(SurfaceHolder holder, int format, int width, int height)
          {}
```

```
43          public void surfaceCreated(SurfaceHolder holder) {        //创建时被调用
44               ……//该处省略了绘制线程启动的代码，读者可自行查看随书光盘的源代码
45          }
46          public void surfaceDestroyed(SurfaceHolder holder) {      //摧毁时被调用
47               ……//该处省略了释放绘制线程的代码，读者可自行查看随书光盘的源代码
48          }
49          class TutorialThread extends Thread{                      //刷帧线程
50               private int span = 400;                              //睡眠的毫秒数
51               private SurfaceHolder surfaceHolder;
52               private ProcessView processView;                     //processView 引用
53               private boolean flag = false;                        //循环标志位
54                    public TutorialThread(SurfaceHolder surfaceHolder, ProcessView processView) {
                                                                      //构造器
55                         this.surfaceHolder = surfaceHolder;
56                         this.processView = processView;            //得到加载界面
57                    }
58                    public void setFlag(boolean flag) {             //设置标志位
59                         this.flag = flag;
60                    }
61               public void run() {                                  //重写 run 方法
62          ……//该处省略了定时调用加载界面的 onDraw 绘制方法的代码，读者可自行查看随书光盘的源代码
63                  if(processView.process >= 100){                   //当加载完成时
64                         if(processView.type == 1){                 //切换到 WelcomeView
65                  processView.activity.myHandler.sendEmptyMessage(4);
                                                  //向主 Activity 发送 Handler 消息
66                         }else if(processView.type == 2){//切到 GameView
67                  processView.activity.myHandler.sendEmptyMessage(6);
                                                  //向主 Activity 发送 Handler 消息
68                         }
69                     }
70                  try{
71                         Thread.sleep(span);                        //睡眠指定毫秒数
72                  }catch(Exception e){                              //捕获异常信息
73                         e.printStackTrace();                       //打印异常信息
74                  }}}}
```

- 第 6～16 行对该类中所用到的成员变量进行声明。
- 第 17～19 行为该类的构造方法，在方法中初始化图片以及重绘线程。
- 第 23～40 行为重写的 onDraw 绘制方法，负责在指定位置上绘制需要的图片，第 30～37 行绘制的是“加载中”3 个字，根据 k 值的不同“加载中”后面点的个数不一样。
- 第 42～48 行为实现接口中的方法，负责在 View 被创建和被摧毁时启动和停止重绘线程，其实现与前面章节介绍过的相同。
- 第 62 行省略的是定时刷新加载界面的代码，其实现与前面章节介绍的刷帧线程的完全相同，读者可参考前面章节的例子或查看随书光盘中的源代码。
- 第 63～69 行表示当加载的进度到达 100 时，根据 type 的值向 Activity 发送 Handler 消息，通知 PlaneActivity 资源已经加载完成，请切换用户界面。

2. 帮助界面 HelpView 类

接下来介绍另一个辅助界面——帮助界面，该界面非常简单，首先绘制一些背景效果图，然后在指定的位置绘制一个“确定”按钮，并捕捉“确定”按钮的事件响应来向 Activity 发送 Handler 消息，其代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 HelpView.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    import android.graphics.Bitmap;                                     //引入相关类
3    ……//该处省略了部分包的引入，读者可自行查看随书光盘源代码的该处
4    import android.view.SurfaceView;                                    //引入相关类
5    public class HelpView extends SurfaceView implements SurfaceHolder.Callback {
6         PlaneActivity activity;
7         private TutorialThread thread;                                 //刷帧的线程
8         Paint paint;                                                   //画笔
9         Bitmap background;                                             //背景图片
10        Bitmap help2;                                                  //帮助图片
11        Bitmap help3;
12        Bitmap ok;                                                     //确定按钮
13        public HelpView(PlaneActivity activity) {                      //构造器
14             ……//该处省略了构造器中的代码，读者可自行查看随书光盘源代码的该处
15        }
16        public void initBitmap(){                                      //初始化图片资源的方法
17             ……//该处省略了图片资源的初始化代码，读者可自行查看随书光盘源代码
18        }
19        public void onDraw(Canvas canvas){                             //自己写的绘制方法
20             ……//该处省略了图片绘制的代码，读者可自行查看随书光盘源代码的该处
21        }
22        public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
23        }
24        public void surfaceCreated(SurfaceHolder holder) {             //创建时被调用
25             ……//该处省略了重绘线程启动的代码，读者可自行查看随书光盘源代码的该处
26        }
27        public void surfaceDestroyed(SurfaceHolder holder) {           //摧毁时被调用
28             ……//该处省略了重绘线程释放的代码，读者可自行查看随书光盘源代码的该处
29        }
30        public boolean onTouchEvent(MotionEvent event) {               //屏幕监听
31             if(event.getAction() == MotionEvent.ACTION_DOWN){
32                  if(event.getX()>360 && event.getX()<360+ok.getWidth()
33                            && event.getY()>250 && event.getY()<250+ok.getHeight()){
                                                                          //单击确定按钮
34                       Message msg1 = activity.myHandler.obtainMessage(7);
35                       activity.myHandler.sendMessage(msg1);           //发送 Handler 消息
36                  }
37             }
38             return super.onTouchEvent(event);                      //返回调用基类的方法
39        }
40        class TutorialThread extends Thread{                       //刷帧线程
41             ……//该处省略了刷帧线程的全部代码，读者可自行查看随书光盘源代码
42        }
43   }
```

说明　该类中的代码与之前介绍过的欢迎界面 WelcomeView 的实现基本相同，在此就不再赘述，读者可参照欢迎界面的介绍进行学习。

3. 其他界面

除了之前介绍过的欢迎界面、加载界面以及帮助界面外，还有胜利界面、失败界面等，其运行效果如图 12-12 和图 12-13 所示。

提示　这两个界面的开发思路与其他界面完全相同，都是先绘制一个背景图片，然后绘制两个按钮并对按钮进行屏幕监听。因本书篇幅有限，并没有给出胜利以及失败界面的代码，读者可自行查阅随书光盘中本案例的源代码。

▲图 12-12　胜利界面

▲图 12-13　失败界面

12.5 游戏界面的框架设计

前面已经介绍了辅助界面的实现，接下来将进入本游戏开发的最主要环节——游戏界面的开发，需要先对游戏界面的整体框架进行介绍。

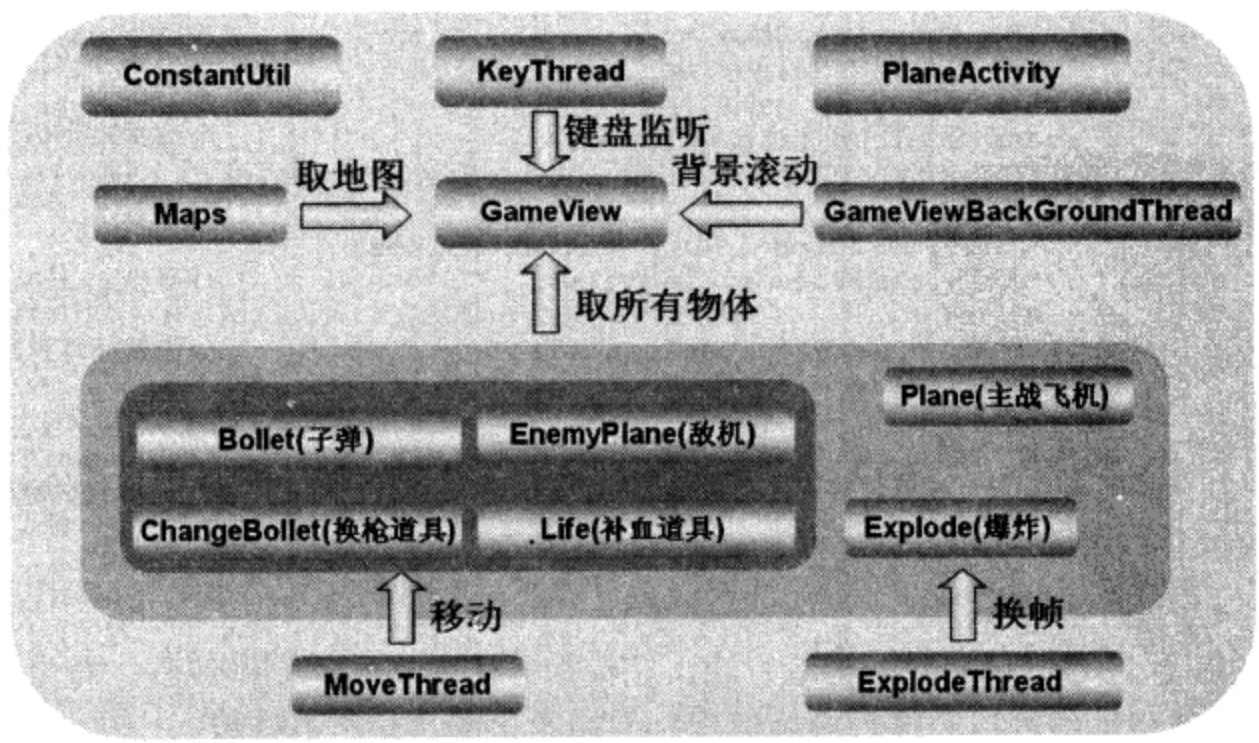
游戏界面的框架如图 12-14 所示，下面根据该框架图，对游戏界面的运行思路进行介绍。

（1）GameView 在初始化时会从 Maps 读取当前关卡的地图信息，包括敌机、补血道具、换枪道具等。

（2）在界面初始化的同时，还需要启动 MoveThread 线程、GameViewBackGroundThread 线程、KeyThread 线程以及 ExplodeThread 线程。

（3）GameView 会在适当的时候从 Plane 类读取主战飞机的信息并进行绘制。

（4）当主战飞机发射子弹时，会创建 Bullet 对象并添加到指定容器中，此时 MoveThread 同样需要定时改变子弹对象的位置。



▲图 12-14　游戏界面基本框架

（5）在以上过程中，全部需要 PlaneActivity 的管理与 ConstantUtil 的参与。

接下来将对 GameView 的框架进行介绍，步骤如下。

（1）首先需要搭建 GameView 类的框架，其代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 GameView.java。

```
1    package wyf.ytl;
2    import java.util.ArrayList;                                    //引入相关类
3    ……//该处省略了部分包的引入，读者可自行查看随书光盘源代码的该处
4    import android.view.SurfaceView;                               //引入相关类
5    public class GameView extends SurfaceView implements SurfaceHolder.Callback{
6         ……//该处省略了成员变量的声明，将在后面给出
7         public GameView(PlaneActivity activity) {                 //构造器
8              ……//该处省略了游戏界面的初始化代码，读者可自行查看随书光盘源代码的该处
9         }
10        public void initSounds(){                                 //初始化声音的方法
11             ……//该处省略了声音资源的初始化代码，读者可自行查看随书光盘源代码的该处
12        }
13        public void playSound(int sound, int loop) {              //播放声音的方法
14             ……//该处省略了播放声音的代码，读者可自行查看随书光盘源代码的该处
15        }
16        public void initBitmap(){                                 //初始化所有图片
17             ……//该处省略了图片资源的初始化代码，将在后面给出
18        }
19        public void onDraw(Canvas canvas){                        //绘制方法
20             ……//该处省略了游戏界面的绘制代码，将在后面给出
21        }
22        public void surfaceChanged(SurfaceHolder holder, int format, int width, int height)
{
23        }
24        public void surfaceCreated(SurfaceHolder holder) {   //创建时启动相应进程
25             ……//该处省略了各个线程的启动工作的代码，读者可自行查看随书光盘源代码的该处
26        }
27        public void surfaceDestroyed(SurfaceHolder holder) {//摧毁时释放相应进程
28             ……//该处省略了各个线程的释放工作的代码，读者可自行查看随书光盘源代码的该处
29        }
30        class TutorialThread extends Thread{                      //刷帧线程
31             private int sleepSpan = 67;                          //睡眠的毫秒数
32             private SurfaceHolder surfaceHolder;                 // surfaceHolder 的引用
33             private GameView gameView;                           //gameView 的引用
34             private boolean flag = false;                        //循环标志位
35             public TutorialThread(SurfaceHolder surfaceHolder, GameView gameView) {
                                                                    //构造器
36                  this.surfaceHolder = surfaceHolder;
37                  this.gameView = gameView;                       //得到 gameView 的引用
38             }
39             public void setFlag(boolean flag) {                  //设置循环标记位
40                  this.flag = flag;
41             }
42             public void run() {                                  //重写 run 方法
43                  ……//该处省略了线程的刷帧代码，读者可自行查看随书光盘源代码的该处
44             }
45        }
46   }
```

- 第 7～13 行为 GameView 的构造方法，在构造方法中初始化刷帧线程、背景滚动线程、键盘监听线程、物体移动线程以及爆炸换帧线程，并初始化图片与声音资源。
- 第 10～12 行初始化声音资源，第 16～18 行初始化图片资源。

- 第 19～21 行为重写的界面绘制方法，该方法负责绘制游戏界面的所有信息，包括主战飞机、敌机、各种道具等。
- 第 22～29 行实现接口中的 3 个方法，负责各个线程的启动与释放工作。
- 第 33～45 行为刷帧线程，负责定时调用游戏界面 GameView 的绘制方法重新绘制屏幕。

（2）接下来给出 GameView 类的成员变量，将下列代码添加到前面代码的第 6 行处。

```
1          int selectMap = 1;                                          //第几关
2          PlaneActivity activity;                                     //Activity 的引用
3          private TutorialThread thread;                              //刷帧的线程
4          GameViewBackGroundThread gameThread;                        //背景滚动线程
5          KeyThread keyThread;                                        //键盘监听线程
6          MoveThread moveThread;                                      //移动物体的线程
7          ExplodeThread explodeThread;                                //爆炸换帧的线程
8          int backGroundIX = 0;                                       //核心图的 x 坐标
9          int i = 0;                                                  //核心图的索引
10         int cloudX = 470;                                           //云彩的 X 坐标
11         Bitmap battleback;                                          //背景的大图元
12         Bitmap[] battlebacks = new Bitmap[ConstantUtil.pictureCount];
                                                                       //装分割以后的图片
13         Bitmap cloud;                                               //云彩
14         Bitmap enemyPlane1;                                         //敌机 1
15         Bitmap enemyPlane2;                                         //敌机 2
16         Bitmap enemyPlane3;                                         //敌机 3
17         Bitmap enemyPlane4;                                         //敌机 4
18         Bitmap[] number = new Bitmap[10];                           //数字数组
19         int[] explodesID = new int[]{                               //爆炸效果的所有帧
20              R.drawable.explode1,
21              R.drawable.explode2,
22              R.drawable.explode3,
23              R.drawable.explode4,
24              R.drawable.explode5,
25              R.drawable.explode6,
26         };
27         Bitmap[] explodes = new Bitmap[explodesID.length];          //爆炸的数组
28         Bitmap hullBackground;                                      //显示生命的背景图片
29         Bitmap hull;                                                //生命的图片
30         Bitmap life;                                                //血块的图片
31         Bitmap changebullet                                         //换枪道具图片
32         int status = 1;          //游戏的状态，1 表示游戏中,2 表示游戏失败即我方飞机没有了生命
33         Paint paint;                                                //画笔
34         Plane plane = new Plane(50, 140, 1, 0, ConstantUtil.life, this);   //初始化我方飞机
35         ArrayList<Bullet> badBullets = new ArrayList<Bullet>();  //敌方飞机发出的子弹
36         ArrayList<Bullet> goodBullets = new ArrayList<Bullet>();//我方飞机发出的子弹
37         ArrayList<Explode> explodeList = new ArrayList<Explode>();   //爆炸
38         ArrayList<ChangeBullet> changeBullets = new ArrayList<ChangeBullet>();
                                                                       //换枪道具
39         ArrayList<EnemyPlane> enemyPlanes;                          //敌方的飞机
40         ArrayList<Life> lifes;                                      //存放补血道具
41         SoundPool soundPool;                                        //声音
42         HashMap<Integer, Integer> soundPoolMap;
43         MediaPlayer mMediaPlayer;                                   //声音
```

说明　该段代码给出了游戏界面的所有成员变量，各个变量表示的含义见代码中的注释。

12.6 游戏实体相关类的实现

前面已经对游戏界面的主框架进行了介绍，接下来将对游戏实体相关的类进行介绍，为以后游戏界面的完善打好基础。

12.6.1 飞机 Plane 类的实现

首先介绍的是飞机类 Plane 的实现，该类为飞机的封装类，其中含有飞机的相关信息以及碰撞检测方法的实现。

1. Plane 类的框架

首先介绍的是 Plane 框架的实现，其代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 Plane.java。

```
1    package wyf.ytl;
2    import android.graphics.Bitmap;                                  //引入相关类
3    ……//该处省略了部分包的引入，读者可自行查看随书光盘源代码的该处
4    import android.graphics.Paint;                                   //引入相关类
5    public class Plane {
6         private int x;                                              //飞机的 x 坐标
7         private int y;                                              //飞机的 y 坐标
8         int life;                                                   //生命
9         private int dir;//飞机的方向,0 静止，1 上，2 右上，3 右，4 右下，5 下，6 左下，7 左，8 左上
10        private int type;                                           //飞机的类型
11        Bitmap bitmap1;                                             //当前向下飞机的图片
12        Bitmap bitmap2;                                             //当前向上飞机的图片
13        Bitmap bitmap3;                                             //当前飞机的图片
14        GameView gameView;                                          //GameView 的引用
15        private int span = 10;                                      //飞机移动步进
16        int bulletType = 1;
17        public Plane(int x, int y, int type, int dir,int life, GameView gameView){
                                                                      //构造器
18             ……//该处省略了成员变量初始化代码，读者可自行查看随书光盘中的源代码
19        }
20        public void initBitmap(){                                   //初始化图片的方法
21             ……//该处省略了图片资源初始化的代码，读者可自行查看随书光盘的源代码
22        }
23        public void draw(Canvas canvas){                            //绘制方法
24             ……//该处省略了主战飞机的绘制方法，将在后面给出
25        }
26        public void fire(){                                         //打子弹的方法
27             ……//该处省略了打子弹的方法，将在后面给出
28        }
29        public boolean contain(Bullet b){                           //判断是否碰到敌机子弹
30             ……//该处省略了与子弹碰撞检测的代码，读者可自行查看随书光盘的源代码
31        }
32        public boolean contain(ChangeBullet cb){                    //判断是否碰到换枪道具
33             ……//该处省略了与换枪道具碰撞检测的代码。读者可自行查看随书光盘的源代码
34        }
35        public boolean contain(EnemyPlane ep){                      //判断是否碰到敌机
36             ……//该处省略了与敌机碰撞检测的方法，将在后面给出
37        }
38        public boolean contain(Life l){                   //检测玩家飞机是否撞上补血道具
```

```
39                ……//该处省略了与补血道具碰撞检测的方法，读者可自行查看随书光盘的源代码
40            }
41            private boolean isContain(int otherX, int otherY, int otherWidth, int otherHeight){
                                                                          //判断两个矩形碰撞
42                ……//该处省略了给定两个继续判断是否碰撞的代码，将在后面给出
43            }
44            ……//该处省略了成员变量的 set 方法，读者可自行查看随书光盘的源代码
45        }
```

- 第 6～16 行为成员变量的声明，其中 dir 表示飞机当前的飞行方向，而 type 为飞机的类型，分别代表敌机 1、敌机 2 或主战飞机等。
- 第 23～25 行为飞机的绘制方法，根据当前飞机的信息在指定位置绘制飞机。
- 第 26～28 行为飞机打子弹的方法，而第 29～40 行为该飞机与各个物体碰撞检测的方法。

2. 绘制方法 draw 的实现

接下来对之前框架的各个方法进行完善，首先完善的是 draw 方法，其代码如下。

```
1     public void draw(Canvas canvas){                                  //绘制方法
2         if(dir == ConstantUtil.DIR_UP){                               //向上
3             canvas.drawBitmap(bitmap2, x, y,new Paint());             //绘制向上的飞机图片
4         }
5         else if(dir == ConstantUtil.DIR_DOWN){                        //向下
6             canvas.drawBitmap(bitmap1, x, y,new Paint());             //绘制向下的飞机图片
7         }
8         else{                                                         //其他情况使用的图片
9             canvas.drawBitmap(bitmap3, x, y,new Paint());
10        }}
```

说明　该方法中根据当前飞机的飞行方向绘制不同的飞机图片，如果需要为飞机添加新的方向，可以再添加新的判断来绘制新的图片。

3. 打子弹方法 fire 的实现

主战飞机或者敌方飞机都会发射子弹打击敌人，而 fire 方法便为发射子弹的方法，其代码如下。

```
1     public void fire(){                                               //打子弹的方法
2         if(bulletType == 1){                                          //当子弹类型为 1 时
3             Bullet b = new Bullet(this.x+75, this.y+8, 1, ConstantUtil.DIR_RIGHT,
              gameView);
4             gameView.goodBullets.add(b);                              //添加到容器中
5         }else if(bulletType == 2){                                    //当子弹类型为 2 时
6             Bullet b = new Bullet(this.x+75, this.y+4, 3, ConstantUtil.DIR_RIGHT,
              gameView);
7             gameView.goodBullets.add(b);                              //添加到容器中
8         }else {                                                       //其他情况
9             Bullet b = new Bullet(this.x+75, this.y+4, 3, ConstantUtil.DIR_RIGHT,
              gameView);
10            gameView.goodBullets.add(b);                              //添加到容器中
11            Bullet b2 = new Bullet(this.x+55, this.y-4, 4, ConstantUtil.DIR_RIGHT_UP,
              gameView);
12            gameView.goodBullets.add(b2);                             //添加到容器中
13            Bullet b3 = new Bullet(this.x+55, this.y+12, 5, ConstantUtil.DIR_RIGHT_DOWN,
          gameView);
14            gameView.goodBullets.add(b3);                             //添加到容器中
15        if(gameView.activity.isSound){                                //需要播放声音时
```

```
16                gameView.playSound(1,0);                    //播放音乐
17            }}
```

说明 该方法根据飞机可发射子弹的类型的不同创建不同的子弹对象，然后将子弹添加到存放子弹的容器中，同时检查声音标志位，当需要播放声音时播放声音。

4. 碰撞敌方飞机方法 contain(EnemyPlane ep)的实现

该方法根据当前飞机的位置与当前敌方飞机的位置判断两者是否已经碰到，如果碰撞，主战飞机的生命自减，如果自减后的生命值小于 0，向 Activity 发送消息通知 Activity 主战飞机已经死亡。该方法有很多重载方法，其原理都相同，只是根据碰到东西的不同做不同的操作，例如，碰到补血道具生命加 1、碰到子弹生命值减 1 等，该方法的代码如下。

```
1    public boolean contain(EnemyPlane ep){                    //检测是否碰撞到敌方飞机
2        if(isContain(ep.x, ep.y, ep.bitmap.getWidth(), ep.bitmap.getHeight())){
                                                               //检测成功
3            this.life--;                                      //自己的生命减 1
4            if(this.life<0){                                  //当生命小于 0 时
5                gameView.status = 2;                          //切换状态
6                if(gameView.mMediaPlayer.isPlaying()){        //当背景音乐正在播放时
7                    gameView.mMediaPlayer.stop();             //停止播放背景音乐
8                }
9                if(gameView.activity.isSound){                //当允许播放声音时
10                   gameView.playSound(3,0);                  //播放游戏失败的声音
11               }
12               gameView.activity.myHandler.sendEmptyMessage(1);
                                                               //向主 Activity 发送 Handler 消息
13           }
14           return true;                                      //返回碰撞成功
15       }
16       return false;                                         //返回碰撞失败
17   }
```

- 第 2 行调用 isContain 方法判断主战飞机与敌方飞机是否碰撞，isContain 方法将在之后进行介绍。
- 第 3～14 行当碰撞成功时，生命减 1，而当生命值减到小于 0 时，改变游戏界面的状态，然后发送 Handler 消息通知 Activity 游戏失败。

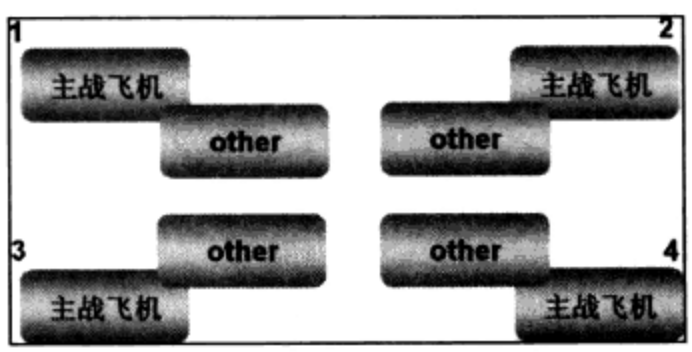
5. 碰撞方法 isContain 的实现

该方法根据判断给定的矩形包围框与当前飞机位置对应的矩形包围框是否有重叠检测碰撞，此种碰撞检测的原理第 7 章已经详细介绍，在此就不进行赘述，下面仅给出该方法的代码。

```
1    private boolean isContain(int otherX, int otherY, int otherWidth, int otherHeight){
                                                               //判断两个矩形是否碰撞
2        int xd = 0;                                           //大的 x
3        int yd = 0;                                           //大的 y
4        int xx = 0;                                           //小的 x
5        int yx = 0;                                           //小的 y
6        int width = 0;
7        int height = 0;
8        boolean xFlag = true;                                 //玩家飞机 x 是否在前
9        boolean yFlag = true;                                 //玩家飞机 y 是否在前
```

```
10          if(this.x >= otherX){                          //主战飞机 x 大于其他物体的 x
11               xd = this.x;
12               xx = otherX;
13               xFlag = false;                             //设置标志位
14          }else{                                          //主战飞机 y 小于其他物体的 x
15               xd = otherX;
16               xx = this.x;
17               xFlag = true;                              //设置标志位
18          }
19          if(this.y >= otherY){                          //主战飞机 y 大于其他物体的 y
20               yd = this.y;
21               yx = otherY;
22               yFlag = false;                             //设置标志位
23          }else{                                          //主战飞机 y 小于其他物体的 y
24               yd = otherY;
25               yx = this.y;
26               yFlag = true;                              //设置标准位
27          }
28          if(xFlag == true){                             //当主战飞机 x 方向在前时
29               width = this.bitmap1.getWidth();
30          }else {                                         //当主战飞机 x 方向在后时
31               width = otherWidth;
32          }
33          if(yFlag == true){                             //当主战飞机 y 方向在前时
34               height = this.bitmap1.getHeight();
35          }else{                                          //当主战飞机 y 方向在后时
36               height = otherHeight;
37          }
38          if(xd>=xx&&xd<=xx+width-1&&
39                    yd>=yx&&yd<=yx+height-1){            //首先判断两个矩形是否重叠
40             double Dwidth=width-xd+xx;                  //重叠区域宽度
41               double Dheight=height-yd+yx;                                  //重叠区域高度
42               if(Dwidth*Dheight/(otherWidth*otherHeight)>=0.20){
                                                            //重叠面积超 20%则判定为碰撞
43                    return true;
44               }
45          }
46          return false;
47     }
```

- 第 10～17 行判断 x、y 的大小并设置标志位，4 种情况如图 12-15 所示。
- 第 38～44 行则根据矩形的宽和高判断重叠的面积是否超过 20%。



▲图 12-15　碰撞检测原理

12.6.2　敌方飞机 EnemyPlane 类的实现

前面已经对主战飞机类进行了介绍，接下来介绍的是敌方飞机类。敌方飞机与主战飞机不同的是敌方飞机会自动按照一定的轨迹进行运动，敌方飞机类的代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 EnemyPlane.java。

```
1    package wyf.ytl;
2    import android.graphics.Bitmap;                        //引入相关类
3    ……//该处省略了部分包的引入，读者可自行查看随书光盘源代码的该处
4    import android.os.Message;                             //引入相关类
```

```
5    public class EnemyPlane {
6         int x = ConstantUtil.screenWidth;                    //飞机的坐标
7         int y;
8         boolean status;                                      //飞机的状态
9         long touchPoint;                                     //触发点，即什么时候出现
10        int type;                                            //飞机的类型
11        int life;                                            //生命
12        int spanX = 10;                                      //飞机移动的 X 像素
13        int spanY = 5;                                       //飞机移动的 Y 像素
14        Bitmap bitmap;                                       //飞机图片
15        int start;                                           //当前出发点
16        int target;                                          //当前目标点
17        int step;                                            //当前处于当前路径片段中第几步
18        int[][] path;
19   public EnemyPlane(int start,int target,int step,int[][] path, boolean status,long
     touchPoint, int type, int life){
20        ……//该处省略了成员变量的初始化工作，读者可自行查看随书光盘的源代码
21        }
22        public void draw(Canvas canvas){                     //绘制方法
23             canvas.drawBitmap(bitmap, x, y, new Paint());   //在指定位置绘制飞机
24        }
25        public void move(){                                  //飞机移动的方法
26             if(step==path[2][start]){                       //一段路径走完,到下一段路径开始
27                  step=0;                                    //当前的步数设为 0
28                  start=(start+1)%(path[0].length);
29                  target=(target+1)%(path[0].length);
30                  this.x=path[0][start];                     //移动
31                  this.y=path[1][start];
32             }else{                                          //一段路径没有走完，继续走
33                  int xSpan=(path[0][target]-path[0][start])/path[2][start];
34                  int ySpan=(path[1][target]-path[1][start])/path[2][start];
35                  this.x=this.x+xSpan;                       //移动飞机
36                  this.y=this.y+ySpan;
37                  step++;                                    //步数自加
38             }
39        }
40        public void fire(GameView gameView){                //打子弹的方法
41             if(type == 3 && Math.random()<ConstantUtil.BooletSpan2){     //关口时
42                  Bullet b1 = new Bullet(x, y, 2, ConstantUtil.DIR_LEFT,gameView);
43                  Bullet b2 = new Bullet(x, y, 2, ConstantUtil.DIR_LEFT_DOWN,gameView);
44                  Bullet b3 = new Bullet(x, y, 2, ConstantUtil.DIR_LEFT_UP,gameView);
45                  gameView.badBullets.add(b1);                          //添加到容器中
46                  gameView.badBullets.add(b2);                          //添加到容器中
47                  gameView.badBullets.add(b3);                          //添加到容器中
48             }else if(Math.random()<ConstantUtil.BooletSpan){          //正常情况
49                  if(this.type == 4){                                   //飞机类型
50                       Bullet b = new Bullet(x, y, 2, ConstantUtil.DIR_RIGHT,gameView);
51                       gameView.badBullets.add(b);                      //添加到容器中
52                  }else{
53                       Bullet b = new Bullet(x, y, 2, ConstantUtil.DIR_LEFT,gameView);
54                       gameView.badBullets.add(b);                      //添加到容器中
55                  }}}
56        private boolean isContain(int otherX, int otherY, int otherWidth, int otherHeight){
                                                                 //判断两个矩形的碰撞
57             ……//该处省略了两个矩形的碰撞检测的代码，读者可自行查看随书光盘的源代码
58        }
59        public boolean contain(Bullet b,GameView gameView){ //判断子弹是否打中敌方飞机
60             ……//该处省略了飞机与子弹碰撞检测的代码，读者可自行查看随书光盘的源代码
61        }
62        ……//该处省略了成员变量的 set 方法，读者可自行查看随书光盘的源代码
63   }
```

- 第 19～21 行为该类的构造方法，在方法中只是对各个成员变量进行赋值。
- 第 25～39 行为敌方飞机移动的方法，该方法中会根据事先定义好的路径进行循环移动，每次取出出发点以及结束点，然后在两点之间进行无极移动。
- 第 40～55 行为敌方飞机发射子弹的方法，根据敌方飞机类型的不同，发射不同的子弹，而当在关口时，需要发射多发子弹来增强游戏的难度。
- 第 56～63 行两个方法的实现与之前主战飞机 Plane 类的基本相同，在此就不再赘述，读者可参照之前的代码进行开发。

12.6.3　子弹 Bullet 类的实现

前面已经介绍了主战飞机和敌方飞机，接下来对子弹类 Bullet 进行介绍，其代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 Bullet.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    import android.graphics.Bitmap;                                     //引入相关类
3    ……//该处省略了部分包的引入，读者可自行查看随书光盘源代码的该处
4    import android.graphics.Paint;                                      //引入相关类
5    public class Bullet {
6         int x;                                                         //子弹的 x 坐标
7         int y;                                                         //子弹的 y 坐标
8         int dir;        //子弹的方向,0 静止，1 上,2 右上，3 右，4 右下，5 下，6 左下，7 左，8 左上
9         int type;                                                      //子弹的类型
10        Bitmap bitmap;                                                 //当前子弹的图片
11        GameView gameView;                                             //gameView 的引用
12        private int moveSpan = 10;                                     //移动的像素
13        public Bullet(int x, int y, int type, int dir,GameView gameView){
14             this.gameView = gameView;                                 //得到 gameView 的引用
15             this.x = x;                                               //得到 x 坐标
16             this.y = y;                                               //得到 y 坐标
17             this.type = type;                                         //得到类型
18             this.dir = dir;                                           //得到方向
19             this.initBitmap();                                        //初始化图片资源
20        }
21        public void initBitmap(){                                      //初始化图片资源的方法
22             if(type == 1){                                            //当类型为 1 时
23                  bitmap = BitmapFactory.decodeResource(gameView.getResources(),
                    R.drawable. bullet1);
24             }else if(type == 2){                                      //类型为 2 时
25                  bitmap = BitmapFactory.decodeResource(gameView.getResources(),
                    R.drawable. bullet2);
26             }else if(type == 3){                                      //类型为 3 时
27                  bitmap = BitmapFactory.decodeResource(gameView.getResources(),
                    R.drawable. bullet3);
28             }else if(type == 4){                                      //类型为 4 时
29                  bitmap = BitmapFactory.decodeResource(gameView.getResources(),
                    R.drawable. bullet4);
30             }else if(type == 5){                                      //类型为 5 时
31                  bitmap = BitmapFactory.decodeResource(gameView.getResources(),
                    R.drawable. bullet5);
32             }
33        }
34        public void draw(Canvas canvas){                               //绘制方法
35             canvas.drawBitmap(bitmap, x, y,null);                     //绘制图片
36        }
37        public void move(){                                            //移动的方法
```

```
38              if(dir == ConstantUtil.DIR_RIGHT){                    //向右移动
39                   this.x = this.x + moveSpan;
40              }else if(dir == ConstantUtil.DIR_LEFT){               //向左移动
41                   this.x = this.x - moveSpan;
42              }else if(dir == ConstantUtil.DIR_LEFT_DOWN){          //向左下移动
43                   this.x = this.x - moveSpan;
44                   this.y = this.y + moveSpan;
45              }else if(dir == ConstantUtil.DIR_LEFT_UP){            //向左上移动
46                   this.x = this.x - moveSpan;
47                   this.y = this.y - moveSpan;
48              }else if(dir == ConstantUtil.DIR_RIGHT_UP){           //右上移动
49                   this.x = this.x + moveSpan;
50                   this.y = this.y - moveSpan;
51              }else if(dir == ConstantUtil.DIR_RIGHT_DOWN){         //右下移动
52                   this.x = this.x + moveSpan;
53                   this.y = this.y + moveSpan;
54              }}}
```

- 第 21～33 行根据当前子弹的类型进行图片的初始化工作。
- 第 37～54 行根据子弹当前的方向移动子弹的位置，只需改变子弹的坐标即可。

12.6.4 其他相关类的实现

前面已经对主要的实体类进行了介绍，接下来将对爆炸类、补血道具类、换枪道具类等其他相关类进行介绍。

1. 爆炸类的实现

首先介绍的是爆炸类的实现，其代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 Explode.java。

```
1    package wyf.ytl;                                                //声明包语句
2    import android.graphics.Bitmap;                                 //引入相关类
3    import android.graphics.Canvas;                                 //引入相关类
4    import android.graphics.Paint;                                  //引入相关类
5    public class Explode {
6         int x;                                                     //爆炸的 x 坐标
7         int y;                                                     //爆炸的 y 坐标
8         Bitmap[] bitmaps;                                          //所有爆炸的帧数组
9         int k = 0;                                                 //当前帧
10        Bitmap bitmap;                                             //当前帧图片
11        Paint paint;                                               //画笔
12        public Explode(int x, int y, GameView gameView){           //构造器
13             this.x = x;
14             this.y = y;                                           //得到爆炸的位置
15             this.bitmaps = gameView.explodes;                     //得到图片帧数组
16             paint = new Paint();                                  //初始化画笔
17        }
18        public void draw(Canvas canvas){                           //绘制方法
19             canvas.drawBitmap(bitmap, x, y, paint);               //绘制图片
20        }
21        public boolean nextFrame(){                                //换帧，成功返回 true
22             if(k < bitmaps.length){                               //当前帧没到最后一帧
23                  bitmap = bitmaps[k];                             //换帧
24                  k++;                                             //k 自动加 1
25                  return true;                                     //返回成功
26             }
```

```
27                return false;
28            }}
```

说明　该类包含一个动画帧数组，当需要换帧时，只需将当前帧换成数组中的下一帧，绘制时根据爆炸的位置绘制当前帧。

2. 爆炸线程类的实现

有了爆炸类，接下来就需要开发爆炸线程类为爆炸类换帧，其代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 ExplodeThread.java。

```
1     package wyf.ytl;
2     import java.util.ArrayList;                                    //引入相关类
3     public class ExplodeThread extends Thread{
4          private boolean flag = true;                              //循环标记位
5          private int span = 100;                                   //睡眠的毫秒数
6          GameView gameView;                                        //GameView 的引用
7          ArrayList<Explode> deleteExplodes = new ArrayList<Explode>();
                                                                     //用于暂时存放需要删除的爆炸
8          public ExplodeThread(GameView gameView){                  //构造器
9               this.gameView = gameView;                            //得到 gameView 的引用
10         }
11         public void setFlag(boolean flag){                        //设置循环标记位
12              this.flag = flag;
13         }
14         public void run(){                                        //重写 run 方法
15              while(flag){
16                   try{                                            //防止并发访问的异常
17                        for(Explode e : gameView.explodeList){
18                             if(e.nextFrame()){}                   //换帧
19                             else{                                 //当没有下一帧时删除该爆炸
20                                  deleteExplodes.add(e);           //删除该爆炸对象
21                             }
22                        }
23                        gameView.explodeList.removeAll(deleteExplodes); //彻底删除
24                        deleteExplodes.clear();                    //清空临时容器
25                   }catch(Exception e){                            //捕获异常
26                        e.printStackTrace();                       //打印异常信息
28                   }
28                   try{
29                        Thread.sleep(span);                        //睡眠指定毫秒数
30                   }catch(Exception e){                            //捕获异常
31                        e.printStackTrace();                       //打印异常信息
32                   }}}}
```

说明　该类只需每隔 span 毫秒对所有的爆炸对象进行换帧，当没有帧可换时，删除该爆炸对象。

3. 道具类的实现

本游戏中的道具有两种，一种是补血道具，另一种是换枪道具，两种道具除了碰撞之后的处理代码稍有不同外其他实现基本相同，所以，此处只给出补血道具的实现，读者可参考该实现方法自行开发换枪道具程序，补血道具的代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 Life.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.graphics.Bitmap;                           //引入相关类
3    import android.graphics.Canvas;                           //引入相关类
4    import android.graphics.Paint;                            //引入相关类
5    public class Life {
6         int x = ConstantUtil.screenWidth;                    //血道具的坐标
7         int y;
8         boolean status;                                      //血道具的状态
9         long touchPoint;                                     //触发点
10        Bitmap bitmap;
11        int start;                                           //当前出发点
12        int target;                                          //当前目标点
13        int step;                                            //当前处于当前路径片段中第几步
14        int[][] path;
15        public Life(int start,int target,int step,int[][] path, boolean status,long
          touchPoint){//构造器
16             this.start=start;                               //当前出发点
17             this.target=target;                             //当前目标点
18             this.step=step;                                 //步数
19             this.path=path;                                 //路径
20             this.status = status;                           //状态
21             this.touchPoint = touchPoint;
22             this.x=path[0][start];                          //起始点的 x 坐标
23             this.y=path[1][start];                          //起始点的 y 坐标
24        }
25        public void draw(Canvas canvas){                     //重绘方法
26             canvas.drawBitmap(bitmap, x, y, new Paint());        //在指定位置绘制图片
27        }
28        public void move(){                                  //道具移动的方法
29             if(step==path[2][start]){                       //一段路径走完,到下一段路径开始
30                  step=0;                                    //步数归 0
31                  start=(start+1)%(path[0].length);
32                  target=(target+1)%(path[0].length);
33                  this.x=path[0][start];                     //当前点的 x 坐标
34                  this.y=path[1][start];
35             }else{                                          //一段路径没有走完，继续走
36                  int xSpan=(path[0][target]-path[0][start])/path[2][start];
37                  int ySpan=(path[1][target]-path[1][start])/path[2][start];
38                  this.x=this.x+xSpan;                       //移动道具
39                  this.y=this.y+ySpan;
40                  step++;                                    //步数自动加 1
41             }}}
```

说明　该类的实现与之前介绍的敌方飞机的实现基本相同，主要是移动方法 move 的开发，在 move 方法中根据路径以及当前的步数来移动物体。

12.7 游戏界面的完善

前面已经介绍了游戏界面的框架以及与游戏实体相关的各个类，接下来将对之前游戏界面的框架进行完善，逐步完成游戏的所有功能。

12.7.1 地图类 Maps 的实现

首先介绍的是地图类，该类中包含所有敌方飞机以及道具的飞行路线，其代码如下。

代码位置：见随书光盘中源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 Maps.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    import java.util.ArrayList;                                         //引入相关类
3    public class Maps {
4         public static int[][] pathA = {                                //路径 A
5              {480, 280, 80, -120},                                     //路径中每个点的 X 坐标
6              {50, 110, 50, 110},                                       //路径中每个点的 Y 坐标
7              {15, 15, 15, 15},                                         //路径中两点间的步数
8         };
9         public static int[][] pathB = {                                //路径 B
10             {480, 280, 80, -120},                                     //路径中每个点的 X 坐标
11             {200, 260, 200, 260},                                     //路径中每个点的 Y 坐标
12             {15, 15, 15, 15},                                         //路径中两点间的步数
13        };
14        ……//该处省略了其他路径的数组，读者可自行查看随书光盘的源代码
15        public static int[][] pathJ = {                                //路径 J
16             {480, -100},                                              //路径中每个点的 X 坐标
17             {140, 140},                                               //路径中每个点的 Y 坐标
18             {60, 25},                                                 //路径中两点间的步数
19        };
20        public static int[][] pathK = {                                //路径 K
21             {310, 260, 350, 350, 10, 10, 350},                        //路径中每个点的 X 坐标
22             {130, 230, 20, 130, 0, 210, 230},                         //路径中每个点的 Y 坐标
23             {30, 20, 30, 20, 40, 25, 35},                             //路径中两点间的步数
24        };
25        public static ArrayList<Life> getFirstLife(){                  //为第一关添加补血道具
26             ArrayList<Life> lifes = new ArrayList<Life>();
27             lifes.add(new Life(0, 1, 0, pathJ, false, 190));          //添加补血道具
28             lifes.add(new Life(0, 1, 0, pathJ, false, 570));          //添加补血道具
29             return lifes;
30        }
31        public static ArrayList<ChangeBullet> getFirstBullet(){
                                                                         //为第一关添加吃了换枪道具
32             ArrayList<ChangeBullet> lifes = new ArrayList<ChangeBullet>();
33             lifes.add(new ChangeBullet(0, 1, 0, pathJ, false, 90));     //添加换枪道具
34             lifes.add(new ChangeBullet(0, 1, 0, pathJ, false, 290));    //添加换枪道具
35             return lifes;
36        }
37        //参数分别为出发点、目标点、当前段上的第几步、路径数组、状态、触发时间、类型、生命
38        public static ArrayList<EnemyPlane> getFirst(){      //得到第一关的所有敌方飞机
39             ArrayList<EnemyPlane> enemyPlanes = new ArrayList<EnemyPlane>();
40             enemyPlanes.add(new EnemyPlane(0, 1, 0, pathA, false, 30, 1, 1));
41             ……//该处省略了部分敌机的初始化工作，读者可自行查看随书光盘中的源代码
42             enemyPlanes.add(new EnemyPlane(0, 1, 0, pathI, false, 600, 1, 1));
                                                                         //添加敌方飞机
43             enemyPlanes.add(new EnemyPlane(0, 1, 0, pathK, false, 640, 3, 10));//关口
44             return enemyPlanes;
45        }}
```

- 第 4～24 行定义了若干个敌方飞机或者道具的飞行路径，每个路径用一个二维数组表示，第一维表示路径中各个点的 x 坐标，第二维表示各个点的 y 坐标，第三维表示从当前点到下一点之间所走的步数。
- 第 25～30 行的方法为第一关添加补血道具，在方法中先创建一个 ArrayList 容器，然后向容器中添加需要的补血道具对象。
- 第 31～36 行的方法为第一关添加换枪道具，在方法中同样需要创建一个容器，然后在容器中添加多个换枪道具的对象。

- 第 38～45 行的方法为添加敌方飞机的方法，与添加道具的方法一样，向容器中添加各个敌方飞机的对象，然后将容器返回。

提示 在真实的游戏开发中，移动物的路径设计可以使用地图设计器，有兴趣的读者可以自行开发飞机游戏的地图设计器，这种设计器只需根据设计人员单击的各个位置设计出一条曲线，并填写曲线中各个点之间移动的步数。

12.7.2 背景滚动类 GameViewBackGroundThread 的实现

前面已经将游戏实体相关的各个类介绍完毕，但是这些物体还不能出现在界面中，使用 GameViewBackGroundThread 类可以在适当的时间将这些物体绘制在界面中，该类同时还负责游戏背景的滚动工作，其代码如下。

代码位置：见随书光盘源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 GameViewBackGroundThread.java。

```
1    package wyf.ytl;                                          //声明包语句
2    public class GameViewBackGroundThread extends Thread{
3         private int sleepSpan = 100;                         //睡眠的毫秒数
4         private int span = 3;                                //图片移动的步长
5         private boolean flag = true;                         //循环标志位
6         GameView gameView;                                   //GameView 的引用
7         long touchTime = 0;                                  //当前所到的时间
8         public GameViewBackGroundThread(GameView gameView){  //构造器
9              this.gameView = gameView;                       //得到 gameView 的引用
10        }
11        public void setFlag(boolean flag){                   //设置标记位
12             this.flag = flag;
13        }
14        public void run(){                                   //重写 run 方法
15             while(flag){                                    //循环
16                  if(gameView.status == 1){                  //游戏进行中
17                       gameView.backGroundIX -= span;        //移动背景图片
18                       if(gameView.backGroundIX <-ConstantUtil.pictureWidth){
19                            gameView.i = (gameView.i+1)%ConstantUtil.pictureCount;
20                            gameView.backGroundIX+=ConstantUtil.pictureWidth;
21                       }
22                       gameView.cloudX -= span;                //移动云彩
23                       if(gameView.cloudX<-1000){              //当坐标小于-1000 时
24                            gameView.cloudX = 1000;
25                       }
26                       touchTime++;                            //时间自动加 1
27                       try{
28                            for(EnemyPlane ep : gameView.enemyPlanes){
                                                                 //到时间出现敌方飞机
29                                 if(ep.touchPoint == touchTime){
30                                      ep.status = true;        //设置可见性
31                                      if(ep.type == 3){        //到关口了
32                                           gameView.status = 3;//设置状态
33                                      }
34                                 }
35                            }
36                            for(Life l : gameView.lifes){    //到时间出现补血道具
37                                 if(l.touchPoint == touchTime){ //到达出现的时间
38                                      l.status = true;         //设置补血道具的可见性
39                                 }
```

```
40                          }
41                          for(ChangeBullet cb : gameView.changeBullets){
                                                                    //到时间出现换枪道具
42                              if(cb.touchPoint == touchTime){//到时间出现换枪道具
43                                  cb.status = true;               //设置可见性
44                              }
45                          }
46                      }catch(Exception e){                        //捕获异常
47                          e.printStackTrace();                    //打印异常信息
48                      }
49                      if(touchTime == 641){                       //到关口时
50                          this.flag = false;
51                      }
52                  }try{
53                      Thread.sleep(sleepSpan);                    //睡眠指定毫秒数
54                  }catch(Exception e){                            //捕获异常
55                      e.printStackTrace();                        //打印异常信息
56                  }}}}
```

- 第 17～21 行移动背景图片，只需每次更改图片的坐标，而当滚动过远时将坐标置回初始值。
- 第 22～25 行移动云彩图片，同样只改变图片的坐标即可。
- 第 28～45 行分别对装有敌机、补血道具、换枪道具的容器进行循环，检测各个对象的触发时间与当前的计时器 touchTime 是否相同，当相同时便将可见性标志位设为 true。

12.7.3　物体移动线程 MoveThread 的实现

前面介绍了敌方飞机以及道具的出现线程，接下来介绍移动线程 MoveThread，该线程主要负责除主战飞机外所有可移动物的移动，其代码如下。

代码位置：见随书光盘源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 MoveThread.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import java.util.ArrayList;                               //引入相关类
3    public class MoveThread extends Thread{
4        private int sleepSpan = 10;                           //睡眠的毫秒数
5        private boolean flag = true;                          //循环标志位
6        GameView gameView;                                    //GameView 的引用
7        ArrayList<Bullet> deleteBullet = new ArrayList<Bullet>();//临时存放需要删除的子弹
8        ArrayList<EnemyPlane> deleteEnemy = new ArrayList<EnemyPlane>();
                                                               //临时存放需要删除的敌方飞机
9        ArrayList<Life> deleteLife = new ArrayList<Life>(); //临时存放需要删除的补血道具
10       ArrayList<ChangeBullet> deleteChangeBullets = new ArrayList<ChangeBullet>();
11       private int countBulletMove = 0;                      //我方子弹移动的计数器
12       private int countMyBulletN = 3;                       //每多少次循环移动一下
13       private int countEnemyMove = 0;                       //敌方飞机移动计数器
14       private int countEnemyN = 8;                          //每多少次循环移动一下
15       private int countEnemyFire = 0;                       //敌方飞机发射子弹计数器
16       private int countEnemyFireN = 20;                     //每多少次循环发射一发子弹
17       private int countEnemyBulletMove = 0;                 //敌方子弹移动的计数器
18       private int countEnemyBulletN = 3;                    //每多少次循环移动一下
19       public MoveThread(GameView gameView){                 //构造器
20           this.gameView = gameView;                         //得到 GameView 的引用
21       }
22       public void run(){                                    //重写 run 方法
23           while(flag){                                      //循环
24               if(countBulletMove == 0){                     //我方子弹移动
25                   try{
26                       for(Bullet b : gameView.goodBullets){//对我方子弹进行循环
```

```
27                                  b.move();                       //移动子弹
28                                  if(b.x<0 || b.x>ConstantUtil.screenWidth
29                                          || b.y<0 || b.y>ConstantUtil.screenHeight){
30                                      deleteBullet.add(b); //当出了屏幕范围后删除子弹
31                                  }
32                                  else{                           //在屏幕范围内
33                                      for(EnemyPlane ep : gameView.enemyPlanes){
34                                          if(ep.status == true){
35                                              if(ep.contain(b,gameView)){
                                                                    //打中敌方飞机
36                                                  Explode e = new Explode(ep.x, ep.y,
                                                    gameView);
37                                                  gameView.explodeList.add(e);
                                                                    //添加一个爆炸对象
38                                                  deleteBullet.add(b);//删除该子弹
39                                              }}}}}
40                              gameView.goodBullets.removeAll(deleteBullet);//彻底删除子弹
41                              deleteBullet.clear();                   //清空临时容器
42                          }catch(Exception e){}                       //捕获异常
43                      }
44                      ……//该处省略了其他移动物的移动代码，读者可自行查看随书光盘中的源代码
45                      if(countEnemyFire == 0){                    //敌方飞机打子弹
46                          try{
47                              for(EnemyPlane ep : gameView.enemyPlanes){
                                                                    //对所有敌方飞机循环
48                                  if(ep.status == true){          //可见性为 true 时
49                                      ep.fire(gameView);          //打子弹
50                                  }
51                              }
52                          }catch(Exception e){                    //捕获异常
53                              e.printStackTrace();                //打印异常信息
54                          }
55                      }
56                      countBulletMove = (countBulletMove+1)%countMyBulletN;  //循环自动加 1
57                      countEnemyMove = (countEnemyMove+1)%countEnemyN;   //循环自动加 1
58                      countEnemyFire = (countEnemyFire+1)%countEnemyFireN;  //循环自动加 1
59                      countEnemyBulletMove = (countEnemyBulletMove+1)%countEnemyBulletN;
                                                                    //循环的自动加 1
60                      try{                                        //睡眠休息
61                          Thread.sleep(sleepSpan);
62                      }catch(Exception e){                        //捕获异常
63                          e.printStackTrace();                    //打印异常信息
64                      }}}}
```

- 第 7～10 行初始化用来临时存放需要删除的移动物的容器。在下面的循环中，遇到需要删除的物体时，先将物体添加到临时容器中，当循环结束后一次性将其删除。
- 第 11～18 行为各个移动物的计数器以及循环量，在下面的循环中，每次循环都将各个计数器自加，而当计数器的值与循环量相同时移动该物体，这种方法的好处是可以将所有物体的移动代码放到一个循环当中。
- 第 24～43 行为主战飞机子弹移动的代码，先对所有子弹进行循环移动，当移动到屏幕之外时将其删除，随后再对每个子弹与敌方飞机进行碰撞检测，查看是否击中敌方飞机。
- 第 44 行省略了其他移动物的移动代码，与主战飞机子弹移动的代码原理相同，读者可自行开发或者查看随书光盘本章案例的源代码。
- 第 45～55 行同样对所有装有敌方飞机的容器进行循环并使每个敌方飞机发射炮弹。

12.7.4　键盘监听线程 KeyThread 的实现

前面已经将本游戏中用到的类基本介绍完毕，接下来介绍最后一个类——键盘监听线程类，该类负责定时检测当前键盘的状态，然后根据状态调用相应的处理代码，其代码如下。

代码位置：见随书光盘源代码/第 12 章/ Plane/src/wyf/ytl 目录下的 KeyThread.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    public class KeyThread extends Thread{
3         int span = 20;                                                 //睡眠的毫秒数
4         int countMove = 0;                                             //飞机移动的计数器
5         int countFine = 0;                                             //飞机发射子弹的计数器
6         int moveN = 3;                                                 //每 3 次循环移动一下
7         int fineN = 5;                                                 //每 5 次循环发一次子弹
8         PlaneActivity activity;                                        //Activity 的引用
9         private boolean flag = true;                                   //循环标志
10        int action;                                                    //键盘状态码
11        private boolean KEY_UP = false;                                //向上键是否被按下
12        private boolean KEY_DOWN = false;                              //向下键是否被按下
13        private boolean KEY_LEFT = false;                              //向左的键被按下
14        private boolean KEY_RIGHT = false;                             //向右的键被按下
15        private boolean KEY_A = false;                                 //A 的键被按下
16        private boolean KEY_B = false;                                 //B 的键被按下
17        public KeyThread(PlaneActivity activity){                      //构造器
18             this.activity = activity;                                 //Activity 的引用
19        }
20        public void run(){                                             //重写的方法
21             while(flag){                                              //循环
22                  action = activity.action;                            //得到当前键盘的状态码
23                  if((action & 0x20) != 0){                            //上
24                       KEY_UP = true;
25                  }else{
26                       KEY_UP = false;
27                  }
28                  ……//该处省略了其他按键是否被按下的检测代码，读者可自行查看随书光盘中的源代码
29                  if((action & 0x01) != 0){                            //B 键
30                       KEY_B = true;
31                  }else{
32                       KEY_B = false;                                  //设置标志位
33                  }
34                  if(activity.gameView.status == 1 || activity.gameView.status == 3){
35                       if(countMove == 0){                   //每循环 moveN 次移动一次
36                            if(KEY_UP == true){                        //向上键被按下
37                                 if(!((activity.gameView.plane.getY() -
38                                 activity.gameView.plane.getSpan()) < ConstantUtil.
                                   top)){
39                                      activity.gameView.plane.setY(
40                                      activity.gameView.plane.getY() - activity.
                                        gameView.plane. getSpan());
41                                 }
42                                 activity.gameView.plane.setDir(ConstantUtil.DIR_UP);
43                            }
44                            if(KEY_DOWN == true){                      //向下键被按下
45                                 if(!((activity.gameView.plane.getY() + activity.
                                   gameView.plane. getSpan()) >
46                                 ConstantUtil.screenHeight - activity.gameView.
                                   plane.bitmapl. getHeight())){
47                                      activity.gameView.plane.setY(
48                                      activity.gameView.plane.getY() + activity.
```

```
                                        gameView.plane. getSpan());
49                                  }
50                                  activity.gameView.plane.setDir(ConstantUtil.DIR_
                                    DOWN);
51                              }
52                              if(KEY_LEFT == true){                    //向左键被按下
53                              if(!((activity.gameView.plane.getX() - activity.
                                gameView.plane. getSpan()) < -40)){
54                                      activity.gameView.plane.setX(
55                                      activity.gameView.plane.getX() - activity.
                                        gameView.plane. getSpan());
56                                  }
57                              }
58                              if(KEY_RIGHT == true){                   //向右键被按下
59                                  if(!((activity.gameView.plane.getX() + activity.
                                    gameView.plane. getSpan()) >
60                                 ConstantUtil.screenWidth - activity.gameView.plane.
                                    bitmap1. getWidth())){
61                                      activity.gameView.plane.setX(
62                                      activity.gameView.plane.getX() + activity.
                                        gameView.plane. getSpan());
63                                  }
64                              }
65                  if(KEY_RIGHT == false && KEY_LEFT == false && KEY_DOWN == false && KEY_UP
                    == false){
66                                  activity.gameView.plane.setDir(ConstantUtil.DIR_STOP);
67                              }
68                              if(countFine == 0){              //每 fineN 次发射一次子弹
69                                  if(KEY_A == true){           //A 键被按下
70                                      activity.gameView.plane.fire();
71                                  }
72                                  if(KEY_B == true){           //B 键被按下
73                                  }}}
74                          countMove = (countMove+1)%moveN;     //countMove 自动加 1
75                          countFine = (countFine+1)%fineN;     //countFine 自动加 1
76                      }
77                      try{
78                          Thread.sleep(span);                  //睡眠指定毫秒数
79                      }catch(Exception e){                     //捕获异常
80                          e.printStackTrace();                 //打印异常信息
81                      }}}}
```

- 第 11～16 行为各个按键是否被按下的标志位，第 23～33 行根据键盘状态码 action 判断当前哪些按键被按下。
- 第 36～64 行根据被按下按键的状态移动主战飞机并更改主战飞机的方向。
- 第 65～73 行当按下按键 A 时调用主战飞机的 fire 方法发射子弹。
- 第 74～75 行将两个计数器分别自动加 1，对 moveN 或 fineN 取余可以直接达到循环自动加 1 的目的。

12.7.5 图片初始化方法 initBitmap 的实现

前面已经介绍过各个线程类的实现，接下来将对图片资源的初始化方法 initBitmap 进行介绍，其代码如下。

```
1    public void initBitmap(){                                    //初始化所有图片
2        if(activity.processView != null){                        //当加载界面不为空时
```

```
3            activity.processView.process += 20;                    //加载界面的进度条
4        }
5        paint = new Paint();                                       //初始化画笔
6        paint.setColor(Color.BLACK);                               //设置画笔颜色
7        cloud = BitmapFactory.decodeResource(getResources(), R.drawable.cloud);
                                                                    //云彩
8        battleback = BitmapFactory.decodeResource(getResources(), R.drawable.
         battleback);
                                                                    //大背景图片
9        for(int i=0; i<battlebacks.length; i++){                   //切成小图片
10           battlebacks[i] = Bitmap.createBitmap(battleback, ConstantUtil.
             pictureWidth*i, 0,
11                ConstantUtil.pictureWidth, ConstantUtil.pictureHeight);   //切割
12       }
13       battleback = null;                                         //释放掉大图
14       enemyPlane1 = BitmapFactory.decodeResource(getResources(), R.drawable.plane4);
                                                                    //敌方飞机 1 的图片
15       enemyPlane2 = BitmapFactory.decodeResource(getResources(), R.drawable.plane5);
                                                                    //敌方飞机 2 的图片
16       enemyPlane3 = BitmapFactory.decodeResource(getResources(), R.drawable.plane6);
                                                                    //敌机方飞 3 的图片
17       enemyPlane4 = BitmapFactory.decodeResource(getResources(), R.drawable.plane7);
                                                                    //敌机方飞 4 的图片
18       hullBackground = BitmapFactory.decodeResource(getResources(),R.drawable.
         hullbackground);
19       hull = BitmapFactory.decodeResource(getResources(), R.drawable.hull);
                                                                    //换血道具的图片
20       life = BitmapFactory.decodeResource(getResources(), R.drawable.life);
                                                                    //换血道具块图片
21       changebullet = BitmapFactory.decodeResource(getResources(), R.drawable.
         changebullet);
22       number[0] = BitmapFactory.decodeResource(getResources(), R.drawable.number0);
23       ……//该处省略了数字 1～8 的初始化代码，读者可自行查看随书光盘的源代码
24       number[9] = BitmapFactory.decodeResource(getResources(), R.drawable.number9);
25       for(int i=0; i<explodes.length; i++){                      //初始化爆炸图片
26           explodes[i] = BitmapFactory.decodeResource(getResources(), explodesID[i]);
27       }
28       for(EnemyPlane ep : enemyPlanes){                          //为敌方飞机初始化图片
29           if(ep.type == 1){                                      //当飞机类型为 1 时
30               ep.bitmap = enemyPlane1;
31           }else if(ep.type == 2){                                //当飞机类型为 2 时
32               ep.bitmap = enemyPlane2;
33           }else if(ep.type == 3){                                //当飞机类型为 3 时
34               ep.bitmap = enemyPlane3;
35           }else if(ep.type == 4){                                //当飞机类型为 4 时
36               ep.bitmap = enemyPlane4;
37           }
38       }
39       for(Life l : lifes){                                       //为补血道具初始化图片
40           l.bitmap = life;                                       //得到图片引用
41       }
42       for(ChangeBullet cb : changeBullets){                      //为换枪道具初始化图片
43           cb.bitmap = changebullet;                              //得到图片引用
44       }}
```

- 第 2～4 行将加载界面中表示进度的整数加 20，使当前前台正在显示的加载界面中的进度条向后走 20%。
- 第 8～13 行先初始化一张较大的游戏界面背景图片，然后根据之前创建数组的长度将大图片进行切割，并将切割后的图片存储到图片数组 battlebacks 中，然后将大图释放掉。这样在之后需

要绘制时，从数组中取出需要的若干幅小图片组合成需要的图片即可。

- 第 14～24 行初始化敌机、血条以及数字等图片资源，而第 25～27 行初始化的是爆炸的动画帧。
- 第 28～38 行根据敌方飞机的类型初始化相应的图片，第 39～44 行则初始化补血道具以及换枪道具的图片资源。

提示 将较大的图片分割成若干小图片，在绘制时只需简单的计算即可得到需要绘制的图片，这样会大大提升游戏的性能，减轻系统绘制图片的负担，避免绘制很多不需要的部分。

12.7.6 绘制方法 onDraw 的实现

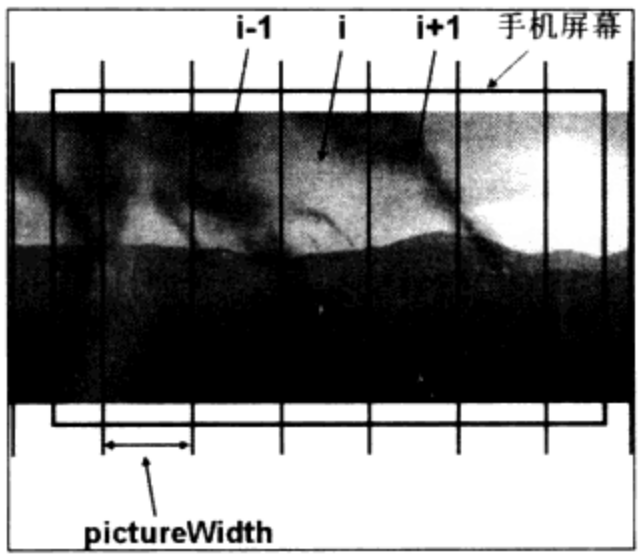
接下来将对该游戏最主要的方法——onDraw 进行介绍，该方法负责绘制游戏界面所有的信息，在方法中需要先将主要绘制物体的坐标复制出来，以防发生同步问题导致屏幕不流畅，然后再根据坐标进行绘制，其代码如下。

```
1    public void onDraw(Canvas canvas){                                  //自己写的绘制方法,并非重写的
2         int backGroundIX=this.backGroundIX;                            //将背景图片的 x 坐标值复制
3         int i=this.i;                                                  //复制 i 值
4         int cloudX = this.cloudX;                                      //复制云彩的坐标值
5         //解决 i 左侧的问题
6         if(backGroundIX>0){                                            //当背景图片的坐标大于 0 时
7              int n=(backGroundIX/ConstantUtil.pictureWidth)+
8                   ((backGroundIX%ConstantUtil.pictureWidth==0)?0:1);//计算 i 左面有几幅图
9              for(int j=1;j<=n;j++){                                   //循环绘制图片
10                  canvas.drawBitmap(
11                            battlebacks[(i-j+ConstantUtil.pictureCount)%
                              ConstantUtil.pictureCount],
12                       backGroundIX-ConstantUtil.pictureWidth*j, //计算出 x 坐标
13                  ConstantUtil.top,                                  //y 坐标
14                       paint                                         //画笔
15                 );
16             }
17        }
18        //解决 i 自己
19        canvas.drawBitmap(battlebacks[i], backGroundIX, ConstantUtil.top, paint);
20        //解决 i 右侧的问题
21        if(backGroundIX<ConstantUtil.screenWidth-ConstantUtil.pictureWidth){
22             int k=ConstantUtil.screenWidth-(backGroundIX+ConstantUtil.pictureWidth);
23             int n=(k/ConstantUtil.pictureWidth)+((k%ConstantUtil.pictureWidth==0)?0:1);
                                                                        //计算 i 右面有几幅图
24             for(int j=1;j<=n;j++){
25                  canvas.drawBitmap(                                 //绘制图片
26                            battlebacks[(i+j)%ConstantUtil.pictureCount],
27                            backGroundIX+ConstantUtil.pictureWidth*j, //x 坐标
28                            ConstantUtil.top,                          //y 坐标
29                            paint                                      //画笔
30                  );
31             }
32        }
33        plane.draw(canvas);                                           //画玩家飞机
34        if(status == 1 || status == 3){                               //游戏中时,关口中时
35             try{                                                     //绘制我方子弹
36                  for(Bullet b:goodBullets){                          //循环所有我方子弹
```

```
37                      b.draw(canvas);                          //绘制
38                  }
39              }catch(Exception e){ }                           //捕获异常
40              try{                                             //绘制敌方飞机
41                  for(EnemyPlane ep:enemyPlanes){              //循环所有敌方飞机
42                      if(ep.status == true){
43                          ep.draw(canvas);                     //绘制
44                      }
45                  }
46              }catch(Exception e){}                            //捕获异常
47              try{                                             //绘制敌方子弹
48                  for(Bullet b:badBullets){                    //循环所有敌方子弹
49                      b.draw(canvas);                          //绘制
50                  }
51              }catch(Exception e){}                            //捕获异常信息
52              try{
53                  for(ChangeBullet cb : changeBullets){        //绘制换枪道具
54                      if(cb.status == true){
55                          cb.draw(canvas);                     //绘制
56                      }
57                  }
58              }catch(Exception e){}                            //捕获异常
59              try{                                             //绘制补血道具
60                  for(Life l : lifes){                         //对所有补血道具循环
61                      if(l.status == true){
62                          l.draw(canvas);                      //绘制
63                      }
64                  }
65              }catch(Exception e){}                            //捕获异常
66              try{                                             //绘制爆炸
67                  for(Explode e : explodeList){                //循环所有爆炸
68                      e.draw(canvas);                          //绘制爆炸
69                  }
70              }catch(Exception e){}                            //捕获异常信息
71          }
72          if(cloudX>-cloud.getWidth() && cloudX<ConstantUtil.screenWidth){//绘制云彩
73              canvas.drawBitmap(cloud, cloudX, ConstantUtil.top, paint);//绘制云彩图片
74          }
75          canvas.drawBitmap(hullBackground, 0, 10, paint);     //画生命处的背景
76          for(int j=0; j<((5-plane.life)<0?5:plane.life); j++){ //绘制表示生命的小数条
77              canvas.drawBitmap(hull, 95+11*j, 13, paint);     //绘制图片
78          }
79          String timeStr = gameThread.touchTime/10+"";         //转换成字符串
80          for(int c=0;c<timeStr.length();c++){                 //循环绘制时间
81              int tempScore=timeStr.charAt(c)-'0';
82              canvas.drawBitmap(number[tempScore], 350+c*22, 12, paint);//绘制图片
83          }
84          canvas.drawRect(0, 0, 480, 10, paint);               //绘制屏幕上边沿
85          canvas.drawRect(0, 310, 480, 320, paint);            //绘制屏幕下边沿
86      }
```

- 第 6～32 行负责绘制游戏界面的滚动背景，为了提高游戏的性能，在初始化背景图片时，将大的背景图片分割成很多份小图片存放在图片数组中，此时根据当前 i 的值、小图片的宽度以及屏幕的宽度，自动计算 i 左边有多少幅小图片以及右边有多少幅小图片，并分别从数组中取出图片进行绘制，如图 12-16 所示。
- 第 35～39 行表示当游戏状态处于正常游戏中或者关口时，对存放所有我方子弹的容器进行循环，分别进行绘制。

- 第 40～71 行同样是对存放需要绘制物体的容器进行循环，获取每个物体的引用，分别调用其自身的绘制方法进行绘制。
- 第 72～74 行先判断云彩的坐标，当需要绘制时才将云彩绘制到指定位置，第 75～78 行绘制屏幕左上角生命的相关信息。
- 第 79～83 行绘制时间图片，先将 touchTime 转换成字符串，然后通过对字符串进行循环取出每个字符，将其减去字符 0 便得到字符所表示的数字，然后根据该数字取出对应的数字图片进行绘制。



▲ 图 12-16　绘制切割后的图片

12.8 游戏的优化与改进

到此本游戏的功能基本开发完成，但是仍然有一些地方值得提升，下面列出部分可以提升的地方，有兴趣的读者可以对该游戏进行进一步开发。

- 成绩的排行

如果为该游戏添加上成绩排行功能，并将玩家的最好成绩保存下来，会提高游戏的可玩性。

- 关卡的设计

前面已经介绍过，如果在设计游戏中敌方飞机、道具的运动轨迹时，使用地图设计器会大幅提高开发效率，为了演示简单，且前面的章节中已经介绍过地图设计的开发思路，所以，本游戏并没有开发飞机游戏的地图设计器，有兴趣的读者可自行开发简单的地图设计器，优化游戏的地图设计。

- 声音的设置

该游戏在默认情况下会播放声音，并且游戏刚启动的欢迎动画中的声音是不能关闭的，在真实的游戏开发中，一般是在游戏启动后首先出现简单的声音设置界面，然后根据玩家选择的设置进行欢迎动画的播放，该技术会在后面章节的案例中介绍，此处就不再详述。

第 13 章　棋牌游戏——中国象棋人机对弈

随着生活节奏的加快，智能手机的发展，人们对于在智能手机上进行娱乐的机会越来越多，这在一定程度上也推动了手机游戏的发展，而棋牌游戏又是手机游戏中趣味性较强的一类。

中国象棋是棋牌类游戏中最受欢迎的游戏之一，其可玩性强且操作简单。本章通过介绍 2D 中国象棋在 Android 平台上的设计与实现，使读者了解 2D 棋牌类游戏的开发过程。

13.1 游戏的背景及功能概述

本节将对中国象棋游戏进行简单的介绍，通过本节的学习，可以使读者对中国象棋游戏有整体了解，知道本章将要开发的案例的具体功能及效果。

13.1.1　背景概述

中国象棋在中国有着悠久的历史，属于二人对抗性游戏的一种，由于用具简单，趣味性强，成为广泛的棋艺活动。中国象棋是我国正式开展的 78 个体育运动项目之一。

本书中开发的中国象棋，以逼真的视角，简便的操作方式和与计算机对弈模式使玩家更能够体验到中国象棋的智慧与乐趣。

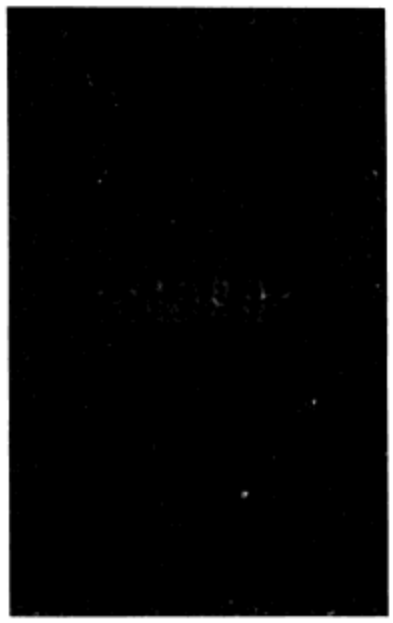
13.1.2　功能介绍

本节将对要介绍的中国象棋单机对弈游戏的功能以及操作方法进行简单的介绍，使读者对该游戏有一定的感性认识，其具体的操作步骤如下。

（1）运行该游戏，首先进入的是欢迎界面，效果如图 13-1 和图 13-2 所示。当欢迎界面运行完毕进入的是游戏界面，刚进入时，游戏为暂停状态，此状态下，玩家和计算机均不可走棋，如图 13-3 所示。

（2）在暂停界面下，还可以点击难度来选择计算机的聪明度，其效果如图 13-4 所示，在该界面下，单击开始按钮时，即可进入玩家操控棋子对弈界面，如图 13-5 所示，其走棋方式如图 13-6 和图 13-7 所示。在该情况下，玩家可以单击悔棋按钮来悔棋，还可以单击新局来重开一局。

（3）当玩家输了，或者计算机输了时，会出玩家输界面或者赢界面，其效果如图 13-8 和图 13-9 所示。在此两个界面下，均可单击确定按钮，再次进入游戏的暂停界面，重新开始下一局。

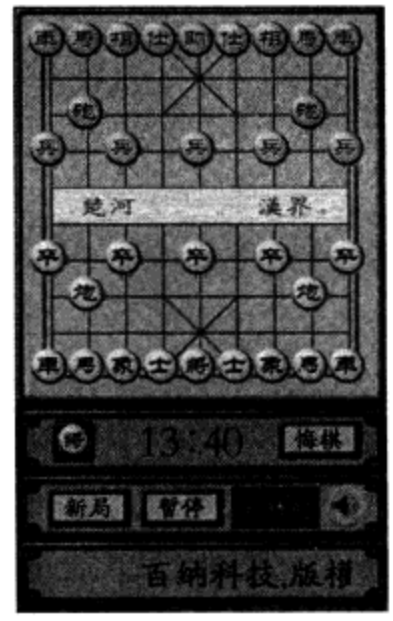


▲图 13-1　欢迎界面 1

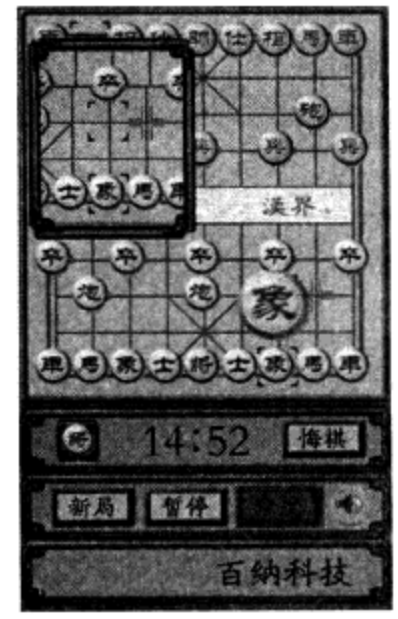
▲图 13-2　欢迎界面 2

▲图 13-3　暂停状态界面

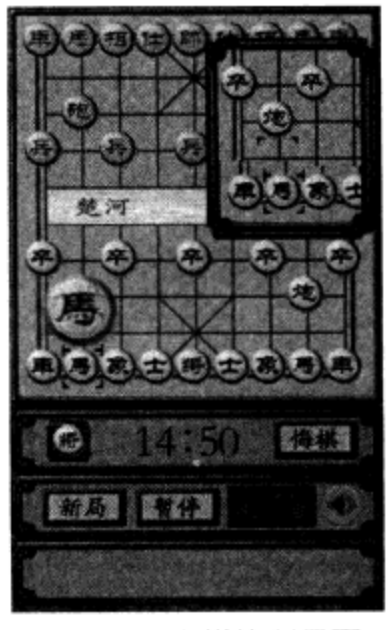
▲图 13-4　难度选择界面



▲图 13-5　对弈界面



▲图 13-6　走棋操控界面 1



▲图 13-7　走棋控制界面 2

▲图 13-8　输界面

▲图 13-9　赢界面

13.2 游戏的策划及准备工作

本节将介绍游戏的整体策划及开发前的一些准备工作，本节所介绍的知识在真实的游戏开发中作用往往是很大的。做好准备工作，在后面的开发过程中才能得以顺利进行。

13.2.1 游戏的策划

接下来对游戏的策划进行简单介绍，该游戏的策划如下。

- 游戏类型

该游戏属于棋牌类游戏中的一种，采用简单的手机操作来实现下棋，运用智能手机的触摸功能，并通过程序自行搜索下棋步骤来实现人机对弈。

- 运行目标平台

游戏目标平台为 Android 2.1，向下兼容 Android 2.0、Android 1.6。

- 目标受众

由于中国象棋游戏操作简单，思维灵活性强，广受人们的喜爱。本游戏以手机为载体，更受到老年人、青年人和儿童等各类人群的喜爱。2D 界面的绚丽实现更是符合各类时尚人士的口味。

- 操作方式

该游戏采用屏幕触摸操控棋子和界面。玩家可以使用手指摸动屏幕来实现各种功能，游戏中玩家触摸要走的棋子，然后拖动到要走到的地方，如果符合规则，则可实现走棋，程序会自行判断输赢。

- 呈现技术

本游戏采用 Android 手机中的 2D 呈现技术，界面绚丽，符合各类人士的美观需求。

13.2.2 Android 平台下游戏的准备工作

下面需要做一些开发前的准备工作，包括搜集和制作图片、声音等，其详细步骤如下。

（1）首先介绍的是本游戏中所用到的图片资源，将所有图片资源都放在项目文件下的 res/drawable-nodpi 文件夹下，其详细情况如表 13-1 所示。

表 13-1　　图片清单

图 片 名	大小（KB）	像素（w×h）	用　　途
ba.png	6.44	56×56	黑方的士
bb.png	6.95	56×56	黑方的象
bc.png	6.78	56×56	黑方的炮
br.png	6.59	56×56	黑方的车
bp.png	6.55	56×56	黑方的卒
bk.png	6.82	56×56	黑方的将
bn.png	6.64	56×56	黑方的马

续表

图 片 名	大小（KB）	像素（w×h）	用　　途
ra.png	6.71	56×56	红方的士
rb.png	6.78	56×56	红方的象
rc.png	6.74	56×56	红方的炮
rr.png	6.77	56×56	红方的车
rp.png	6.76	56×56	红方的兵
rk.png	6.82	56×56	红方的将
rn.png	6.88	56×56	红方的马
paotai1.png	3.51	56×56	兵的左临边坐
paotai2.png	3.54	56×56	兵的右临边坐
Paotai3.png	4.07	56×56	炮座
selected.png	2.89	56×56	标识计算机走棋的位置
chuhe.jpg	20.1	380×46	楚河
baina.jpg	18.8	480×320	欢迎界面 1
bnkjs.jpg	16.0	480×320	欢迎界面 2
beijintu.jpg	99.5	480×800	背景图
floor2.jpg	28.6	480×528	棋盘
guanggao1.jpg	60.7	480×528	广告图片 1
floor.jpg	286	480×528	广告图片 2
beijing.jpg	19.6	480×90	各按钮的背景框
beijing3.jpg	21.1	480×90	难度选择条的背景框
beijingkuangtu.png	32.5	216×266	视野窗口框
caidanxiaqifang.png	8.48	55×55	指示何方走棋的背景框
chonwan.jpg	20.4	114×56	新局按钮
huiqi.jpg	21.6	114×56	悔棋按钮
kaiqishengy.png	8.32	55×55	开启声音状态下的声音按钮
guanbishengy.png	8.54	55×55	关闭声音状态下的声音按钮
nanduxuanz.jpg	21.3	114×56	难度选择为可用状态下
nonandu.jpg	16.1	114×56	难度选择为不可用状态下
start.jpg	20.7	114×56	开始按钮
zhanting.jpg	20.5	114×56	暂停按钮
yingjiemian.png	37.9	255×255	赢界面下的背景图
shujiemian.png	37.8	255×255	输界面下的背景图
shuyingfugai.png	5.17	480×800	输赢界面下的覆盖半透明图

续表

图 片 名	大小（KB）	像素（w×h）	用　　途
queding.png	20.6	114×56	确定按钮
guanggao2.png	8.41	480×90	文字广告条
d0.png	3.26	55×55	数字 0
d1.png	2.96	55×55	数字 1
d2.png	3.25	55×55	数字 2
d3.png	3.27	55×55	数字 3
d4.png	3.04	55×55	数字 4
d5.png	3.19	55×55	数字 5
d6.png	3.30	55×55	数字 6
d7.png	3.11	55×55	数字 7
d8.png	3.39	55×55	数字 8
d9.png	3.31	55×55	数字 9
dunhao.png	2.96	55×55	顿号

（2）本游戏用到的声音资源，将声音资源放在项目目录中的 res/raw 文件夹下，其详细情况如表 13-2 所示。

表 13-2　　声音清单

声音文件名	大小（KB）	格　　式	用　　途
noxiaqi.mp3	7.37	Mp3	将军的声音
dong.mp3	4.64	Mp3	走棋声音
win.mp3	72.4	Mp3	赢了的声音
loss.mp3	12.2	Mp3	输了的声音

13.3 游戏的架构

本节将对该游戏的架构进行介绍，在整体上对游戏的设计进行详细介绍，使读者能够更容易地理解后面的章节。

13.3.1　各个类简要介绍

为了让读者能够更好地理解各个类的作用，下面将其分成 5 部分进行介绍，而各个类的详细代码将在后面的章节中相继给出。

1. 公共类

Activity 的实现类 Chess_DJB_Activity，该类是通过扩展 Activity 得到的，是整个游戏的控制器，

也是整个游戏的程序入口。

2. 游戏界面相关类 WelcomeView

该类为游戏刚刚进入时的欢迎界面，采用渐变技术实现动画的效果，在欢迎界面下不对任何键盘操作和屏幕触控进行处理。

3. 界面相关类

- 游戏界面类 GameView

此类为游戏中界面的实现类，复杂绘制游戏场景中的所有物体，如棋盘、棋子，各个操作按钮，各种效果图等。

- 游戏界面相关常量类 ViewConstant

该类为整个游戏提供一些常量，以及一些相关的参数。为游戏开发后期的调试减轻工作量。

4. 走法引擎相关类

- 常量类 Constant

该类为整个的走法引擎提供相关必要的参数，如子力价值、棋盘状况、走棋规则数组等。

- 方法类 Chess_LoadUtil

该类为走法引擎提供一些必要的方法。

- 走法引擎的主类 LoadUtil

该类为计算机走棋的主要计算类，根据当前棋盘搜索出最好的下棋步骤。

- 比较器类 MyComparator

该类为在计算机进行搜索时，提供排序时的比较器。

- 走棋步骤类 StackplayChess

该类为在进行对弈时，将玩家和计算机的走棋步骤放入栈中。当玩家需要悔棋和判断重复局面时用。

13.3.2 游戏框架简介

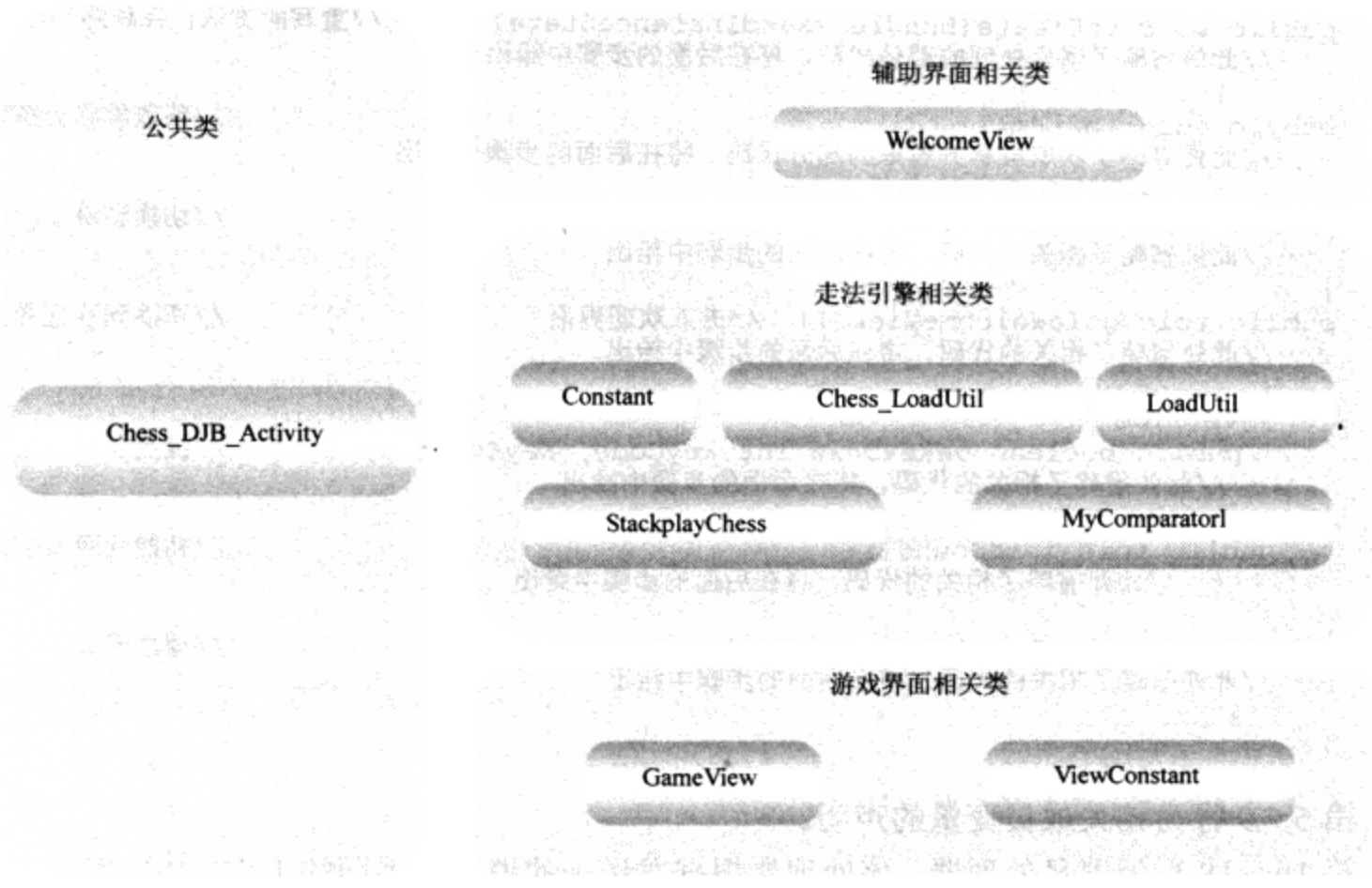
前面已经对该游戏中所用到的类进行了简单介绍，可能读者并没有理解游戏的架构以及游戏的运行过程，本节将从游戏的整体架构上进行介绍，使读者对本游戏有更深的理解，其框架如图 13-10 所示。

接下来按照程序运行的顺序介绍各个类的作用以及整体的运行框架，其详细步骤如下。

（1）启动游戏，首先进入的是欢迎界面 WelcomeView 类，当欢迎界面播放完毕，即可进入游戏界面 GameView 类，进入后，游戏界面出于暂停状态，玩家可以点击开始直接进行游戏，也可点击难度按钮进行对难度的选择，然后点击开始按钮即可开始游戏。

（2）在开始游戏后，玩家可触摸要下的棋子，然后拖动到要下到的地方，如果符合规则，则可实现走棋，玩家下完棋后，轮到计算机走棋，利用 LoadUtil 类搜索出走棋步骤，然后走棋即可。程序自行判断输赢。对双方的思考时间也都有限制，超过思考时间还没走棋则判输。

（3）在游戏开始界面下，如果玩家赢了，则跳到赢界面，而如果输了则会跳到输界面。在此两个界面下，均可点击确定按钮，回到游戏最初的状态，游戏界面下的暂停界面。



▲图 13-10　框架图

13.4 主控制类——Chess_DJB_Activity

从本节开始将正式进入游戏的开发过程，首先介绍的是游戏的控制器 GJXQActivity 类，该类的主要作用是，在适当的时间初始化相应的界面，并根据其他界面发送回来的消息切换到用户需要的界面。其开发的详细步骤如下。

（1）先介绍 Chess_DJB_Activity 的主要框架，其详细代码如下。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 Chess_DJB_Activity.java。

```
1    package com.bn.chess;
2    ……//此处省略了部分类和包的引入代码，读者可自行查看光盘的源代码
3    enum WhichView{WELCOME_VIEW,GAME_VIEW};                    //标识当前所在界面的枚举
4    public class Chess_DJB_Activity extends Activity {
5         GameView gameView;                                    //游戏界面的引用
6         WhichView wv;                                         //枚举变量
7         WelcomeView wvv;                                      //欢迎界面类的引用
8         SoundPool soundPool;                                  //声音池
9         HashMap<Integer, Integer> soundPoolMap; //声音池中声音 ID 与自定义声音 ID 的 Map
10        Handler hd=new Handler(){                             //消息处理器
11             @Override
12             public void handleMessage(Message msg) {              //消息处理方法
13                  switch(msg.what) {
14                  case 0:
```

```
15                          wv=WhichView.GAME_VIEW;
16                          goToGameView();                          //切换到游戏界面
17                          break;
18                  }}};
19      @Override
20  public void onCreate(Bundle savedInstanceState) {     //重写的方法，在程序启动时调用
21  ……//此处省略了消息处理的部分代码，将在后面的步骤中给出
22          }
23  public void initPm()  {                                          //获取屏幕分辨率的方法
24  ……//此处省略了获取屏幕分辨率的部分代码，将在后面的步骤中给出
25      }
26  public void goToGameView(){                                      //切换到游戏界面的方法
27  ……//此处省略了相关的代码，将在后面的步骤中给出
28  }
29  public void goToWelcomeView(){  //进入欢迎界面                   //切换到欢迎界面的方法
30  ……//此处省略了相关的代码，将在后面的步骤中给出
31      }
32      @Override
33        public boolean onKeyDown(int keyCode, KeyEvent e) {        //键盘监听方法
34     ……//此处省略了相关的代码，将在后面的步骤中给出
35      }
36      public void initSound(){                                     //初始化声音方法
37        ……//此处省略了相关的代码，将在后面的步骤中给出
38      }
39  public void playSound(int sound, int loop) {                     //播放声音
40  ……//此处省略了相关的代码，将在后面的步骤中给出
41        }
42  }
```

- 第 5～9 行为相关成员变量的声明。
- 第 10～18 行为消息处理器。该处理器根据发送来的消息做不同的工作，如当发送过来的消息为 0 时，进入游戏界面。
- 第 19～22 行为重写的 onCreate 方法，该方法在程序运行时首先被调用。
- 第 23～25 行为获取屏幕分辨率的方法，并在此时计算整个游戏时的图片缩放比例，界面相关参数等。

（2）接下来介绍的是 Chess_DJB_Activity 类中的 onCreate 方法，其详细代码如下。

```
1     @Override
2        public void onCreate(Bundle savedInstanceState) {
3           super.onCreate(savedInstanceState);
4           requestWindowFeature(Window.FEATURE_NO_TITLE);          //设置全屏显示
5           getWindow().setFlags(
6              WindowManager.LayoutParams.FLAG_FULLSCREEN ,
7            WindowManager.LayoutParams.FLAG_FULLSCREEN );          //强制竖屏
8              setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
9           setVolumeControlStream(AudioManager.STREAM_MUSIC);//游戏过程中只允许多媒体音量
10          initPm();                                               //调整屏幕分辨率
11          initSound();                                            //初始化声音
12          goToWelcomeView();}                                     //切换到欢迎界面
```

（3）然后介绍的是 Chess_DJB_Activity 中的 initPm 方法，其详细代码如下。

```
1     public void initPm() {                                        //获取屏幕分辨率
2           DisplayMetrics dm=new DisplayMetrics();
3           getWindowManager().getDefaultDisplay().getMetrics(dm);
4           int tempHeight=(int) (height=dm.heightPixels);          //得到当前屏幕的高度
5           int tempWidth=(int) (width=dm.widthPixels);             //得到当前屏幕的宽度
```

```
6           if(tempHeight>tempWidth) {                          //如果高度大于宽度
7               height=tempHeight;                              //则宽带就为宽度，高度为高度
8               width=tempWidth;
9           } else {                                            //否则说明当前屏幕被转为横屏
10              height=tempWidth;                               //需要矫正
11              width=tempHeight;
12          }
13              float zoomx=width/480;                          //得到横向比例，因为最初状态的
14              float zoomy=height/800;                         //分辨率为 480*800，以此为标准
15              if(zoomx>zoomy){                                //如果横向大于纵向
16                   xZoom=yZoom=zoomy;                         //则缩放比按照纵向
17              }else{
18                   xZoom=yZoom=zoomx;                         //否则按照横向
19              }
20              sXtart=(width-480*xZoom)/2;                     //得到游戏界面的左上方坐标
21              sYtart=(height-800*yZoom)/2;
22              initChessViewFinal();                           //初始化相关变量
23      }
```

- 第 2～5 行为得到屏幕分辨率。
- 第 6～12 行为根据当前屏幕分辨率进行矫正，防止当前手机状态为横向从而得到不正确的缩分比。
- 第 13～23 行为根据屏幕分辨率计算界面缩放比，并对各界面相关变量进行初始化。

（4）接下来介绍的是 Chess_DJB_Activity 中的 goToGameView 方法，其详细代码如下。

```
1    public void goToGameView(){
2         gameView=new GameView(Chess_DJB_Activity.this);      //创建游戏界面对象
3         setContentView(gameView);                            //切换到游戏界面
4         wv=WhichView.GAME_VIEW;                              //标识当前界面为游戏界面
5       }
```

（5）Chess_DJB_Activity 中的 goToWelcomeView 方法，其详细代码如下。

```
1    public void goToWelcomeView(){
2         if(wvv==null){
3              wvv=new WelcomeView(this);                      //创建欢迎界面对象
4         }
5         setContentView(wvv);                                 //切换到欢迎界面
6         wv=WhichView.WELCOME_VIEW;                           //标识当前界面为欢迎界面
7       }
```

（6）Chess_DJB_Activity 中的 onKeyDown 方法，其详细代码如下。

```
1    @Override
2         public boolean onKeyDown(int keyCode, KeyEvent e) { //键盘监听方法
3         if(keyCode!=4)  {                              //如果不是返回键，则不作处理
4              return true;
5         }
6         if(wv==WhichView.WELCOME_VIEW) {               //如果当前界面为欢迎界面，则不作处理
7              return true;
8         }
9         if(wv==WhichView.GAME_VIEW){                   //如果当前界面为游戏界面
10             gameView.threadFlag=false;                 //线程结束
11             System.exit(0);                            //系统退出
12             return true;
13        }
14        System.exit(0);                                 //否则系统退出
15        return true;
```

```
16        }
```

（7）Chess_DJB_Activity 中的 initSound 方法，其详细代码如下。

```
1     public void initSound(){                                          //初始化声音方法
2               soundPool = new SoundPool(4, AudioManager.STREAM_MUSIC, 100); //声音池
3             soundPoolMap = new HashMap<Integer, Integer>();  //创建存放声音 ID 的 Map
4             soundPoolMap.put(1, soundPool.load(this, R.raw.noxiaqi, 1)); //将军声音
5             soundPoolMap.put(2, soundPool.load(this, R.raw.dong, 1)); //玩家走棋
6             soundPoolMap.put(4, soundPool.load(this, R.raw.win, 1)); //赢了
7             soundPoolMap.put(5, soundPool.load(this, R.raw.loss, 1)); //输了
8         }
```

（8）最后介绍的是 Chess_DJB_Activity 中的 playSound 方法，其详细代码如下。

```
1      public void playSound(int sound, int loop) {
2             AudioManager mgr = (AudioManager)this.getSystemService(Context.AUDIO_SERVICE);
3             float streamVolumeCurrent = mgr.getStreamVolume(AudioManager.STREAM_MUSIC);
4             float streamVolumeMax = mgr.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
5             float volume = streamVolumeCurrent / streamVolumeMax;
6             soundPool.play (
7               soundPoolMap.get(sound),                              //声音资源 id
8               volume,                                               //左声道音量
9               volume,                                               //右声道音量
10              1,                                                    //优先级
11              loop,                                                 //循环次数，-1 表示永远循环
12              0.5f                                                  //回放速度 0.5f～2.0f 之间
13            );}
```

13.5 辅助界面相关类

前一节介绍了主控制类 Chess_DJB_Activity，本节将对该游戏的辅助界面相关类进行介绍，该游戏的辅助界面类主要是欢迎界面 WelcomeView 类，本节将对该游戏的第一个界面，即欢迎界面进行介绍，该界面是在游戏启动后出现的第一个界面，其详细步骤如下。

（1）首先介绍的是欢迎界面的框架，然后再介绍其详细代码。其框架代码如下。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 WelcomeView.java。

```
1     package com.bn. chess;
2     ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
3     public class WelcomeView extends SurfaceView
4     implements SurfaceHolder.Callback {                               //实现生命周期回调接口
5          Chess_DJB_Activity activity;                                 //Activity 的引用
6          Paint paint;                                                 //画笔
7          int currentAlpha=0;                                          //当前的不透明值
8          int screenWidth=480;                                         //屏幕宽度
9          int screenHeight=320;                                        //屏幕高度
10         int sleepSpan=50;                                            //动画的时延 ms
11         Bitmap[] logos=new Bitmap[2];                                //Logo 图片数组
12         Bitmap currentLogo;                                          //当前 Logo 图片引用
13         int currentX;                                                //图片位置
14         int currentY;
15         public WelcomeView(Chess_DJB_Activity activity){
16              super(activity);
17              this.activity = activity;
18              this.getHolder().addCallback(this);    //设置生命周期回调接口的实现者
19              paint = new Paint();                                    //创建画笔
```

```
20              paint.setAntiAlias(true);                          //打开抗锯齿
21    logos[0]=scaleToFit(BitmapFactory.decodeResource(activity.getResources(),
      R.drawable.baina),xZoom);
22    logos[1]=scaleToFit(BitmapFactory.decodeResource(activity.getResources(),
      R.drawable.bnkjs),xZoom);
23          }
24          public void onDraw(Canvas canvas){
25              paint.setColor(Color.BLACK);                       //设置画笔颜色
26              paint.setAlpha(255);                               //设置不透明度为 255
27              canvas.drawRect(0, 0, width, height, paint);       //绘制黑填充矩形用以清除背景
28              if(currentLogo==null)return;                       //进行平面贴图
29              paint.setAlpha(currentAlpha);
30              canvas.drawBitmap(currentLogo, currentX, currentY, paint);
31          }
32          public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3){
33          }
34          public void surfaceCreated(SurfaceHolder holder) {  //创建时被调用
35              new Thread(){
36              ……//此处省略了该线程的部分代码，将在后面的步骤中给出
37              }.start();
38
39          }
40          public void surfaceDestroyed(SurfaceHolder arg0)
41          {                                                          //销毁时被调用
42          }}
```

- 第 5～14 行为声明了一些需要用到的成员变量。
- 第 15～23 行为构造器，其中第 19 行创建了画笔，第 21～22 行加载了欢迎界面上的两幅图片，并进行等比例的缩放，以适应不同的屏幕分辨率。
- 第 24～31 行为绘制函数，依次绘制两张图片，用一个线程来按时改变其不透明度的参数来实现渐变的效果。第一张图片渐变完毕，再采用同样的方式对第二张图片进行同样的操作。
- 第 34～39 行，此函数为在该界面创建的时候被调用，此时该函数启用一个线程，来实现渐变的效果。

（2）然后介绍的是在 WelcomeView 类中的 surfaceCreated 方法中省略的部分代码，其详细代码如下。

```
1     new Thread(){
2          public void run(){
3               for(Bitmap bm:logos){                                  //对数组进行循环
4                    currentLogo=bm;                                   //当前图片的引用
5                    currentX=(int) (width/2-bm.getWidth()/2);         //图片位置
6                    currentY=(int) (height/2-bm.getHeight()/2);;
7                    for(int i=255;i>-10;i=i-10){              //动态更改图片的不透明度
8                         currentAlpha=i;                      //变换透明度，总的变化为 255 次
9                         if(currentAlpha<0) {                 //如果当前不透明度小于零
10                             currentAlpha=0;
11                   }                                                 //将不透明度设置为零
12                   SurfaceHoldermyholder=WelcomeView.this.getHolder();//获取回调接口
13                   Canvas canvas = myholder.lockCanvas();            //获取画布
14                     try{
15                        synchronized(myholder) {                     //同步
16                             onDraw(canvas);                         //进行绘制
17                        }}
18                   catch(Exception e){
19                        e.printStackTrace();                         //打印异常信息
20                     }
```

```
21                  finally{
22                      if(canvas!= null) {                          //如果当前画布不为空
23                          myholder.unlockCanvasAndPost(canvas);//解锁画布
24                      }}
25                  try{
26                      if(i==255) {                        //若是新图片，多等待一会儿
27                          Thread.sleep(1000);             //线程睡眠
28                      }
29                      Thread.sleep(sleepSpan);            //线程睡眠
30                  }
31                  catch(Exception e) {                    //抛出异常
32                      e.printStackTrace();
33                  }}}
34                  activity.hd.sendEmptyMessage(8);        //发送消息，开始加载棋子模型
35              }}.start();
```

- 该线程通过强制性 for 循环，第一次循环，对欢迎界面图片 1 进行不透明度的变化，每变化一次画一次，就达到了渐变的效果。当第一张图片播放完毕，线程等待一小段时间，切换到第二幅图片，通过同样的处理，就构成了整个欢迎界面的动态效果。详细情况请见注释。
- 第 13～17 行为绘制界面，首先锁定画布，然后调用 onDraw 函数进行界面绘制。

13.6 游戏界面相关类

前面已经介绍了欢迎界面，从本节开始将进入游戏开发最主要的环节，即游戏界面开发环节。本节先对整个游戏界面类进行介绍，然后再细节地介绍各个与游戏界面相关的类，逐步完成游戏界面的开发。

13.6.1 游戏界面绘制类 GameView

接下来将对整个游戏界面绘制类 MySurfaceView 进行介绍，首先介绍的是其整体框架，然后才进行详细代码的开发，整体框架的详细代码如下。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 GameView.java。

```
1     package com.bn.chess;
2     ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
3     public class GameView extends SurfaceView implements SurfaceHolder.Callback{
4     ……//省略了一些成员变量，将在后面的步骤中给出
5         public GameView(Context context) {                          //构造器
6             ……//此处省略了相关的代码，将在后面的步骤中给出
7         }
8         public void initArrays()  {                                 //得到当前棋盘数组
9         ……//此处省略了相关的代码，将在后面的步骤中给出
10        }
11        @Override
12        public void onDraw(Canvas canvas)    {                      //主绘制方法
13        ……//此处省略了相关的代码，将在后面的步骤中给出
14        }
16        public void onDrawWindowindow(Canvas canvas,float sXtart,float sYtart)
          { //绘制棋盘以及棋子等
17        ……//此处省略了相关的代码，将在后面的步骤中给出
18        }
19        public void onDrawWindowMenu(Canvas canvas,float sXtart,float sYtart){
                                                                      //绘制按钮选项等
```

```
20          ……//此处省略了相关的代码，将在后面的步骤中给出
21          }
22          public void initColer()    {                                    //初始化颜色数组
23          ……//此处省略了相关的代码，将在后面的步骤中给出
24          }
25          public void drawScoreStr(Canvas canvas,String s,float width,float height) {
                                                                            //绘制字符串方法
26          ……//此处省略了相关的代码，将在后面的步骤中给出
27          }
28          public void initBitmap(){                                       //初始化图片
29          ……//此处省略了相关的代码，将在后面的步骤中给出
30          }
31          @Override
32          public void surfaceChanged(SurfaceHolder holder, int format, int width,
33                     int height) {}
34          @Override
35          public void surfaceCreated(SurfaceHolder holder) {
36                ……//此处省略了相关的代码，将在后面的步骤中给出
37          }
38          public void newThread()    {                                    //新建线程方法
39               ……//此处省略了相关的代码，将在后面的步骤中给出
40          }
41          @Override
42          public void surfaceDestroyed(SurfaceHolder holder) {}
43          @Override
44          public boolean onTouchEvent(MotionEvent e){                     //屏幕触摸监听回调方法
45          ……//此处省略了相关的代码，将在后面的步骤中给出
46          }
47          public void onDrawcanvas(){                                     //绘制方法
48          ……//此处省略了相关的代码，将在后面的步骤中给出
49          }
50    }
```

- 第 12～14 行为主要的绘制方法，该方法为主绘制方法，是这个界面的核心方法。
- 第 16～18 行为绘制棋盘和棋子的方法。
- 第 19～21 行为绘制棋盘底下的选项按钮的方法。
- 第 38～40 行为新建一个线程的方法，调用此方法即可新建一个线程，来进行刷帧和记录时间，变化滚动广告条的坐标等。
- 第 44～46 行为屏幕触摸监听的回调方法，该方法最为重要，详细情况在后面的步骤中给出。

介绍完了绘制类的整体框架，下面将分别给出各个部分的详细代码，其步骤如下。

（1）首先介绍的是该类的成员变量，其详细代码如下。

```
1           Chess_DJB_Activity father;                    //主类 Chess_DJB_Activity 的引用
2           Bitmap[][] chessBitmap;                                         //象棋棋子图片
3           Bitmap chessZouQiflag;                                          //标志走棋的位置图片
4           Bitmap paotai;                                                  //炮台
5           Bitmap paotai2;                                                 //左边沿兵下的兵坐
6           Bitmap paotai1;                                                 //右边沿兵下的兵坐
7           Bitmap chuhe;                                                   //楚河图片
8           Paint paint;                                                    //画笔
9           Bitmap chessQipan2;                                             //棋盘边框
10          Bitmap huiqi;                                                   //悔棋按钮图片
11          Bitmap chonWan;                                                 //新局按钮
12          Bitmap[] iscore=new Bitmap[10];                                 //数字图
13          Bitmap dunhao;                                                  //顿号
14          Bitmap beijint;                                                 //背景图
15          Bitmap minueBeijint;                                            //菜单背景图
```

```
16        Bitmap ifPlayChess;                                    //标识走棋方的背景
17        Bitmap isPlaySound;                                    //开启声音按钮图片
18        Bitmap noPlaySound;                                    //关闭声音按钮
19        Bitmap start;                                          //开始按钮
20        Bitmap suspend;                                        //暂停按钮
21        Bitmap nandutiaoZ;                                     //难度按钮
22        Bitmap nonandutiaoZ;                                   //难度不可用时的按钮
23        Bitmap guanggao1[]=new Bitmap[2];                      //广告图片
24        Bitmap guanggao2;                                      //广告条 2
25        Bitmap winJiemian;                                     //赢界面
26        Bitmap loseJiemian;                                    //输界面
27        Bitmap fugaiTu;                                        //覆盖图
28        Bitmap queDinButton;                                   //确定按钮
29        Bitmap beijint3;                                       //难度选择拖动条背景
30        Bitmap beijint4;                                       //剪裁框背景
31        int[][] color=new int[20][3];                 //拖动条中的小方格的颜色数组
32        int length;                                   //难度数，就是拖动条的小方框数目
33        int huiqibushu=0;                                            //悔棋的步数
34        float guanggao2X=0;                                          //最下方广告条的 X 坐标
35        boolean playChessflag;                        //走棋方标志位，false 为黑方走棋
36        SurfaceHolder holder;                                        //画布
37        Canvas canvas;
38        int ucpcSquares[]=new int[256];               //代表棋子在棋盘中的位置的数组
39        Stack<StackplayChess> stack=new Stack<StackplayChess>();//存放走棋步骤的栈
40        float xMove;
41        float yMove;                                                 //触摸移动时的坐标
42        boolean cMfleg=true;                                         //触摸是否有效
43        boolean isFlage;                                             //是否为第一次走棋
44        boolean xzflag = false;                                      //是否为选中.
45        boolean threadFlag=true;                                     //线程是否运行
46        int xzgz = 0                                                 //选中的初始格子
47        boolean flag;                                                //是否为移动标志
48        boolean huiqiFlag=false;                                     //点击处为悔棋标志
49        boolean chonwanFlag=false;                                   //点击处为新局标志
50        boolean isnoNanDu=true;                                      //难度是否可选
51        boolean dianjiNanDu;                                         //点击处为难度
52        boolean dianjiXinJu;                                         //点击处为新局
53        boolean dianjiKaiShi;                                        //点击处为开始
54        boolean dianjishengyin;                                      //点击为声音区域
55        boolean dianjiJDT;                                           //点击处为拖动
56        boolean dianjiQueDing;                                       //点击处为确定按钮
57        boolean nanduBXZ;                                            //点击处为难度按钮区域
58        int bzcol;                                                   //点击的目标列
59        int bzrow;                                                   //点击的目标行
```

（2）接下来介绍的是游戏界面类的构造器，其详细代码如下。

```
1     public GameView(Context context) {
2             super(context);
3             this.father=(Chess_DJB_Activity)context;   //得到 Activity 的引用
4             this.getHolder().addCallback(this);        //设置生命周期回调接口的实现者
5             paint = new Paint();                                   //创建画笔
6             paint.setAntiAlias(true);                              //打开抗锯齿
7             isnoStart=false;                                       //一开始为暂停状态
8             length=nanduXS*4;                          //难度拖动条的长度为初始难度的 4 倍
9             initColer();                               //初始化难度滚动条颜色数组
10            LoadUtil.Startup();                                    //初始化棋子数组
11            initArrays();                              //初始化棋盘对应的棋子数组
12            initBitmap();                                          //初始化图片
13            LoadUtil.sdPlayer=0;                                   //先走棋方均为玩家
14            endTime=zTime;                                         //总时间
```

```
15        }
```

说明　该构造器主要对界面相关和走法引擎相关的成员变量进行初始化。

（3）然后介绍的是复制数组方法，其详细代码如下。

```
1    public void initArrays(){
2            for(int i=0;i<256;i++)     {
3                ucpcSquares[i]=LoadUtil.ucpcSquares[i];  //将数组复制到对应棋盘的数组
4            }}
```

说明　另外声明一个数组来记录棋盘上的棋子是因为，程序在搜索下棋步骤时会改变 LoadUtil.ucpcSquares 中的值，如果直接用该数组来进行棋盘绘制，其中会出现错乱，所以，每次在下棋完毕后均要进行数组复制。绘制时依此数组进行绘制棋盘上的棋子。

（4）接下来介绍的是主界面的主绘制方法 onDraw 方法，其详细代码如下。

```
1    @Override
2        public void onDraw(Canvas canvas){
3            canvas.drawColor(Color.argb(255, 0, 0, 0));          //清除背景
4            canvas.drawBitmap(beijint,sXtart,sYtart, null);      //绘制整体背景图
5            if(isnoStart){                                       //如果已经开始博弈了
6
     onDrawWindowindow(canvas,ViewConstant.sXtart,ViewConstant.sYtart);//绘制棋盘
7                if(flag){                                  //为拖着棋子移动状态，绘制剪切窗口
8                    float left=xMove>sXtart+5*xSpan?windowXstartLeft:windowXstartRight;
9                    float top=windowYstart;
10                   float right=left+windowWidth;
11                   float bottom=top+windowHeight;//计算窗口的大小及位置
12                   canvas.save();                                  //保护当前界面
13                   canvas.clipRect(new RectF(left,top,right,bottom));
                                                        //剪切需要裁减的窗口
14                   onDrawWindowindow(canvas,sXtartCk,sYtartCk);//再次绘制棋盘
17                   canvas.restore();                  //恢复剪裁，在此中间的会被剪裁掉
18                   canvas.drawBitmap(beijint4,left-6,top-6, null);
                                                        //绘制窗口的相框
19               }}else{                                //暂停状况下绘制广告图片
20               canvas.drawBitmap(guanggao1[(int) ((Math.abs(guanggao2X/40)%2))],
                 sXtart,sYtart, null);
21           }
22       onDrawWindowMenu(canvas,ViewConstant.sXtart,ViewConstant.sYtart);
                                                        //棋盘下面的各个选项按钮
23           if(yingJMflag){                            //如果是赢了
24               canvas.drawBitmap(fugaiTu,sXtart,sYtart, null);        //画覆盖图
25               canvas.drawBitmap(winJiemian,sXtart+2.5f*xSpan,sYtart+5f*ySpan,
                 null);//赢背景覆盖图
26               if(dianjiQueDing){                     //如果点击了确定按钮的区域
27                   canvas.drawBitmap(scaleToFit(queDinButton,1.2f), //放大绘制
28               sXtart+3.7f*xSpan,sYtart+8.3f*ySpan, null);
29               }else                                  //否则按原来比例绘制确定按钮
30               canvas.drawBitmap(queDinButton,sXtart+3.9f*xSpan,sYtart+8.5f*ySpan,
                 null);
31           }
32           else if(shuJMflag){                        //如果玩家输了
33               canvas.drawBitmap(fugaiTu,sXtart,sYtart, null);        //画覆盖图
34               canvas.drawBitmap(loseJiemian,sXtart+2.5f*xSpan,sYtart+5f*ySpan,
```

```
                null);//赢背景覆盖图
35              if(dianjiQueDing){                          //如果点击的是确定按钮的区域
36              canvas.drawBitmap(scaleToFit(queDinButton,1.2f)//绘制放大的确定按钮
37          ,sXtart+3.7f*xSpan,sYtart+8.3f*ySpan, null);
38          }else                                           //否则绘制原来比例的确定按钮
38              canvas.drawBitmap(queDinButton,sXtart+3.9f*xSpan,sYtart+8.5f*ySpan,
                null);//确定按钮
39          }}
```

- 第 5～19 行为，在对局开始的情况下，在棋盘区域绘制棋盘，否则绘制广告。其中第 7～18 行为当玩家触摸了一个棋子，拖着走的情况下，绘制其棋子被拖动到的地方周围的那一小块区域，即裁减窗口，使玩家更加清晰地看到界面。实现原理是，将棋盘重绘一次，只是在重画此次时，将不必要的部分裁减掉了，就只剩下该小窗口。
- 第 22 行为绘制棋盘底下的各个选项按钮。
- 第 23～31 行为当玩家赢时，绘制赢界面，该界面只是在原来的暂停状态界面上面覆盖了一层半透明的图片，在最上面绘制了赢界面窗口，再多加了一个确定按钮图片。确定按钮同样在点击时会放大绘制，达到点击按钮的效果。
- 第 32～39 行为当玩家输时，绘制输界面，其绘制方法和绘制赢界面是同样的。

（5）棋盘绘制方法 onDrawWindowindow 其详细代码如下。

```
1    public void onDrawWindowindow(Canvas canvas,float sXtart,float sYtart){
2        canvas.drawBitmap(chessQipan2,sXtart,sYtart, null);
3        paint.setColor(Color.RED);                          //设置画笔颜色
4        paint.setStrokeWidth(3);                            //设置线的粗细
5        for(int i=0;i<10;i++){                              //画棋盘横线
6        canvas.drawLine(xSpan+sXtart,ySpan+ySpan*i+sYtart,
7                        sXtart+xSpan*9, sYtart+ySpan+ySpan*i, paint);
8        }
9        for(int i=0;i<9;i++){                               //画棋盘竖线
10       canvas.drawLine(sXtart+xSpan+i*xSpan,sYtart+ySpan,
11                       sXtart+xSpan+xSpan*i, sYtart+ySpan*10, paint);}
12       canvas.drawLine(sXtart+xSpan*4,sYtart+ySpan, sXtart+xSpan*6, sYtart+ySpan*3,
         paint); //绘制九宫
13       canvas.drawLine(sXtart+xSpan*6,sYtart+ySpan, sXtart+xSpan*4, sYtart+ySpan*3,
         paint);    //绘制斜线
14       canvas.drawLine(sXtart+xSpan*4,sYtart+ySpan*8, sXtart+xSpan*6, sYtart+ySpan*10,
         paint);
15       canvas.drawLine(sXtart+xSpan*6,sYtart+ySpan*8, sXtart+xSpan*4, sYtart+ySpan*10,
         paint);
16       paint.setStrokeWidth(5);                            //设置线的粗细，绘制边框
17       canvas.drawLine(sXtart+0.8f*xSpan,sYtart+0.8f*ySpan, sXtart+9.2f*xSpan,
         sYtart+0.8f*ySpan, paint);
18       canvas.drawLine(sXtart+0.8f*xSpan,sYtart+0.8f*ySpan,sXtart+0.8f*xSpan,
         sYtart+10.2f*ySpan, paint);
19       canvas.drawLine(sXtart+9.2f*xSpan,sYtart+0.8f*ySpan,sXtart+9.2f*xSpan,
         sYtart+10.2f*ySpan, paint);
20       canvas.drawLine(sXtart+0.8f*xSpan,sYtart+10.2f*ySpan,sXtart+9.2f*xSpan,
         sYtart+10.2f*ySpan,paint);
21       canvas.drawBitmap(chuhe,sXtart+xSpan+1.8f,sYtart+5*ySpan+1.0f, null);
                                                             //绘制楚河
22       canvas.drawBitmap(paotai,sXtart+2*xSpan-chessR*0.86f,sYtart+3*ySpan-chessR*
         0.86f, null);//绘制
23       canvas.drawBitmap(paotai,sXtart+2*xSpan-chessR*0.86f,sYtart+8*ySpan-chessR*
         0.86f, null);//炮台
24       canvas.drawBitmap(paotai,sXtart+8*xSpan-chessR*0.86f,sYtart+3*ySpan-chessR*
```

```
        0.86f, null);
25      canvas.drawBitmap(paotai,sXtart+8*xSpan-chessR*0.86f,sYtart+8*ySpan-chessR*
        0.86f, null);
26      canvas.drawBitmap(paotai2,sXtart+1*xSpan-chessR*0.86f,sYtart+4*ySpan-
        chessR*0.86f, null);//绘制
27      canvas.drawBitmap(paotai,sXtart+3*xSpan-chessR*0.86f,sYtart+4*ySpan-chessR*
        0.86f, null);//兵坐
28      canvas.drawBitmap(paotai,sXtart+5*xSpan-chessR*0.86f,sYtart+4*ySpan-chessR*
        0.86f, null);
29      canvas.drawBitmap(paotai,sXtart+7*xSpan-chessR*0.86f,sYtart+4*ySpan-chessR*
        0.86f, null);
30      canvas.drawBitmap(paotai1,sXtart+9*xSpan-chessR*0.86f,sYtart+4*ySpan-
        chessR*0.86f, null);
31      canvas.drawBitmap(paotai2,sXtart+1*xSpan-chessR*0.86f,sYtart+7*ySpan-
        chessR*0.86f, null);
32      canvas.drawBitmap(paotai,sXtart+3*xSpan-chessR*0.86f,sYtart+7*ySpan-chessR*
        0.86f, null);
33      canvas.drawBitmap(paotai,sXtart+5*xSpan-chessR*0.86f,sYtart+7*ySpan-chessR*
        0.86f, null);
34      canvas.drawBitmap(paotai,sXtart+7*xSpan-chessR*0.86f,sYtart+7*ySpan-chessR*
        0.86f, null);
35      canvas.drawBitmap(paotai1,sXtart+9*xSpan-chessR*0.86f,sYtart+7*ySpan-
        chessR*0.86f, null);
36           for(int i=0;i<10;i++){                          //绘制棋子
37                for(int j=0;j<9;j++){
38                     if(ucpcSquares[(i+3)*16+j+3]!=0){    //如果不为零，说明有棋子
39      canvas.drawBitmap(chessBitmap[ucpcSquares[(i+3)*16+j+3]/16][
40      ucpcSquares[(i+3)*16+j+3]%8],sXtart+j*xSpan-chessR+xSpan,sYtart+i*ySpan-
        chessR+ySpan, null);
41                     }}}
42           if(flag) {                                      //如果需要则绘制拖拉效果
43           canvas.drawBitmap(chessZouQiflag,sXtart+
44                (Chess_LoadUtil.FILE_X(Chess_LoadUtil.SRC(xzgz))-2)*xSpan-chessR,
45                sYtart+(Chess_LoadUtil.RANK_Y(Chess_LoadUtil.SRC(xzgz))-2)
                  *ySpan-chessR, null);
46           canvas.drawBitmap(chessZouQiflag,sXtart+(bzcol+1)*xSpan-chessR,
47                sYtart+(bzrow+1)*ySpan-chessR, null);     //绘制移动时移动到的某格
48           }
49           if(cMfleg&&stack.size()>0) {                    //绘制计算机走棋的标志
50           canvas.drawBitmap(chessZouQiflag,sXtart+(
51           Chess_LoadUtil.FILE_X(Chess_LoadUtil.SRC(mvResult))-2)*xSpan-chessR,
52               sYtart+(Chess_LoadUtil.RANK_Y(Chess_LoadUtil.SRC(mvResult))-2)
                 *ySpan-chessR,null);
53           canvas.drawBitmap(chessZouQiflag,sXtart+
54                (Chess_LoadUtil.FILE_X(Chess_LoadUtil.DST(mvResult))-2)*xSpan-chessR,
55                sYtart+(Chess_LoadUtil.RANK_Y(Chess_LoadUtil.DST(mvResult))-2)
                  *ySpan-chessR, null);
56           }
57           if(flag){                                       //绘制拖着走的棋子，将其放大绘制
58                paint.setAlpha(200);//绘制放大的棋子
59                canvas.drawBitmap(scaleToFit(chessBitmap[ucpcSquares[xzgz]/16][
60                ucpcSquares[xzgz]%8],2),xMove-2*chessR,yMove-2*chessR, paint);
61           }}
```

- 第 3～11 行为绘制棋盘的横向线和竖向线。首先设置其颜色，然后设置其线的大小，最后绘制。
- 第 12～15 行为绘制九宫中的 4 条斜线。
- 第 16～20 行为绘制棋盘外围较粗的边框直线。第 21 行为绘制楚河。
- 第 22～35 行为绘制炮座和兵坐。

- 第 36～40 行为绘制棋子，根据棋盘数组，在相应的位置绘制相应的棋子。
- 第 41～48 行当玩家拖着棋子走时，绘制其拖动到的格子标志。
- 第 49～56 行为绘制计算机走棋的位置，绘制刚刚计算机所走棋子的初始位置和最终位置。
- 第 57～61 行为当玩家拖着棋子走时，绘制其放大的棋子。

（6）棋盘下方版块中的各个部件的绘制方法，其代码如下所示。

```
1    public void onDrawWindowMenu(Canvas canvas,float sXtart,float sYtart){
2         canvas.drawBitmap(scaleToFit(minueBeijint,1f),sXtart,sYtart+11.0f*ySpan,
          null);//菜单背景图
3         canvas.drawBitmap(ifPlayChess,sXtart+1f*xSpan,sYtart+11.4f*ySpan, null);
                                                        //绘制背景
4    if(playChessflag) {                                //如果是红方走棋，绘制红色的将
5         canvas.drawBitmap(scaleToFit(chessBitmap[1][0],0.9f),sXtart+1.1f*xSpan,
          sYtart+11.45f*ySpan, null);
6         }else{                                        //否则绘制黑色的将
7         canvas.drawBitmap(scaleToFit(chessBitmap[0][0],0.9f),sXtart+1.1f*xSpan,
          sYtart+11.45f*ySpan, null);
8         }
9         drawScoreStr(canvas,endTime/1000/60<10?"0"+endTime/1000/60:endTime/
          1000/60+"",
10                        sXtart+3f*xSpan,sYtart+11.4f*ySpan);     //绘制时间的分钟数
11        canvas.drawBitmap(dunhao,sXtart+scoreWidth*2+3f*xSpan,sYtart+11.4f*ySpan,
          null);//顿号
12        drawScoreStr(canvas,endTime/1000%60<10?"0"+endTime/1000%60:endTime/
          1000%60+"",
13                        scoreWidth*3+sXtart+3f*xSpan,sYtart+11.4f*ySpan);
                                                        //绘制时间的秒数
14        if(huiqiFlag) {                               //如果按下了悔棋按钮，放大绘制
15        canvas.drawBitmap(scaleToFit(huiqi,1.2f),sXtart+6.9f*xSpan,sYtart+11.25f*
          ySpan, null);
16        }else{
17             canvas.drawBitmap(huiqi,sXtart+7f*xSpan,sYtart+11.35f*ySpan, null);
18        }                                             //按原比例绘制
19        canvas.drawBitmap(scaleToFit(minueBeijint,1f),sXtart,sYtart+12.8f*ySpan,
          null);//菜单背景图
20        if(chonwanFlag) {                             //如果按下新局按钮，放大绘制
21        canvas.drawBitmap(scaleToFit(chonWan,1.2f),sXtart+0.3f*xSpan,sYtart+12.9f
          *ySpan, null);
22        }else{
23             canvas.drawBitmap(chonWan,sXtart+0.8f*xSpan,sYtart+13.2f*ySpan, null);
24        }                                             //按原比例绘制
25        if(isnoStart) {                               //开始暂停，如果已经开始
26        if(dianjiKaiShi) {                            //如果点击暂停按钮，放大绘制
27             canvas.drawBitmap(scaleToFit(suspend,1.2f),sXtart+3.0f*xSpan,sYtart
               +13.0f*ySpan, null);
28        }else
29             canvas.drawBitmap(suspend,sXtart+3.3f*xSpan,sYtart+13.2f*ySpan, null);
30        }else{                                        //按原比例绘制
31             if(dianjiKaiShi) {                       //如果，在暂停状态下点击此处了
32        canvas.drawBitmap(scaleToFit(start,1.2f),sXtart+3.0f*xSpan,sYtart+13.
          0f*ySpan, null);//放大绘制开始
33             }else                                    //按原比例绘制开始按钮
34             canvas.drawBitmap(start,sXtart+3.3f*xSpan,sYtart+13.2f*ySpan, null);
35             }
36        if(!isnoStart) {//难度是否可选，如果可选，则表示当前状态下可以点击，为暂停状态下是为可选
37        if(dianjiNanDu){                              //如果点击了难度按钮，放大绘制
38        canvas.drawBitmap(scaleToFit(nandutiaoZ,1.2f),sXtart+5.5f*xSpan,sYtart
          +13.0f*ySpan, null);
```

```
39          }else                                          //否则按原比例绘制
40               canvas.drawBitmap(nandutiaoZ,sXtart+5.8f*xSpan,sYtart+13.2f*ySpan, null);
41          }else{                                         //否则绘制灰度的难度按钮
42               canvas.drawBitmap(nonandutiaoZ,sXtart+5.8f*xSpan,sYtart+13.2f*ySpan, null);
43               }
44          if(isnoPlaySound) {                            //是否开启声音,如果是已经开启了声音
45          if(dianjishengyin) {                           //如果点击了开区域，则放大绘制
46               canvas.drawBitmap(scaleToFit(isPlaySound,1.2f),sXtart+8.0f*xSpan,
                 sYtart+13.1f*ySpan, null);
47          }else                                          //否则按原比例绘制
48               canvas.drawBitmap(isPlaySound,sXtart+8.3f*xSpan,sYtart+13.2f*ySpan, null);
49          }else{                                         //否则有叉的声音按钮
50               if(dianjishengyin){                        //如果点击了此区域，则放大绘制
51               canvas.drawBitmap(scaleToFit(noPlaySound,1.2f),sXtart+8.0f*xSpan,
                 sYtart+13.1f*ySpan, null);
52               }else                                      //否则按原比例绘制
53               canvas.drawBitmap(noPlaySound,sXtart+8.3f*xSpan,sYtart+13.2f*ySpan, null);
54               }
55          if(!isnoStart&&nanduBXZ) {                     //如果为按下了难度
56               canvas.drawBitmap(beijint3,sXtart,sYtart+14.6f*ySpan, null);
                                                            //绘制难度拖动条背景
57                    Rect r=new Rect(50,50,200,200);       //创建矩形
58                    if(dianjiJDT){                        //如果是点击了拖动区域
59                         length=(int)((xMove-sXtart)/(xZoom*20));
                                                            //计算要绘制的矩形个数，即长度
60                         nanduXS=(int)((xMove-sXtart)/(90*xZoom));     //计算难度系数
61                         if(nanduXS<1){                   //如果难度小于 1，肯定就为 1
62                              nanduXS=1;
63                         }
64                         if(nanduXS>5){                   //最高难度为 5
65                              nanduXS=5;
66                         }
67                         Constant.LIMIT_DEPTH=nanduXS*6;//搜索步骤总的为 30 步，分成 5 个等级
68                    }
69                    if(length<1){
70                         length=1;
71                    }else if(length>20){                  //最大矩形数为 20，最小为 1
72                         length=20;
73                    }
74                    for(int i=0;i<length;i++){            //循环绘制颜色矩形
75                    paint.setARGB(color[i][0], color[i][1], color[i][2], 0);
                                                            //设置画笔颜色，从颜色数组中取出
76                    r=new Rect((int) (sXtart+60*xZoom+i*xZoom*18),(int)(sYtart+15.3f
                      *ySpan),//创建矩形
77                    (int)(sXtart+60*xZoom+(i)*xZoom*18+13*xZoom),(int) (sYtart+15.3f*
                      ySpan+32*xZoom));
78                    canvas.drawRect(r, paint);            //绘制矩形
79                    }}else{                               //否则绘制广告条
80                    canvas.drawBitmap(minueBeijint,sXtart,sYtart+14.6f*ySpan, null);
                                                            //绘制广告条幅背景
81                    float left=sXtart+40*xZoom;           //计算广告裁减的矩形坐标
82                    float top=sYtart+15f*ySpan;
83                    float right=sXtart+9*xSpan;
84                    float bottom=sYtart+15.2f*ySpan+40*xZoom;
85                    canvas.save();                        //保护当前界面
86                    canvas.clipRect(new RectF(left,top,right,bottom));
                                                            //裁减该矩形区域的界面
87                    canvas.drawBitmap(guanggao2,guanggao2X,sYtart+14.8f*ySpan, null);
                                                            //绘制广告条幅
88                    canvas.restore();                     //恢复，后面画出来的不再剪裁
89               }}
```

- 第 2～18 行为绘制标识走棋方的图片，思考的剩余时间，以及悔棋按钮。剩余时间是每次轮到一方走棋时，将时间设置到最大值，另外一个线程会对其计时，当时间减到零时，该方为输，绘制时先绘制分钟数，然后绘制顿号，最后绘制秒数。每个数字都是单一的图片。对应悔棋按钮，当玩家点击到悔棋按钮的区域时，放大绘制，如果抬起来的时候任然在该区域时，说明玩家点击了该按钮。
- 第 19～54 行为绘制重玩按钮、开始或者暂停按钮、难度选择按钮，以及声音开启与关闭按钮。其中 19～24 行为绘制重玩，其是否绘制放大或者原比例绘制和上面绘制悔棋规则一样，其中第 25～35 行为绘制开始或者暂停按钮，当该界面在对局状态下时绘制暂停按钮，当该界面在暂停界面时绘制开始按钮，绘制方法和绘制悔棋按钮一样。36～43 行为绘制难度选择按钮，当在开始状态下，为不可选，即绘制灰度的难度文字图片，在暂停状态下时才可绘制。第 44～54 行为绘制声音标识按钮。
- 第 55～79 行为绘制难度拖动条，该拖动条为在按下了难度选择的情况下才绘制，其中第 59 行为计算需要的矩形个数，第 74～79 行为根据计算出来的矩形个数循环绘制需要的矩形。其颜色在给定的颜色数组中取，为了使颜色和界面统一风格，此处规定了颜色的渐变范围。这样可达到拖动选择难度的效果。
- 第 79～89 行为绘制滚动的条幅广告，不断改变其横坐标，并且将其裁减到对应区域，达到了滚动条的效果。当然其文字依然为一张图片，详细情况请见注释。

（7）初始化颜色数组的方法 initColer，其详细代码如下。

```
1    public void initColer(){
2              int r=(200-61)/20;                    //从其他地方找到需要渐变的颜色范围
3              int g=(159-24)/20;                    //然后得到其 RGB 值，将其等分 20 份
4              int b=(107-6)/20;                     //既可实现从一种颜色渐变到另外颜色
5              for(int i=0;i<20;i++){
6                        color[i][0]=61+i*r;
7                        color[i][1]=24+i*g;
8                        color[i][2]=5+i*b;
9              }}
```

（8）绘制数字组成的字符串的方法，drawScoreStr 其详细代码如下。

```
1    public void drawScoreStr(Canvas canvas,String s,float width,float height) {//绘制
     字符串方法
2         String scoreStr=s;
3         if(s.length()<2){
4              s="0"+s;
5         }
6         for(int i=0;i<scoreStr.length();i++){                //循环绘制数字
7         int tempScore=scoreStr.charAt(i)-'0';                //得到该数字
8         canvas.drawBitmap(iscore[tempScore], width+i*scoreWidth,height, null);
                                                               //从数组中取出对应的图片
9              }}
```

（9）初始化图片 initBitmap 的方法，其详细代码如下。

```
1    public void initBitmap(){
2              float xZoom=ViewConstant.xZoom;               //得到缩放比例
3              beijint4=scaleToFit(BitmapFactory.decodeResource(getResources(),
4    R.drawable.beijingkuangtu),xZoom)    ;                  //剪裁框
```

```
5                beijint3=scaleToFit(BitmapFactory.decodeResource(getResources(),
6    R.drawable.beijing3),xZoom);                                    //难度选择拖动它背景
7                fugaiTu=scaleToFit(BitmapFactory.decodeResource(getResources(),
8    R.drawable.shuyingfugai),xZoom);                                //覆盖图
9                winJiemian=scaleToFit(BitmapFactory.decodeResource(getResources(),
10   R.drawable.yingjiemian),xZoom);                                 //赢界面
11               loseJiemian=scaleToFit(BitmapFactory.decodeResource(getResources(),
12   R.drawable.shuijiemian),xZoom);                                 //输界面
13               ……//此处省略了其他图片的初始化，请读者自行查看随书光盘
14           }
```

说明　该方法将所有界面需要的图片进行初始化，并且根据缩放比例，每次调用 scaleToFit 方法将图片按照比例缩放，得到想要的比例的图片，用来自适应任意屏幕。

（10）接下来介绍的是在切换屏幕时，由系统调用的 surfaceCreated 方法实现，其详细代码如下。

```
1    @Override
2        public void surfaceCreated(SurfaceHolder holder) {
3            this.holder=holder;                                     //得到画布
4            canvas = null;
5          try {// 锁定整个画布，在内存要求比较高的情况下，建议参数不要为 null
6            canvas = holder.lockCanvas(null);
7            synchronized (holder) {
8            onDraw(canvas);                                         //绘制
9            }
10         } finally {
11           if (canvas != null) {
12           holder.unlockCanvasAndPost(canvas);                     //并释放锁
13           }   }
14         newThread();                                              //创建刷帧线程
15       }
```

说明　该方法为系统在切换到该界面时调用的方法，然后在该方法中调用绘制方法来绘制界面。再调用创建线程方法创建一个线程来对界面进行不断刷帧。

（11）接下来介绍的是新建线程的方法 newThread，其详细代码如下。

```
1    public void newThread(){
2            new Thread(){
3                @Override
4                public void run(){
5                    while(threadFlag) {
6                        if(isnoStart) {
7                            if(endTime-500<0) {            //如果思考时间没有了
8                                if(!cMfleg) {              //如果计算机正在走棋
9                                    yingJMflag=true;       //玩家赢了
10                                   LoadUtil.Startup();    //初始化棋盘
11                                   initArrays();          //初始化数组
12                                   endTime=zTime;         //总时间归位
13                                   isnoStart=false;       //该状态为暂停状态
14                                   dianjiJDT=false;       //点击拖动为 false
15                               }else{                     //否则为玩家输了
16                                   shuJMflag=true;        //标识为输了
17                                   LoadUtil.Startup();    //初始化棋盘
18                                   initArrays();          //初始化数组
19                                   endTime=zTime;         //总时间归位
```

```
20                                                  isnoStart=false;     //回归暂停状态
21                                                  dianjiJDT=false;     //点击拖动处为不可拖动
22                                          }}else {
23                                          endTime-=500;     //总时间每次减少 500 毫秒
24                                      }}
25                                  guanggao2X-=10;                //滚动条幅广告的 X 左边每次减 10
26                                  if(guanggao2X<-400*xZoom) {//如果减少到-400，则回归到 400
27                                      guanggao2X=400*xZoom;
28                                  }
29                                  onDrawcanvas();//重绘方法                //绘制
30                                  try {
31                                          Thread.sleep(500);          //每次休息 500 毫秒
32                                      } catch (InterruptedException e) {
33                                          e.printStackTrace();
34                                      }}}}.start();
35          }
```

- 第 6～24 行为在开始界面下，如果某方在规定时间内没有下完棋，则表示该方输了，当某一方输了时，转到相应的界面，并初始化相应的各个变量。否则时间每次自减 500 毫秒，并刷屏一次。
- 第 25～28 行为将滚动条幅的 *X* 坐标每次减少 10，当到达最左边时，再次回归到最右边。循环从右边滚动到左边。

（12）接下来介绍的是屏幕触摸事件的回调方法 onTouchEvent，其方法中又分为点击格子确定部分，触摸按下部分，触摸移动部分和触摸抬起部分，下面将分别对其进行介绍，整体源代码请读者自行查看随书光盘。这里首先介绍点击格子确定部分，即判断其点击点位置靠近于那个格子。其详细代码如下。

```
1     if(!cMfleg) {                                                    //如果正在进行计算机走棋
2                     return false;
3                 }
4                 int col = (int)( (e.getX()-sXtart)/xSpan); //计算得到其所在格子左下方的行列
5                 int row = (int) ((e.getY()-sYtart)/ySpan); //判断其属于格子周围 4 个点的那个
6                 if(((e.getX()-col*xSpan-sXtart)*(e.getX()-col*xSpan-sXtart)+
                                                                   //如果在该范围内，其为左下方行列
7                     (e.getY()-row*ySpan-sYtart)*(e.getY()-row*ySpan-sYtart))
8                     <xSpan/2*xSpan/2)          {
9                     bzcol=col-1;
10                    bzrow=row-1;
11                }
12                else if(((e.getX()-col*xSpan-sXtart)*(e.getX()-col*xSpan-sXtart)+
                                                                   //在该范围内为左上方行列
13                    (e.getY()-(row+1)*ySpan-sYtart)*(e.getY()-(row+1)*ySpan-sYtart))
14                    <xSpan/2*xSpan/2)     {
15                    bzcol=col-1;
16                    bzrow=row;
17                }else if(                                  //如果在该范围内，其为在右上方的行列
18                    ((e.getX()-(1+col)*xSpan-sXtart)*(e.getX()-(1+col)*xSpan-sXtart)
19                    +(e.getY()-(row+1)*ySpan-sYtart)*(e.getY()-(row+1)*ySpan-sYtart))
20                    <xSpan/2*xSpan/2)     {
21                    bzcol=col;
22                    bzrow=row;
23                }
24                else if(                                   //如果在该范围内，其为在右下方的行列
25                    ((e.getX()-(1+col)*xSpan-sXtart)*(e.getX()-(1+col)*xSpan-sXtart)+
26                    (e.getY()-row*ySpan-sYtart)*(e.getY()-row*ySpan-sYtart))
27                    <xSpan/2*xSpan/2) {
28                    bzcol=col;
```

```
29                  bzrow=row-1;
30                  }
```

说明　该部分通过计算点击处和周围的 4 个格子的距离，如果与某个格子的距离在规定半径内则说明点击的是该格子。

接下来介绍当玩家按下屏幕时的处理程序，其详细代码如下。

```
1     if(e.getAction()==MotionEvent.ACTION_DOWN) {      //如果按下
2                   if(yingJMflag||shuJMflag) {       //如果当前为赢界面
3          if(e.getX()>sXtart+3.9f*xSpan&&e.getY()>sYtart+8.5f*ySpan&&//如果是确定按钮区域
4               e.getY()<sYtart+8.5f*ySpan+1.5f*ySpan&&e.getX()
5               <sXtart+3.9f*xSpan+2*xSpan)     {
6                              dianjiQueDing=true;    //设其对应标志位 true，绘制时放大绘制
7                         }
8                         return true;
9                   }
10                  if(e.getX()>sXtart+0.5f*xSpan&&e.getY()>sYtart+13.3f*ySpan&&
                                                     //点击区域为新局按钮区域
11                        e.getY()<sYtart+13.3f*ySpan+1*ySpan&&e.getX()
12                        <sXtart+0.5f*xSpan+2*xSpan){
13                        chonwanFlag=true;            //对应标识设为 true,绘制时放大绘制
14                        return true;
15                  }
16                  if(e.getX()>sXtart+7f*xSpan&&e.getY()>sYtart+11.5f*ySpan&&
                                                     //点击处为悔棋按钮区域
17                        e.getY()<sYtart+11.5f*ySpan+1*ySpan&&e.getX()
18                        <sXtart+7f*xSpan+2*xSpan){
19                        huiqiFlag=true;              //对应标识设为 true,绘制时放大绘制
20                        return true;
21                  }
22                  if(e.getX()>sXtart+3.3f*xSpan&&e.getY()>sYtart+13.5f*ySpan&&
23                        e.getY()<sYtart+13.5f*ySpan+1*ySpan&&e.getX()
24                        <sXtart+3.3f*xSpan+2*xSpan) {              //点击处为开始按钮区域
25                        dianjiKaiShi=true;           //对应标识设为 true,绘制时放大绘制
26                        return true;
27                  }
28                  if(e.getX()>sXtart+5.8f*xSpan&&e.getY()>sYtart+13.5f*ySpan&&
29                        e.getY()<sYtart+13.5f*ySpan+1*ySpan&&e.getX()
30                        <sXtart+5.8f*xSpan+2*xSpan) {              //点击处为难度按钮区域
31                        dianjiNanDu=true;            //对应标识设为 true,绘制时放大绘制
32                        return true;
33                  }
34                  if(e.getX()>sXtart+8.3f*xSpan&&e.getY()>sYtart+13.5f*ySpan&&
35                        e.getY()<sYtart+13.5f*ySpan+1*ySpan&&e.getX()
36                        <sXtart+8.3f*xSpan+2*xSpan) {              //点击处为声音按钮区域
37                        dianjishengyin=true;         //对应标识设为 true,绘制时放大绘制
38                        return true;
39                  }
40                  if(e.getX()>sXtart&&e.getY()>sYtart+15.3f*ySpan&&
41                        e.getY()<sYtart+15.3f*ySpan+1*ySpan&&e.getX()
42                        <sXtart+10f*xSpan+2*xSpan) {               //难度选择进度条区域
43                        xMove=e.getX();                            //移动坐标设置为此处
44                        dianjiJDT=true;              //对应标识为 true,根据坐标绘制方块数
45                        return true;
46                  }if((bzrow+3)*16+bzcol+3<0||(bzrow+3)*16+bzcol+3>255||!isnoStart)
{
47                        return false; //否则看点击处是否为棋盘，或者是否开始了，开始了才对棋
                                        //盘操作有效
48                  }
```

```
49                  if(ucpcSquares[(bzrow+3)*16+bzcol+3]!= 0
50                  &&ucpcSquares[(bzrow+3)*16+bzcol+3]/16==0) {   //如果点击处有棋子而且
                                                                   //是自己的棋子
51                      xzflag=true;
52                      xzgz=(bzrow+3)*16+bzcol+3;        //选中的行列对应的一维数组下标
53                      flag=true;                        //将选中棋子的标志位设为 true
54                      onDrawcanvas();                   //重绘
55                      return true;
56                  }}
```

- 第 2～9 行为在输界面或者赢界面时，判断其点击处是否为确定按钮的区域，如果是，将其对应标志位设为 true，绘制时放大绘制，在下面抬起事件处理时，如果是在该区域抬起则说明玩家点击了确定按钮，则作出相应的处理。否则恢复原比例绘制。
- 第 10～46 行为，不在输赢界面的情况下，对应每个按钮的判断，如果点击到某个按钮的区域，该按钮可用，则将其对应标志设为 true，绘制时放大绘制。
- 第 46～48 行为判断其点击处均不为各个按钮区域，则判断其是否为对局开始，只有在开始状态下，才能对棋盘区域进行操作。并且点击的行列对应的一维数组下标要在范围内。因为整个棋盘是用 256 长度的一维数组表示的。
- 第 49～56 行为，在棋盘可操作的情况下，判断点击的行列处是否为自己的棋子，如果是，记录该棋子的位置，并且将标志位设为 true，绘制时绘制裁减窗口，并且将选中的棋子放大以后再绘制一次。并且其坐标为玩家触摸移动的位置。

介绍完了触摸事件的按下事件的处理代码，下面介绍其触摸移动事件处理代码，其详细代码如下。

```
1     else if(e.getAction()==MotionEvent.ACTION_MOVE){          //如果触摸事件为移动事件
2                      float x=xMove=e.getX();                  //得到其移动到的坐标
3                     float y=yMove=e.getY();
4                     if(xzflag) {                        //如果已经选中棋子，则此为要下棋
5                          if(x>48*10*xZoom+sXtart-windowWidth/2) {   //判断其是否在右边边沿，
6                              x=48*10*xZoom+sXtart-windowWidth/2;
7                          }
8                          else if(x<sXtart+windowWidth/2+0*48*xZoom){
                                                           //判断其是否在左边边沿
9                              x=sXtart+windowWidth/2-0*48*xZoom;
10                         }
11                         if(y>48*11*xZoom+sYtart-windowHeight/2) {
                                                           //判断其是否在棋盘的下边沿
12                             y=48*11*xZoom+sYtart-windowHeight/2;
13                         }
14                         else if(y<sYtart+windowHeight/2+0*48*xZoom){
                                                           //判断其是否为棋盘的上边沿
15                             y=sYtart+windowHeight/2-0*48*xZoom;
16                         }
17                         if(xMove>sXtart+5*xSpan) {  //判断其是否为移动在棋盘的右边部分
18                             sXtartCk=sXtart-(x-windowWidth/2-windowXstartLeft);
19                             sYtartCk=sYtart-(y-windowHeight/2-windowYstart);

20                         }
21                         else{                           //否则为棋盘的坐标部分
22                             sXtartCk=sXtart-(x-windowWidth/2-windowXstartRight);
23                             sYtartCk=sYtart-(y-windowHeight/2-windowYstart);
24                         }
25                         onDrawcanvas();                 //重绘方法
26                          return true;
27                    }else{
```

```
28                          xzflag=false;                    //否则恢复，不为选中状态
29                          flag=false;                      //选中了设为 false
30                          onDrawcanvas();                  //重绘方法
31                            return true;
32                      }}
```

● 第 4～16 行为在其选中棋子的情况下，根据其移动到的坐标，来计算将要绘制的视窗的坐标，如果其移动到边界，则视窗不再移动，将其边界与视窗的边界对齐，否则将移动到的位置绘制在视窗的中心。

● 第 17～24 行为根据移动到的位置，来确定要将视窗绘制在屏幕的左上方还是右上方，如果移动到的位置在右边，则绘制在左上方；如果在左边，则绘制在右上方。

最后介绍的是 onTouchEvent 方法中的抬起事件处理代码，该部分可分为输赢界面下的处理情况，操作各个选项按钮处理情况和走棋时的情况，下面分别进行介绍，首先介绍其输赢界面下的处理情况，其详细代码如下。

```
1                       if(yingJMflag||shuJMflag) {                //如果当前为赢界面
2                            if(dianjiQueDing==true){              //按下时为确定按钮区域
3                       if(e.getX()>sXtart+3.9f*xSpan&&e.getY()>sYtart+8.5f*ySpan&&
4                                       e.getY()<sYtart+8.5f*ySpan+1.5f*ySpan&&e.getX()
5                                       <sXtart+3.9f*xSpan+2*xSpan) {       //还在此区域抬起
6                                       shuJMflag=false;          //输赢标志恢复为 false
7                                       yingJMflag=false;
8                                       LoadUtil.Startup();       //初始化棋盘
9                                       initArrays();             //初始化数组
10                                      LoadUtil.sdPlayer=0;      //走棋方为自己
11                                      endTime=zTime;            //总时间
12                                      isnoStart=false;          //为暂停状态
13                                 }
14                                 dianjiQueDing=false;}          //恢复点击标志
15                            onDrawcanvas();                     //重绘方法
16                            return true;}
```

说明　在输赢界面下，当玩家点击确定按钮时，将输赢标识置为 false，并重新初始化棋盘，复制棋盘数组到绘制棋盘数组，总时间等归为最初状态。

接下来介绍的是，在按下了某个选项按钮的情况下抬起触摸时的处理代码，其详细代码如下。

```
1                       if(e.getX()>sXtart+7f*xSpan&&e.getY()>sYtart+11.5f*ySpan&&
2                                 e.getY()<sYtart+11.5f*ySpan+1*ySpan&&e.getX()
3                                 <sXtart+7f*xSpan+2*xSpan){ //如果抬起是在悔棋按钮区域
4                            if(huiqiFlag==true){              //如果按下时在悔棋按钮区域
5                                 if(!stack.empty()&&stack.size()>1)  {
                                                              //如果栈不为空，而且其大小大于 1
6                                      if(huiqibushu>huiqiBS){    //连续悔棋步数不能超过规定
7                                      Toast.makeText(father, "悔棋步数已经超过规定!",
8                                      Toast.LENGTH_SHORT
9                                      ).show();                  //超过规定还悔棋，弹出 toast 提醒
10                                     return true;}
11                                     huiqibushu++;              //否则悔棋步数自动加 1
12                           StackplayChess chess=stack.pop();          //从栈顶取出
13                           LoadUtil.UndoMovePiece(chess.mvResult, chess.pcCaptured);
                                                              //撤销走棋
14                           chess=stack.pop();                 //每次悔棋，撤销两步
15                           LoadUtil.UndoMovePiece(chess.mvResult, chess.pcCaptured);
16                                     if(!stack.empty()){    //得到上一次计算机走棋的位置
```

```
17                                  mvResult=stack.peek().mvResult;}
18                                  initArrays();           //复制绘制数组
19                              }
20                              huiqiFlag=false;            //悔棋标识置为 false
21                              onDrawcanvas();             //重绘
22                              return false; }
23                          }else if(huiqiFlag){            //如果抬起时不是该区域,
24                              huiqiFlag=false;            //但是按下时是按下了悔棋区域。
25                              onDrawcanvas();             //重绘
26                              return true;}
27                          if(e.getX()>sXtart+0.5f*xSpan&&e.getY()>sYtart+13.3f*ySpan&&
28                                  e.getY()<sYtart+13.3f*ySpan+1*ySpan&&e.getX()
29                                  <sXtart+0.5f*xSpan+2*xSpan){    //如果抬起时为新局按钮区域
30                              if(chonwanFlag==true){          //如果按下时为新局按钮区域
31                                  shuJMflag=false;            //输赢标志设为 false
32                                  yingJMflag=false;
33                                  isnoStart=false;
34                                  endTime=zTime;              //总时间回归原始
35                                  stack.clear();              //将栈清空
36                                  LoadUtil.Startup();         //初始化棋盘
37                                  initArrays();               //复制数组
38                              }
39                              chonwanFlag=false;              //否则新局标志设为 false
40                              onDrawcanvas();                 //重绘
41                              return true;
42                          }
43                          else if(chonwanFlag){               //如果抬起不在该区域
44                              chonwanFlag=false;              //对应标志设为 false
45                              onDrawcanvas();                 //重绘
46                              return true;}
47                          if(dianjiKaiShi) {                  //如果按下时为开始区域
48                              if(e.getX()>sXtart+3.3f*xSpan&&e.getY()>sYtart+13.5f*ySpan&&
49                                      e.getY()<sYtart+13.5f*ySpan+1*ySpan&&e.getX()
50                                      <sXtart+3.3f*xSpan+2*xSpan) {  //开始区域
51                                  isnoStart=!isnoStart;       //如果是开始则点击后为暂停
52                                  nanduBXZ=false;             //难度拖动条为不可选
53                                  if(!isnoStart) {            //如果为暂停状态
54                                      dianjiJDT=false;        //难度拖动条为不可拖动
55                                      isnoTCNDxuanz=false;    //难度选择也为不可用
56                                  }
57                                  dianjiKaiShi=false;         //点击开始标识设为 false
58                                  onDrawcanvas();             //重绘
59                              }
60                              dianjiKaiShi=false;             //点击开始标识设为 false
61                              onDrawcanvas();                 //重绘
62                              return true;
63                          }
64                          if(dianjiNanDu) {                   //按下时为难度区域
65                              if(e.getX()>sXtart+5.8f*xSpan&&e.getY()>sYtart+13.5f*ySpan&&
66                                      e.getY()<sYtart+13.5f*ySpan+1*ySpan&&e.getX()
67                                      <sXtart+5.8f*xSpan+2*xSpan) {  //抬起时为难度区域
68                                  if(!isnoStart) {                //暂停状态下才可用
69                                      nanduBXZ=!nanduBXZ;         //标志位置反
70                                      isnoTCNDxuanz=!isnoTCNDxuanz;
71                                  }else{
72                                      isnoTCNDxuanz=false;        //标志位
73                                      dianjiJDT=false;
74                                  }
75                                  isnoNanDu=!isnoNanDu;
76                              }
77                              dianjiNanDu=false;
78                              onDrawcanvas();                 //重绘
```

```
79                  return true;
80              }if(dianjishengyin) {                          //如果按下的为声音区域
81                  if(e.getX()>sXtart+8.3f*xSpan&&e.getY()>sYtart+13.5f*ySpan&&
82                          e.getY()<sYtart+13.5f*ySpan+1*ySpan&&e.getX()
83                          <sXtart+8.3f*xSpan+2*xSpan) {  //抬起时也为声音区域
84                      isnoPlaySound=!isnoPlaySound; }     //声音置反
85                  dianjishengyin=false;              //按下标志位设为 false
86                  onDrawcanvas();                         //重绘
87                  return true; }
```

- 第 1～26 行为悔棋操作，如果按下时在悔棋按钮区域，并且抬起时也在悔棋按钮区域，则为点击了悔棋按钮，该操作是从栈中取出走棋步骤，然后撤销该走棋步骤，每次执行重复两次走棋才能达到悔棋功能。
- 第 27～46 行为新开棋局按钮操作，当点击新开棋局按钮时，重新初始化棋盘和绘制棋盘，将相应的标志位归位即可。
- 第 47～63 行为点击开始或者暂停按钮后，该区域在开始状态下绘制暂停按钮图片，在暂停状态下绘制的是开始按钮图片。所以每次点击时，置反即可。当然在开始状况下，难度按钮就为不可操作状态，绘制时绘制灰度的难度按钮图片表示此时不可以操作。
- 第 64～80 行为点击了难度按钮后的处理代码。当难度按钮为可选状态时，每次按下时，拖动条可拖动状态置反即可，如果可拖动则绘制拖动条，否则依然绘制广告条幅。
- 第 80～87 行为点击声音按钮区域时的处理代码，每次点击时，将其是否播放声音标志位置反即可。

介绍完了以上两部分，最后需要介绍的是当触控区域在棋盘区域内时，对应不同的触控动作触发的走棋操作代码具体如下所列。

```
1                   if(xzflag) {                            //是否点击了要走的棋子
2                       if((bzrow+3)*16+bzcol+3<0||(bzrow+3)*16+bzcol+3>255){
                                                            //抬起时对应的行列在棋盘范围
3                           xzflag=false;                   //恢复选中棋子标志位
4                           flag=false;
5                           onDrawcanvas();                 //重绘
6                           return true; }
7                       if(bzrow<0||bzrow>9||bzcol<0||bzcol>8) {
                                                        //如果抬起时，对于行列不在棋盘范围
8                           xzflag=false;               //恢复标志位
9                           flag=false;
10                          onDrawcanvas();             //重绘
11                          return true; }
12                          int sqDst = DST(xzgz+((bzrow+3)*16+bzcol+3)*256);
                                                        //得到目标格子的数组下标
13                          int pcCaptured = ucpcSquares[sqDst];//得到目的格子的棋子
14                          int mv=xzgz+((bzrow+3)*16+bzcol+3)*256;//计算得到走棋步骤
15                          if(stack.size()>12&&//判断是否走车轱辘棋
16                              mv==stack.get(stack.size()-4).mvResult&&
17                              stack.get(stack.size()-1).mvResult==stack.get
                                (stack.size()-5).mvResult&&
18                              stack.get(stack.size()-5).mvResult==stack.get
                                (stack.size()-9).mvResult&&
19                              stack.get(stack.size()-2).mvResult==stack.get
                                (stack.size()-6).mvResult&&
20                              stack.get(stack.size()-6).mvResult==stack.get
                                (stack.size()-10).mvResult&&
21                              stack.get(stack.size()-3).mvResult==stack.get
```

```
                                (stack.size()-7).mvResult&&
22                              stack.get(stack.size()-3).mvResult==stack.get
                                (stack.size()-11).mvResult&&
23                              stack.get(stack.size()-4).mvResult==stack.get
                                (stack.size()-8).mvResult&&
24                              stack.get(stack.size()-8).mvResult==stack.get
                                (stack.size()-12).mvResult){
25                              xzflag=false;
26                              flag=false;
27                              onDrawcanvas();//重绘方法
28                              Toast.makeText(//同时发 Toast 提示玩家

29                              father,
30                              "不能重复局面走子，请重新走子。",
31                              Toast.LENGTH_SHORT).show();
32                              return true;
33                          }
34                           if (LegalMove(mv)) {            //如果走棋符合规则
35                  if (MakeMove(mv, 0)) {                   //如果没有被将军
36                         initArrays();                     //初始化数组
37                         father.playSound(2,1);            //播放声音，玩家走棋
38                         huiqibushu=0;                     //悔棋标志置零
39                         onDrawcanvas();                   //重绘
40                        stack.push(new StackplayChess
41                  (xzgz+((bzrow+3)*16+bzcol+3)*256,pcCaptured));//走棋步骤入栈
42                  if (IsMate()) {                          //如果玩家赢了
43                  LoadUtil.Startup();                      //初始化棋盘
44                       initArrays();                       //初始化数组
45                  yingJMflag=true;
46                  father.playSound(4,1);                   //播放声音,赢了
47                  onDrawcanvas();                   //重绘
48                   } else {                         //计算机走棋
49                         new Thread(){              //启动一个线程进行计算机走棋
50                             @Override
51                             public void run(){
52                             endTime=zTime;                //时间初始化
53                             playChessflag=true;           //正在走棋
54                             cMfleg=false;                 //正在走棋标志
55                             onDrawcanvas();               //重绘
56                             SearchMain();                 //搜索走棋步骤
57                             int sqDst = DST(mvResult);    //得到目标格子数组下标
58                             int pcCaptured = ucpcSquares[sqDst];//得到目的格子的棋子
59                             MakeMove(mvResult, 0);        //走棋
60                  stack.push(new StackplayChess(mvResult,pcCaptured));//走棋步骤入栈
61                                     initArrays();         //数组操作
62                                     if(LoadUtil.IsMate()){ //如果计算机赢了，
63                                          LoadUtil.Startup(); //初始化棋盘
64                                          initArrays();     //初始化数组
65                                          shuJMflag=true;   //输标志设为 true
66                                          father.playSound(5,1);  //播放声音,输了
67                                     }else
68                                     father.playSound(2,1);//播放声音,计算机走棋了
69                                     cMfleg=true;       //下完棋子，玩家可以操控了。
70                                     playChessflag=false;
71                                     endTime=zTime;
72                                     onDrawcanvas();    //重绘
73                             }  }.start();}}
74                             xzflag=false;              //为不可选中棋子
75                             flag=false;                //移动为 false
76                             onDrawcanvas();            //重绘
77                  }else{
78                             xzflag=false;              //为不可选中棋子
```

```
79                              flag=false;                          //移动为 false
80                              onDrawcanvas();                      //重绘
81                          }}
82                  return true;
83              }
```

- 第 2～33 行为判断玩家抬起的时候对应格子是否在范围和是否为重复局面。如果不在或者为重复局面则恢复对应的标志位，并且返回。
- 第 12～14 行为计算目标格子对应数组下标，目标格子上的棋子和代表走法的数字。这里的走法用一个数字表示，这个数字对 256 求余数得到起始位置对应的数组下标，整除 256 得到目标格子对应的数组下标。
- 15～33 行为判断玩家是否为重复局面走子，即和计算机来回的走同一步棋，如果同样的来回超过 3 次，则玩家不能再继续走该走法，即不符合规则。实现原理为查看计算机走棋和玩家走棋的各自前 6 步中的前 1 步、3 步、5 步是否均相同，前 2 步、4 步、6 步是否也均相同，玩家当前要走步骤是否和玩家的前 2 步、4 步、6 步相同。如果各自的两种情况均相同，并且当前步骤满足要求，则说明为重复局面，因所有的走棋步骤均存入栈中了，所以直接取出比较即可。
- 第 34～48 行为玩家走棋操作，如果走棋符合规则，并且走棋后不被将军，则为可行，则走棋，走完棋后判断是否玩家赢了，如果赢了则切换到赢界面。否则不作处理。
- 第 48～73 行为计算机走棋，由于该步骤耗时较多，所以，重新开一个线程来进行走棋步骤的搜索。并且在走棋状态下玩家不可操控界面，搜索完毕后将走棋步骤入栈并走棋，然后判断玩家是否输了，如果输了切换到输界面。否则不作处理。

13.6.2　游戏界面常量类 ViewConstant

介绍完了整个游戏的界面绘制类，接下来介绍的是与界面绘制类相关的常量类 ViewConstant，该类为界面提供了一些静态常量和静态方法，其详细代码如下。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 ViewConstant.java。

```
1    package com.bn.chess;
2    import android.graphics.Bitmap;
3    import android.graphics.Matrix;
4    public class ViewConstant {
5         public static float sXtart=0;                          //棋盘的起始坐标
6         public static float sYtart=0;
7         public static boolean yingJMflag;                      //赢界面标志
8         public static boolean shuJMflag;                       //输界面标志
9         public static float height;                            //屏幕高度
10       public static float width;                              //屏幕宽度
11       public static int huiqiBS=2;                            //最大悔棋步数
12        public static boolean isnoPlaySound=true;              //是否播放声音
13        public static boolean isComputerPlayChess=false;//是否为计算机走棋,初始为不是
14        public static boolean isHeqi=false;                    //是否为和棋
15        public static boolean isnoStart=false;                 //是否为开始
16        public static boolean isnoTCNDxuanz;                   //是否为弹出了难度选择
17        public static int nanduXS=1;                           //难度系数
18        public static int thinkDeeplyTime=1;                   //计算机思考时间
19        public static int zTime=900000;                        //每走一步最大思考时间
20        public static int endTime=zTime;                       //思考时间
21        public static float xZoom=1F;                          //X 方向的缩放比例
```

```
22        public static float yZoom=1F;                                  //Y 方向上的缩放比例
23        public static float xSpan=48.0f*xZoom;                         //X 方向上的单元格大小
24        public static float ySpan=48.0f*yZoom;                         //Y 方向上的单元格大小
25        public static float scoreWidth = 7*xZoom*4f;                   //时间数字间隔
26        public static float sXtartCk;                                  //裁减窗口的起始坐标
27        public static float sYtartCk;
28        public static float windowWidth=200*xZoom;                     //裁减窗口的大小
29        public static float windowHeight=400*xZoom;
30        public  static  float  windowXstartLeft=sXtart+(5*xSpan-windowWidth)/2*xZoom;
                                                                         //小窗口的起始坐标
31        public static float windowXstartRight=sYtart+5*xSpan+(5*xSpan-windowWidth)/2*xZoom;
32        public static float windowYstart=sYtart+ySpan*4*xZoom;
33        public static float chessR=30*xZoom;                           //棋子半径
34        public static float fblRatio=0.6f*xZoom;                       //棋子缩放比例
35        public static Bitmap scaleToFit(Bitmap bm,float fblRatio) {  //缩放图片的方法
36        int width = bm.getWidth();                                     //图片宽度
37        int height = bm.getHeight();                                   //图片高度
38        Matrix matrix = new Matrix();
39        matrix.postScale((float)fblRatio, (float)fblRatio);
                                                   //图片等比例缩小为原来的 fblRatio 倍
40        Bitmap bmResult = Bitmap.createBitmap(bm, 0, 0, width, height, matrix, true);
                                                   //声明位图
41        return bmResult;                                               //返回被缩放的图片
42      }
43        public static void initChessViewFinal(){                       //初始化变量的方法
44             xSpan=48.0f*xZoom;                                        //单元格大小
45             ySpan=48.0f*yZoom;
46             scoreWidth = 7*xZoom*4f;                                  //时间数字间隔
47             windowWidth=200*xZoom;                                    //窗口的大小
48             windowHeight=250*xZoom;
49             windowXstartLeft=sXtart+(5*xSpan-windowWidth)/2*xZoom;//小窗口的起始坐标
50             windowXstartRight=sYtart+5*xSpan+(5*xSpan-windowWidth)/2*xZoom;
51             windowYstart=sYtart+ySpan*1*xZoom;
52             chessR=30*xZoom;                                          //棋子半径
53             fblRatio=0.6f*xZoom;                                      //棋子缩放比例
54        }}
```

- 第 5～34 行为声明了各个静态变量，供界面绘制时使用。通过改变其中某些变量即可不动其他任何代码实现整体调整的效果。
- 第 35～42 行为将图片进行等比例缩放的方法，其中第 38～40 行为根据需要缩放图片。
- 第 43～54 行为初始化各个静态变量，在获得了手机屏幕分辨率后调用，用来实现自适应屏幕。

13.7 走法引擎相关类

完成了游戏界面的开发，最后介绍的是游戏开发中最为重点部分，走法引擎部分。该引擎通过最小最大搜索算法，以局面价值分数作依据来搜索出最好的下棋步骤。其过程是首先生成该局面下所有的走法，然后走一步进行一次局面评估。当某一步的局面评估价值最大时即为最佳下棋步骤。

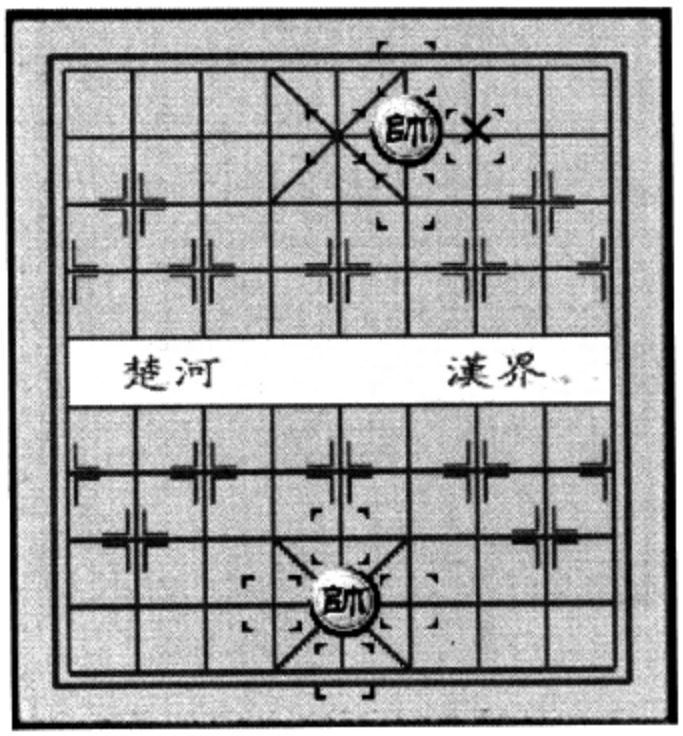
13.7.1 常量类 Constant

该类提供了整个走法引擎所需要的常量，如每个棋子在棋盘中对应位置的子力价值，每个棋子对应的常量，各个棋子走棋规则的规则数组等。在介绍其代码之前，首先对各个棋子的走棋规则做

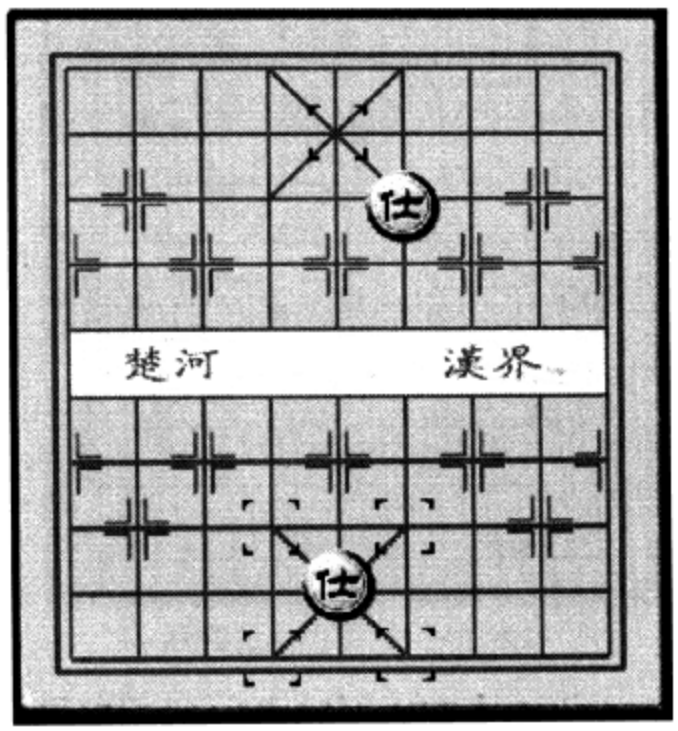
一下介绍，其步骤如下。

（1）王的走棋规则，王只能在九宫之内活动，其步长也为一，直着走子，如图 13-11 所示。有叉的地方为不可走。有方框而且没有叉的地方为可走，后面的情况也如此。

（2）士的走棋规则，士只能在九宫之内活动，其步长为二，斜着走子，如图 13-12 所示。



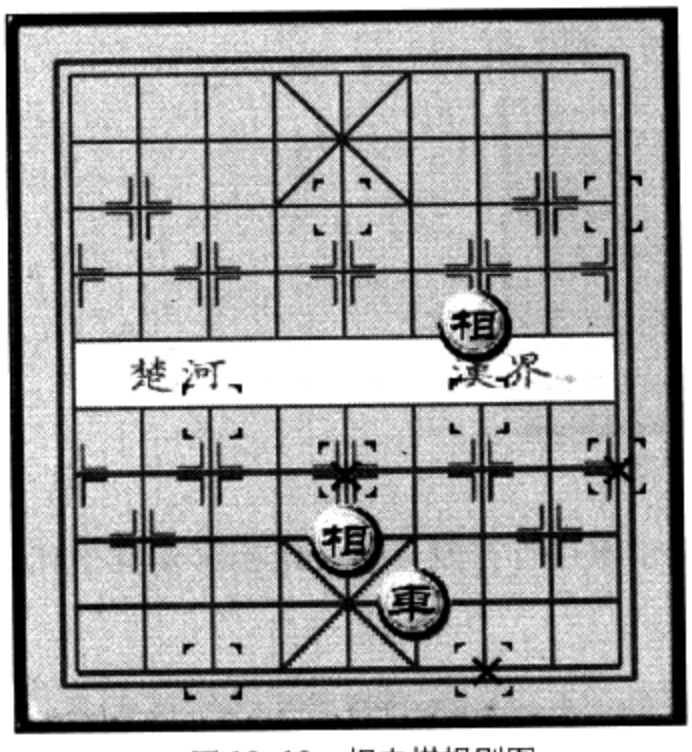
▲图 13-11　王走棋规则图



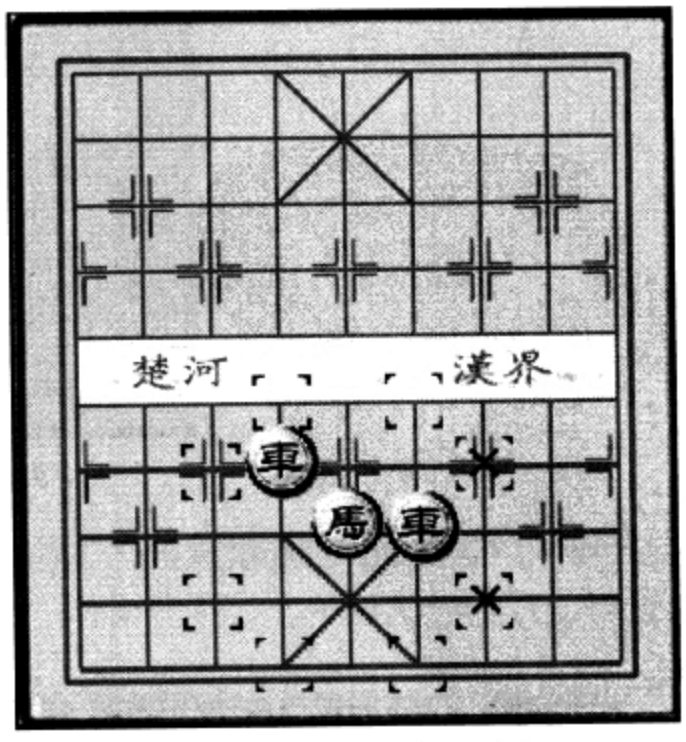
▲图 13-12　士走棋规则图

（3）相的走棋规则，相只能在楚河的一边，即该方一开始所在局面的一边活动，步长为 3，斜着走子，田心不能有子，如图 13-13 所示。

（4）马的走棋规则，马能在整个棋盘内活动，该子为走斜日，不能有蹩腿。其着棋规则如图 13-14 所示。



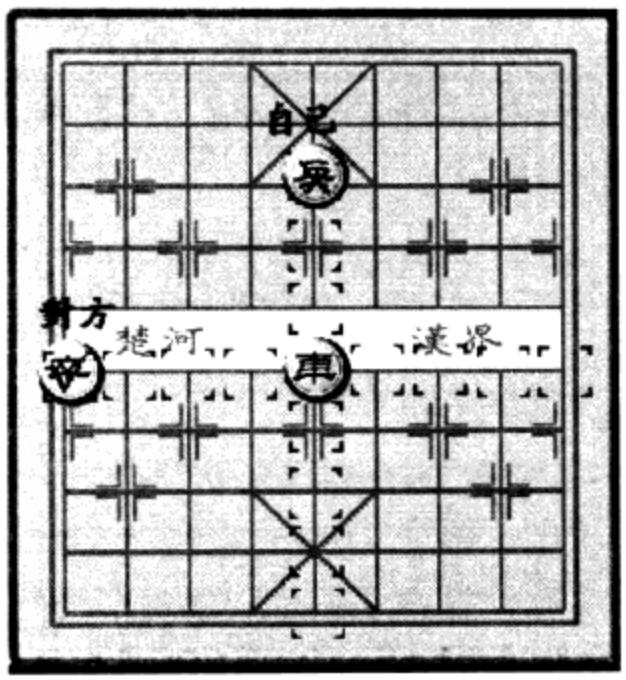
▲图 13-13　相走棋规则图



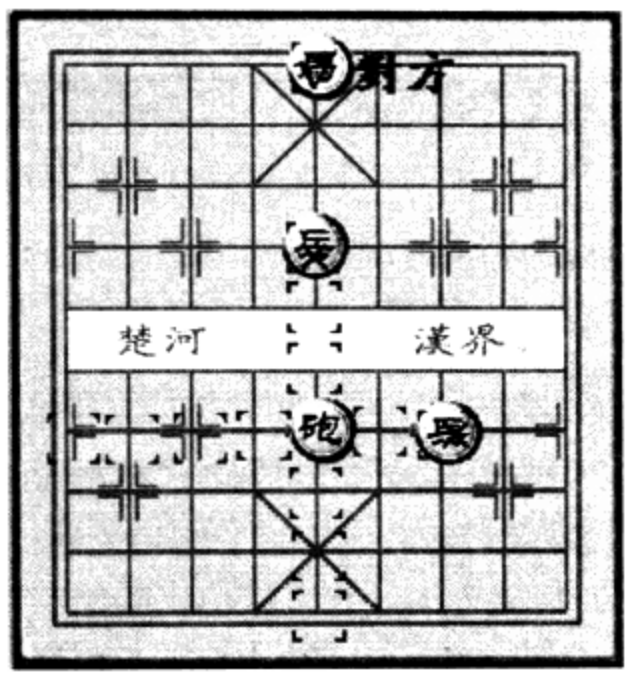
▲图 13-14　马走棋规则图

（5）车的走棋规则，车能在整个棋盘内活动，该子为直着走棋，其走棋规则如图 13-15 所示。

（6）砲的走棋规则，砲能在整个棋盘内活动，该子为直走，隔子吃子。其走棋规则如图 13-16 所示。

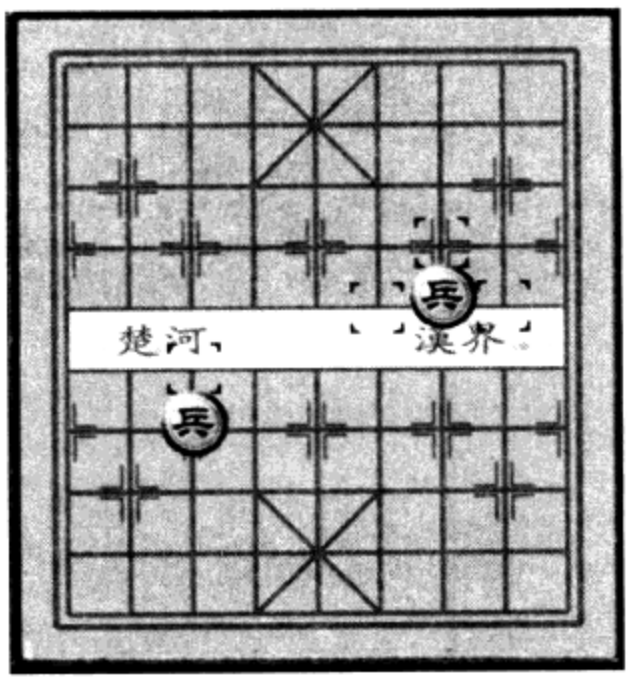


▲图 13-15　车走棋规则图



▲图 13-16　砲走棋规则图

（7）兵的走棋规则，兵在还未过楚河之前只能向前走，每次只能走一步，过了楚河后可以横着和向前直着走，每次也只能走一步，走棋规则如图 13-17 所示。



▲图 13-17　兵走棋规则图

介绍完了各个棋子的走棋规则，下面正式开始对走法引擎进行开发。首先给出常量类中代码，其常量类的详细代码如下。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 Constant.java。

```
1    package com.bn.chess;
2    public class Constant {
```

```
3         public final static int RANK_TOP = 3;                    // 棋盘范围，上方范围
4         public final static int RANK_BOTTOM = 12;                //下方范围
5         public final static int FILE_LEFT = 3;                   //左边范围
6         public final static int FILE_RIGHT = 11;                 //右边范围
7         public final static int PIECE_KING = 0;                  // 棋子编号，王
8         public final static int PIECE_ADVISOR = 1;               //士
9         public final static int PIECE_BISHOP = 2;                //相
10        public final static int PIECE_KNIGHT = 3;                //马
11        public final static int PIECE_ROOK = 4;                  //车
12        public final static int PIECE_CANNON = 5;                //砲
13        public final static int PIECE_PAWN = 6;                  //兵
14        public final static int MAX_GEN_MOVES = 128;             // 最大的生成走法数，
15        public static int LIMIT_DEPTH =30;                       // 最大的搜索深度
16        public final static int MATE_VALUE = 10000;              // 最高分值，即将死的分值
17        public final static int WIN_VALUE = MATE_VALUE - 100;
                                                        // 搜索出胜负分值界限，超出此值为杀棋
18        public final static int ADVANCED_VALUE = 3;      // 先行权分值
19        public static short ccInBoard[]=new short[] {   // 判断棋子是否在棋盘中的数组
20          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,        0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0,
21          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,        0, 0, 0, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 0, 0, 0, 0,
22          0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,        0, 0, 0, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 0, 0, 0, 0,
23          0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,        0, 0, 0, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 0, 0, 0, 0,
24          0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,        0, 0, 0, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 0, 0, 0, 0,
25          0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,        0, 0, 0, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 0, 0, 0, 0,
26          0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,        0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0,
27          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,        0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0
28        };
29        public static short ccInFort[] = {                      // 判断棋子是否在九宫的数组
30          ……//该处省略了数组中的部分数字，请读者自行查看随书光盘
31          0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,        0, 0, 0, 0, 0, 0, 1,
            1, 1, 0, 0, 0, 0, 0, 0, 0,
32          0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
33           ……//该处省略了数组中的部分数字，请读者自行查看随书光盘
34        };
35        public static short ccLegalSpan[] = {
36           ……//该处省略了数组中的部分数字，请读者自行查看随书光盘
37          0, 0, 0, 0, 0, 3, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0,        0, 0, 0, 0, 0, 0, 2,
            1, 2, 0, 0, 0, 0, 0, 0, 0,
38          0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,        0, 0, 0, 0, 0, 0, 2,
            1, 2, 0, 0, 0, 0, 0, 0, 0,
39          0, 0, 0, 0, 0, 3, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0,
40           ……//该处省略了数组中的部分数字，请读者自行查看随书光盘
41        };      // 判断步长是否符合特定走法的数组，1=帅(将)，2=仕(士)，3=相(象)
42        public static short ccKnightPin[] = {            // 根据步长判断马是否蹩腿的数组
43           ……//该处省略了数组中的部分数字，请读者自行查看随书光盘
44          0,  0,  0,  0,  0,  0,-16,  0,-16,  0,  0,  0,  0,  0,  0,  0,
45          0,  0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,
46          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
47          0,  0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,
48          0,  0,  0,  0,  0,  0, 16,  0, 16,  0,  0,  0,  0,  0,  0,  0,
49           ……//该处省略了数组中的部分数字，请读者自行查看随书光盘
50        };
51        public static short ccKingDelta[] = {-16, -1, 1, 16};        // 帅(将)的步长
52        public static short ccAdvisorDelta[] = {-17, -15, 15, 17};   // 仕(士)的步长
```

```
53        public static short ccKnightDelta[][] =     // 马的步长，以帅(将)的步长作为马腿
54                                  {{-33, -31}, {-18, 14}, {-14, 18}, {31, 33}};
55        public static short ccKnightCheckDelta[][] = {{-33, -18}, {-31, -14},
56                                 {14, 31}, {18, 33}};                  // 马被将军的步长
57        public static  short cucpcStartup[] = {                        // 棋盘初始位置
58          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
59          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
60          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
61          0,  0,  0, 20, 19, 18, 17, 16, 17, 18, 19, 20,  0,  0,  0,  0,
62          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
63          0,  0,  0,  0, 21,  0,  0,  0,  0,  0, 21,  0,  0,  0,  0,  0,
64          0,  0,  0, 22,  0, 22,  0, 22,  0, 22,  0, 22,  0,  0,  0,  0,
65          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
66          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
67          0,  0,  0, 14,  0, 14,  0, 14,  0, 14,  0, 14,  0,  0,  0,  0,
68          0,  0,  0,  0, 13,  0,  0,  0,  0,  0, 13,  0,  0,  0,  0,  0,
69          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
70          0,  0,  0, 12, 11, 10,  9,  8,  9, 10, 11, 12,  0,  0,  0,  0,
71          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
72          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
73          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
74        };
75        public static  short cucvlPiecePos[][] = {                     // 子力位置价值表
76          {                                                            // 炮的子力价值
77            0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
78            0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
79            0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
80            0,  0,  0,100,100, 96, 91, 90, 91, 96,100,100,  0,  0,  0,  0,
81            0,  0,  0, 98, 98, 96, 92, 89, 92, 96, 98, 98,  0,  0,  0,  0,
82            0,  0,  0, 97, 97, 96, 91, 92, 91, 96, 97, 97,  0,  0,  0,  0,
83            0,  0,  0, 96, 99, 99, 98,100, 98, 99, 99, 96,  0,  0,  0,  0,
84            0,  0,  0, 96, 96, 96, 96,100, 96, 96, 96, 96,  0,  0,  0,  0,
85            0,  0,  0, 95, 96, 99, 96,100, 96, 99, 96, 95,  0,  0,  0,  0,
86            0,  0,  0, 96, 96, 96, 96, 96, 96, 96, 96, 96,  0,  0,  0,  0,
87            0,  0,  0, 97, 96,100, 99,101, 99,100, 96, 97,  0,  0,  0,  0,
88            0,  0,  0, 96, 97, 98, 98, 98, 98, 98, 97, 96,  0,  0,  0,  0,
89            0,  0,  0, 96, 96, 97, 99, 99, 99, 97, 96, 96,  0,  0,  0,  0,
90            0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
91            0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
92            0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
93          }
94              ……//此处省去了其他棋子的子力价值表，请读者自行查看随书光盘中的源代码
95    };}
```

- 第 3～6 行为棋盘的范围，棋盘用长度为 256 的一维数组模拟 16*16 的二维数组。所以，棋子所在数组中的对应二维数组位置为该段代码中给出范围。
- 第 7～13 行给出了代表各个棋子的常量。
- 第 14～18 行给出了搜索时要用到的相关常量，其中第 14 行为每次搜索一个局面时，搜索的最多的走法数，如果超出该给出数目则不再生成剩下的走法。第 15 行为给出搜索时搜索的最大深度。深度为每次搜索时走完多少步棋后才返回局面，评估分数。搜索越深，计算机越聪明。第 16 行为给出了在将死的情况下的界面评估分数，当然将死对方时，该方的局面分数是最大的，为无穷大，用该数表示无穷大。
- 第 19～28 行为判断棋子位置是否在棋盘范围内的数组，在判断一个走法是否合法时会用到。
- 第 29～34 行为判断一个棋子是否在九宫之内的数组，均用长度为 256 的一维数组来表示。如果对应该位置为 1，表示为在九宫以内。

- 第 35～41 行给出了王、士和相的走棋步长，即横向和竖向的步长。
- 第 42～50 行给出了马的蹩腿的步长数组，分别有左上、左下、右上和右下。
- 第 51～54 行分别给出了王、士、马的步长，这里马腿位置的步长用王的步长。
- 第 55～56 行为判断马是否将军时候的步长。
- 第 57～95 行为每个棋子在棋盘中对应位置的子力价值，子力价值表示一个棋子在棋盘中一个位置时的价值，而将一方所有在棋盘上的棋子的子力价值全加起来表示该方在该局面下的局面价值。局面价值越大，说明该局面对该方越有利，这是程序搜索最好下棋步骤的依据。此处只给出了炮的子力价值表，其他棋子的子力价值表请读者自行查看随书光盘。

13.7.2　工具类 Chess_LoadUtil

该类为该引擎提供一些必要的方法工具，其详细代码如下。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 Chess_LoadUtil.java。

```
1    package com.bn.chess;
2    import static com.bn.chess.Constant.*;
3    public class Chess_LoadUtil {                          // 判断棋子是否在棋盘中
4         public static boolean IN_BOARD(int sq) {
5           return ccInBoard[sq] != 0;
6         }
7         public static boolean IN_FORT(int sq) {            // 判断棋子是否在九宫中
8           return ccInFort[sq] != 0;
9         }
10        public static int RANK_Y(int sq) {                 // 获得格子的纵坐标 row
11          return sq >> 4;
12        }
13        public static int FILE_X(int sq) {                 // 获得格子的横坐标 col
14          return sq & 15;
15        }
16        public static int COORD_XY(int x, int y) {         // 根据纵坐标和横坐标获得格子
17          return x + (y << 4);
18        }
19        public static int SQUARE_FLIP(int sq) {            // 翻转格子
20          return 254 - sq;
21        }
22        public static int FILE_FLIP(int x) {                       // 纵坐标水平镜像
23          return 14 - x;
24        }
25        public static int RANK_FLIP(int y) {                       // 横坐标垂直镜像
26          return 15 - y;
27        }
28        public static int MIRROR_SQUARE(int sq) {                  // 格子水平镜像
29          return COORD_XY(FILE_FLIP(FILE_X(sq)), RANK_Y(sq));
30        }
31        public static int SQUARE_FORWARD(int sq, int sd) {         // 格子水平镜像
32          return sq - 16 + (sd << 5);
33        }
34        public static boolean KING_SPAN(int sqSrc, int sqDst) {
                                                             // 走法是否符合帅(将)的步长
35          return ccLegalSpan[sqDst - sqSrc + 256] == 1;
36        }
37        public static boolean ADVISOR_SPAN(int sqSrc, int sqDst) {
                                                             // 走法是否符合仕(士)的步长
38          return ccLegalSpan[sqDst - sqSrc + 256] == 2;
39        }
```

```
40        public static boolean BISHOP_SPAN(int sqSrc, int sqDst) {
                                                        // 走法是否符合相(象)的步长
41          return ccLegalSpan[sqDst - sqSrc + 256] == 3;
42        }
43        public static int BISHOP_PIN(int sqSrc, int sqDst) {     // 相(象)眼的位置
44          return (sqSrc + sqDst) >> 1;
45        }
46        public static int KNIGHT_PIN(int sqSrc, int sqDst) {     // 马腿的位置
47          return sqSrc + ccKnightPin[sqDst - sqSrc + 256];
48        }
49        public static boolean HOME_HALF(int sq, int sd) {        // 是否未过河
50          return (sq & 0x80) != (sd << 7);
51        }
52        public static boolean AWAY_HALF(int sq, int sd) {        // 是否已过河
53          return (sq & 0x80) == (sd << 7);
54        }
55        public static boolean SAME_HALF(int sqSrc, int sqDst) { // 是否在河的同一边
56          return ((sqSrc ^ sqDst) & 0x80) == 0;
57        }
58        public static boolean SAME_RANK(int sqSrc, int sqDst) { // 是否在同一行
59          return ((sqSrc ^ sqDst) & 0xf0) == 0;
60        }
61        public static boolean SAME_FILE(int sqSrc, int sqDst) { // 是否在同一列
62          return ((sqSrc ^ sqDst) & 0x0f) == 0;
63        }
64        public static int SIDE_TAG(int sd) {        // 获得红黑标记(红子是 8，黑子是 16)
65          return 8 + (sd << 3);
66        }
67        public static int OPP_SIDE_TAG(int sd) {    // 获得对方红黑标记
68          return 16 - (sd << 3);
69        }
70        public static int SRC(int mv) {                      // 获得走法的起点
71          return mv & 255;
72        }
73        public static int DST(int mv) {                      // 获得走法的终点
74          return mv >> 8;
75        }
76        public static int MOVE(int sqSrc, int sqDst) {       // 根据起点和终点获得走法
77          return sqSrc + sqDst * 256;
78        }
79        public static int MIRROR_MOVE(int mv) {              // 走法水平镜像
80          return MOVE(MIRROR_SQUARE(SRC(mv)), MIRROR_SQUARE(DST(mv)));
81        }}
```

说明 该类为整个引擎提供了各种需要的方法，多数方法都是通过操作常量类中的相应数据来为其服务的。

13.7.3 走法引擎主类 LoadUtil

该类为整个引擎的主类，是整个引擎的核心，该类通过最大最小搜索，Alpha-Beta 搜索，迭代加深搜索，Alpha 截断，Beta 截断等算法来共同搜索出最佳的下棋步骤。下面就对其进行详细介绍，在介绍其详细代码之前，首先对整体框架做一个简单介绍，然后再对其进行详细讲解。其框架代码如下。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 LoadUtil.java。

```
1     package com.bn.chess;
2     import static com.bn.chess.Constant.*;
3     import static com.bn.chess.Chess_LoadUtil.*;
```

```
4    import java.util.Arrays;
5    public class LoadUtil {
6         public static int sdPlayer=1;                          // 轮到谁走，0=红方，1=黑方
7         public static int ucpcSquares[]=new int[256];          // 棋盘上的棋子
8         public static int vlWhite, vlBlack;                    // 红、黑双方的子力价值
9         public static int nDistance;                           // 距离根节点的步数
10        public static int mvResult;                            // 计算机走的棋
11        public static int nHistoryTable[]=new int[65536];      // 历史表
12        public static void initHistorytable(){                 //初始化历史表
13        for(int i=0;i<nHistoryTable.length;i++){
14             nHistoryTable[i]=0;
15        }}
16        public static void ChangeSide() {                      // 交换走子方
17           sdPlayer = 1 - sdPlayer;
18        }
19        public static void AddPiece(int sq, int pc) {          // 在棋盘上放一枚棋子
20           ……//此处省略了相关的代码，将在后面的步骤中给出
21        }
22        public static void DelPiece(int sq, int pc) {          // 从棋盘上拿走一枚棋子
23           ……//此处省略了相关的代码，将在后面的步骤中给出
24        }
25        public static int Evaluate()  {                                // 局面评价函数
26          return (sdPlayer == 0 ? vlWhite - vlBlack : vlBlack - vlWhite) + ADVANCED_VALUE;
27        }
28        public static void UndoMakeMove(int mv, int pcCaptured) {     // 撤销走一步棋
29           ……//此处省略了相关的代码，将在后面的步骤中给出
30        }
31        public static void Startup() {                                 // 初始化棋盘
32         ……//此处省略了相关的代码，将在后面的步骤中给出
33        }
34        public static int MovePiece(int mv) {                          // 搬一步棋的棋子
35         ……//此处省略了相关的代码，将在后面的步骤中给出
36        }
37        public static void UndoMovePiece(int mv, int pcCaptured) {
                                                                         // 撤销搬一步棋的棋子
38         ……//此处省略了相关的代码，将在后面的步骤中给出
39        }
40        public static boolean MakeMove(int mv, int pcCaptured) {// 走一步棋
41          ……//此处省略了相关的代码，将在后面的步骤中给出
42        }
43        public static int GenerateMoves(Integer[] mvs)  {              // 生成所有走法
44          ……//此处省略了相关的代码，将在后面的步骤中给出
45        }
46        public static boolean LegalMove(int mv)  {                     // 判断走法是否合理
47          ……//此处省略了相关的代码，将在后面的步骤中给出
48        }
49          public static boolean Checked()  {                           // 判断是否被将军
50         ……//此处省略了相关的代码，将在后面的步骤中给出
51        }
52        public static boolean IsMate() {                               // 判断是否被杀
53         ……//此处省略了相关的代码，将在后面的步骤中给出
54        }
55        public static int SearchFull(int vlAlpha, int vlBeta, int nDepth) {
                                                                         // 超出边界的 Alpha-Beta 搜索过程
56          ……//此处省略了相关的代码，将在后面的步骤中给出
57        }
58        public static void SearchMain() {// 迭代加深搜索过程
59        ……//此处省略了相关的代码，将在后面的步骤中给出
60        }}
```

- 第 19～21 行为在棋盘上放一枚棋子，当放下一枚棋子时，要将放下的该棋子对应该位置的

子力价值数值加到该方在该局面下的局面价值数值上。

- 第 22～24 行为在棋盘上拿走一枚棋子，当拿走一枚棋子时，要将拿走的该棋子对应该位子的子力价值数值从该方在该局面下的局面价值数值上减去。
- 第 25～27 行为计算局面价值评估方法，该方法给出当前局面下该方相对于对方的局面价值差值，分数越大表示越有利。
- 第 28～30 行为撤销一步下棋，该方法是在搜索过程中，要先走一步棋然后给出局面评估价值，在返回后将该下棋步骤撤销。
- 第 31～33 行为初始化棋盘方法，该方法根据棋子最初的位置，将其存入搜索数组。
- 第 34～36 行为搬一步棋子方法，该方法为将走棋步骤传入，然后将该步骤棋子起始位置和目标位置的棋子搬走，在把起始位置上的棋子搬到目标位置，与此同时，局面价值等要进行相应的变更。
- 第 37～39 行为撤销搬一步棋的方法，该方法要传入走棋的步骤，以及该走棋步骤下的目标位置上的棋子。先将该棋子搬到初始位置，如果该步骤下的目标位置上的棋子不为空，即是吃子走法时，把被吃掉的棋子重新搬到该位置，以此来实现撤销搬一步棋的功能。
- 第 40～42 行为走一步棋，该方法先走一步棋，如果走棋后被将军了，则撤销该走棋步骤，并返回 false，表示被将军，此步不能走，撤销走法。
- 第 43～45 行为生成所有走法方法，该方法根据该局面来生成所有该方的走棋步骤。
- 第 46～48 行为在其生成了所有走法后，判断每个走法是否合理的方法。只会把合理的保留下来。
- 第 49～51 行为判断该局面下该方是否被将军。
- 第 55～57 行为最主要的方法，该方法通过递归调用，遍历所有走法，然后得出每个走法下的局面价值，根据其价值的大小搜索出最佳下棋步骤。
- 第 58～60 行为迭代加深搜索，该方法为从 0 到最大搜索深度进行搜索。

（1）在棋盘上放一枚棋子的方法 AddPiece，其详细代码如下。

```
1    public static void AddPiece(int sq, int pc) {              // 在棋盘上放一枚棋子
2                  ucpcSquares[sq] = pc;
3                  if (pc < 16) { // 红方加分，黑方(注意"cucvlPiecePos"取值要相反)减分
4                    vlWhite += cucvlPiecePos[pc - 8][sq];      //将其子力价值加到红方上
5                  } else {
6                    vlBlack += cucvlPiecePos[pc - 16][SQUARE_FLIP(sq)];
                                                              //将其子力价值加到黑方上
7                  }}
```

（2）在棋盘上拿走一枚棋子方法 DelPiece，其详细代码如下。

```
1    public static void DelPiece(int sq, int pc) { // 从棋盘上拿走一枚棋子
2                  ucpcSquares[sq] = 0;
3                  if (pc < 16) {// 红方减分，黑方(注意"cucvlPiecePos"取值要相反)加分
4                    vlWhite -= cucvlPiecePos[pc - 8][sq];      //将其子力价值从红方中减去
5                    } else {
6                    vlBlack -= cucvlPiecePos[pc - 16][SQUARE_FLIP(sq)];
                                  //将其棋子对应子力价值从红方局面对应的总子力价值中减去
7                  }}
```

（3）撤销走一步棋方法 UndoMakeMove，其详细代码如下。

```
1    public static void UndoMakeMove(int mv, int pcCaptured) {     // 撤销走一步棋
```

```
2                nDistance --;                                  //走棋步数自减
3                ChangeSide();                                  //交换走棋子
4                UndoMovePiece(mv, pcCaptured);                 //撤销走棋子
5            }
```

（4）搬一步棋的棋子方法 MovePiece，其详细代码如下。

```
1    public static int MovePiece(int mv) {                      // 搬一步棋的棋子
2          int sqSrc, sqDst, pc, pcCaptured;
3          sqSrc = SRC(mv);                                     //得到起始位置的数组下标
4          sqDst = DST(mv);                                     //得到目标位置的数组下标
5          pcCaptured = ucpcSquares[sqDst];                     //得到目的格子的棋子
6          if (pcCaptured != 0) {                               //目的位置不为空
7            DelPiece(sqDst, pcCaptured);                       //删掉目标格子棋子
8          }
9          pc = ucpcSquares[sqSrc];                             //得到初始格子上的棋子
10         DelPiece(sqSrc, pc);                                 //删掉初始格子上的棋子
11         AddPiece(sqDst, pc);                                 //在目标格子上放上棋子
12         return pcCaptured;                                   //返回原来目标格子上的棋子
13       }
```

（5）撤销搬一步棋的棋子方法 UndoMovePiece，其详细代码如下。

```
1    public static void UndoMovePiece(int mv, int pcCaptured) {// 此方法功能为悔棋一步
2          int sqSrc, sqDst, pc;
3          sqSrc = SRC(mv);                                     //得到起始格子
4          sqDst = DST(mv);                                     //得到目标格子
5          pc = ucpcSquares[sqDst];                             //得到目标格子上的棋子
6          DelPiece(sqDst, pc);                                 //删除目标格子上的棋子
7          AddPiece(sqSrc, pc);                                 //在起始格子上放上棋子
8          if (pcCaptured != 0) {                               //如果目标格子上起初有棋子
9            AddPiece(sqDst, pcCaptured);                       //将该棋子放在目标格子上
10         }}
```

（6）走一步棋的方法 MakeMove，其详细代码如下。

```
1    public static boolean MakeMove(int mv, int pcCaptured) { // 走一步棋
2          pcCaptured = MovePiece(mv);                        //将起始位置的棋子搬到目标格子上
3          if (Checked()) {                                     //如果将军
4            UndoMovePiece(mv, pcCaptured);                     //撤销此步棋
5            return false;                                      //返回将军
6          }
7          ChangeSide();                                        //交换走子方
8          nDistance ++;                                        //走棋步骤自动加 1
9          return true;                                         //返回为被将军
10       }
```

（7）生成所有走法方法 GenerateMoves，其详细代码如下。

```
1    public static int GenerateMoves(Integer[] mvs)  {// 生成所有走法
2          int i, j, nGenMoves, nDelta, sqSrc, sqDst;
3          int pcSelfSide, pcOppSide, pcSrc, pcDst;
4          nGenMoves = 0; // 生成所有走法，需要经过以下几个步骤:
5          pcSelfSide = SIDE_TAG(sdPlayer);            // 获得红黑标记(红子是 8，黑子是 16)
6          pcOppSide = OPP_SIDE_TAG(sdPlayer);                  // 获得对方红黑标记
7          for (sqSrc = 0; sqSrc < 256; sqSrc ++) {             //循环整个数组
8            pcSrc = ucpcSquares[sqSrc];               // 1. 找到一个本方棋子，再做以下判断:
9            if ((pcSrc & pcSelfSide) == 0) {                   //如果该棋子不是自己的
```

```
10            continue;                                          //则返回
11          }                                                    //找到一个自己的棋子了
12          switch (pcSrc - pcSelfSide) {                        // 2. 根据棋子确定走法
13          case PIECE_KING:                                     //如果是王
14            for (i = 0; i < 4; i ++) {                         //可走 4 步
15              sqDst = sqSrc + ccKingDelta[i];        //根据步长得到可走的目标格子
16              if (!IN_FORT(sqDst)) {                           //如果不在九宫内
17                continue;}                                     //结束该次
18              pcDst = ucpcSquares[sqDst];                      //得到终点棋子
19              if ((pcDst & pcSelfSide) == 0) {       //如果目标格子上的棋子不是自方棋子
20                mvs[nGenMoves] = MOVE(sqSrc, sqDst);           // 得到一个走法
21                nGenMoves ++;                          //存放走棋步骤的数组下标自动加 1
22              } }
23            break;
24          case PIECE_ADVISOR:                                  //如果是士
25            for (i = 0; i < 4; i ++) {                         //4 个方向可走
26              sqDst = sqSrc + ccAdvisorDelta[i];               //得到目标格子
27              if (!IN_FORT(sqDst)) {                           //如果不在九宫内则返回
28                continue;  }
29              pcDst = ucpcSquares[sqDst];                      //得到目标格子上的棋子
30              if ((pcDst & pcSelfSide) == 0) {                 //如果该棋子不是自己的棋子
31                mvs[nGenMoves] = MOVE(sqSrc, sqDst);           //将走棋步骤存入
32                nGenMoves ++;
33              }}
34            break;
35          case PIECE_BISHOP:                                   //如果是相
36            for (i = 0; i < 4; i ++) {                         //同样 4 个方向可走
37              sqDst = sqSrc + ccAdvisorDelta[i];     //得到田心格子对应的数组下标
38              if (!(IN_BOARD(sqDst) && HOME_HALF(sqDst, sdPlayer) && ucpcSquares[sqDst]
                == 0)) {
39                continue; }//如果目标格子不在棋盘内或者过河了或者田心有棋子则都不符合相的走棋规则
40              sqDst += ccAdvisorDelta[i];            //得到目标棋子对应的数组下标
41              pcDst = ucpcSquares[sqDst];                      //得到目标格子上的棋子
42              if ((pcDst & pcSelfSide) == 0) {       //如果目标格子上的棋子不是自己的
43                mvs[nGenMoves] = MOVE(sqSrc, sqDst);           //得到一个走法
44                nGenMoves ++;
45              }}
46            break;
47          case PIECE_KNIGHT:                                   //如果是马
48            for (i = 0; i < 4; i ++) {                         //4 个方向
49              sqDst = sqSrc + ccKingDelta[i];        //得到马腿格子对应数组的下标
50              if (ucpcSquares[sqDst] != 0) {                   //如果马腿上有棋子则返回
51                continue; }
52              for (j = 0; j < 2; j ++) {                       //每个方向上的左右方可走
53                sqDst = sqSrc + ccKnightDelta[i][j];     //得到目标格子对应的数组下标
54                if (!IN_BOARD(sqDst)) {                  //如果目标格子不在棋盘中
55                  continue;}
56                pcDst = ucpcSquares[sqDst];              //得到目标格子对应数组下标
57                if ((pcDst & pcSelfSide) == 0) {         //如果目标格子上的棋子不是自己的
58                  mvs[nGenMoves] = MOVE(sqSrc, sqDst);   //得到一个走法
59                  nGenMoves ++;
```

```
60                } }}
61              break;
62            case PIECE_ROOK:                                  //如果是车
63              for (i = 0; i < 4; i ++) {                      //4 个方向可走
64                nDelta = ccKingDelta[i];            //得到一个方向上的步长，该步长为王的
65                sqDst = sqSrc + nDelta;                       //得到目标格子对应数组下标
66                while (IN_BOARD(sqDst)) {           //循环，如果该格子不在棋盘内则结束循环
67                  pcDst = ucpcSquares[sqDst];                 //得到目标格子上的棋子
68                  if (pcDst == 0) {                           //如果目标格子上的棋子为空
69                      mvs[nGenMoves] = MOVE(sqSrc, sqDst);//则可行，生成一个走法
70                      nGenMoves ++;
71                  } else {                                    //否则在存在棋子的情况下
72                    if ((pcDst & pcOppSide) != 0) {           //如果目标格子是对方的棋子
73                      mvs[nGenMoves] = MOVE(sqSrc, sqDst);//则生成一个走法
74                      nGenMoves ++;
75                    }                                 //车不可越子，该方向上遇到第一颗棋子
76                    break;                                    //在该方向上就结束
77                  }
78                  sqDst += nDelta;                            //目标格子加上步长
79                } }
80              break;
81            case PIECE_CANNON:                                //如果是炮
82              for (i = 0; i < 4; i ++) {                      //4 个方向可走
83                nDelta = ccKingDelta[i];                      //其步长为王的步长
84                sqDst = sqSrc + nDelta;                       //得到目标格子对应数组下标
85                while (IN_BOARD(sqDst)) {                     //如果目标格子在棋盘上
86                  pcDst = ucpcSquares[sqDst];                 //得到目标格子上的棋子
87                  if (pcDst == 0) {                           //如果目标格子上没有棋子
88                    mvs[nGenMoves] = MOVE(sqSrc, sqDst);  //得到一个走法
89                    nGenMoves ++;
90                  } else {                                    //否则结束
91                    break;
92                  }                                           //如果在该方向上没有遇到棋子
93                  sqDst += nDelta;                  //进入下一循环，查看该方向上下一格
94                }                                             //如果遇到棋子，则跳出循环
95                sqDst += nDelta;                              //查看翻山后该方向上的情况
96                while (IN_BOARD(sqDst)) {                     //目标格子在棋盘内
97                  pcDst = ucpcSquares[sqDst];                 //得到目标格子上的棋子
98                  if (pcDst != 0) {                           //如果目标格子上有棋子
99                    if ((pcDst & pcOppSide) != 0) {           //如果该棋子是对方的
100                     mvs[nGenMoves] = MOVE(sqSrc, sqDst);//得到一个走法
101                     nGenMoves ++;
102                   }
103                   break;                                    //翻山后遇到棋子就结束
104                 }
105                 sqDst += nDelta;                            //没有遇到棋子，进行查看下一格
106               } }
107             break;
108           case PIECE_PAWN:                                  //如果是兵或者卒
109             sqDst = SQUARE_FORWARD(sqSrc, sdPlayer);        //格子水平镜像，将黑红方统一
110             if (IN_BOARD(sqDst)) {                          //如果目标格子是在棋盘内
111               pcDst = ucpcSquares[sqDst];                   //得到目标格子上棋子
112               if ((pcDst & pcSelfSide) == 0) {              //如果目标格子上的棋子不是自己的
113                 mvs[nGenMoves] = MOVE(sqSrc, sqDst);        //得到一个走法
114                 nGenMoves ++;
115               }}
116             if (AWAY_HALF(sqSrc, sdPlayer)) {               //如果已经过河
117               for (nDelta = -1; nDelta <= 1; nDelta += 2) {     //3 个方向可走
118                 sqDst = sqSrc + nDelta;                     //得到目标格子对应数组下标
119                 if (IN_BOARD(sqDst)) {                      //如果该格子在棋盘内
120                   pcDst = ucpcSquares[sqDst];               //得到目标格子上的棋子
121                   if ((pcDst & pcSelfSide) == 0) {          //如果该棋子不是自己的棋子
```

```
122                     mvs[nGenMoves] = MOVE(sqSrc, sqDst);//得到一个走法
123                     nGenMoves ++;
124                   }}}}
125             break;
126           } }
127         return nGenMoves;                                  //返回装有所有走法的数组
128       }
```

- 第 13～23 行为生成王的所有走法，根据步长得到目标格子数组下标，然后判断其是否在棋盘内并且在九宫之内，判断其目标格子是不是自己的棋子，条件均满足则生产一个走法。
- 第 24～34 行为生成士的所有走法，该生成走法原理和王相似。
- 第 35～46 行为生成相的所有走法，目标格子要在棋盘上，其田心不能有棋子，目标格子上的棋子不能是自己的。
- 第 47～61 行为生成马的所有走法，其特殊规则是马腿上不能有其他棋子。
- 第 62～80 行为生成车的所有走法，车直走但是不能越子而行。
- 第 81～107 行为生成炮的所有走法，炮在没有遇到棋子前，走法和车相同，如果要越子而行，必须是要能吃掉对方的棋子。
- 第 108～126 行为生成兵的所有走法，兵在过河前只能向前走一步，过河后可走 3 个方向，即向前和向左或者向右，均只能走一步。

（8）判断走法是否合理的方法 LegalMove，其详细代码如下。

```
1     public static boolean LegalMove(int mv)  {              // 判断走法是否合理
2           int sqSrc, sqDst, sqPin;
3           int pcSelfSide, pcSrc, pcDst, nDelta;             // 1. 判断起始格是否有自己的棋子
4           sqSrc = SRC(mv);                                        //得到起始格子
5           pcSrc = ucpcSquares[sqSrc];                             //得到目标格子
6           pcSelfSide = SIDE_TAG(sdPlayer);                // 获得红黑标记(红子是 8，黑子是 16)
7           if ((pcSrc & pcSelfSide) == 0) {                        //如果不是自己的棋子
8             return false;                                         //则走法不合理
9           }                                               // 2. 判断目标格是否有自己的棋子
10          sqDst = DST(mv);                                        //得到目标格子
11          pcDst = ucpcSquares[sqDst];                             //得到目标格子的棋子
12          if ((pcDst & pcSelfSide) != 0) {                        //如果是自己的棋子
13            return false;                                         //则走法不合理
14          }
15          switch (pcSrc - pcSelfSide) {                   // 3. 根据棋子的类型检查走法是否合理
16          case PIECE_KING:                                        //如果是王
17            return IN_FORT(sqDst) && KING_SPAN(sqSrc, sqDst);//目标格子在九宫并且步长符合
18          case PIECE_ADVISOR:                                     //如果是士
19            return IN_FORT(sqDst) && ADVISOR_SPAN(sqSrc, sqDst);
                                                            //目标格子在九宫并且步长符合
20          case PIECE_BISHOP:                                      //如果是相
21            return SAME_HALF(sqSrc, sqDst) && BISHOP_SPAN(sqSrc, sqDst) &&//目标格子未过河
22                ucpcSquares[BISHOP_PIN(sqSrc, sqDst)] == 0;   //步长符合且田心无棋子
23          case PIECE_KNIGHT:                                      //如果是马
24            sqPin = KNIGHT_PIN(sqSrc, sqDst);                     //得到马腿的位置
25            return sqPin != sqSrc && ucpcSquares[sqPin] == 0;
                                                        //马腿位置不对或者马腿有子不合理
26          case PIECE_ROOK:                                        //车
27          case PIECE_CANNON:                                      //炮
28            if (SAME_RANK(sqSrc, sqDst)) {                        //如果在同一行
29              nDelta = (sqDst < sqSrc ? -1 : 1);                  //得到步长
30            } else if (SAME_FILE(sqSrc, sqDst)) {                 //如果在同一列
31              nDelta = (sqDst < sqSrc ? -16 : 16);                //得到步长
```

```
32            } else {
33              return false;                                    //否则为不合理
34            }
35            sqPin = sqSrc + nDelta;                  //得到起始格子和目标格子之间的格子
36            while (sqPin != sqDst && ucpcSquares[sqPin] == 0) {  //从第一个开始循环
37              sqPin += nDelta;                       //如果没有遇到棋子，则检测下一格
38            }
39            if (sqPin == sqDst) {                            //如果最后到了目标格子
40              return pcDst == 0 || pcSrc - pcSelfSide == PIECE_ROOK;
                                                     //目标格子为空或者，该棋子是车
41            } else if (pcDst != 0 && pcSrc - pcSelfSide == PIECE_CANNON) {
                                                     //该格子不为空，，
42              sqPin += nDelta;                               //查看下一个格子
43              while (sqPin != sqDst && ucpcSquares[sqPin] == 0) {
                                                     //查看炮是否翻山，该格子不是目标格子
44                sqPin += nDelta;                             //查看下一个格子
45              }                                              //直到到达目标格子为止
46              return sqPin == sqDst;               //如果没有到达目标格子，则不合理
47            } else {
48              return false;                                  //否则不合理
49            }
50          case PIECE_PAWN:                                   //如果是兵
51            if (AWAY_HALF(sqDst, sdPlayer) && (sqDst == sqSrc - 1 || sqDst == sqSrc +
1)) {
52              return true;                                   //如果已经过河，是横向走一步
53            }
54            return sqDst == SQUARE_FORWARD(sqSrc, sdPlayer); //目标格子为向前走一步
55          default:
56            return false;
57          }}
```

- 该部分首先查看起始格子是否为自己的棋子，然后判断目标格子是不是自己的棋子，最后才根据不同棋子判断其走法是否合理。其判断方法，均为查看其步长是否合理，是否符合走棋规则。
- 对于相则查看其不能过河，田心不能有棋子；对于马，则马腿不能有棋子；对于车则首先判断其方向，在其方向上，目标格子和起始格子之间不能有棋子，对于炮则分两种情况，翻山吃棋子和走棋不吃棋子，不吃棋子的情况和车相同，吃棋子，则只看其中间是否只有一颗棋子。详细情况请见注释。

（9）判断是否被将军的方法 Checked，该方法是先找到王，然后将王按照各个棋子的走棋规则走棋，如果走到的地方是该棋子，说明被该棋子将军。其详细代码如下。

```
1     public static boolean Checked()  {                  // 判断是否被将军
2           int i, j, sqSrc, sqDst;
3           int pcSelfSide, pcOppSide, pcDst, nDelta;
4           pcSelfSide = SIDE_TAG(sdPlayer);              // 获得红黑标记(红子是 8，黑子是 16)
5           pcOppSide = OPP_SIDE_TAG(sdPlayer);           //获得对方的红黑标记
6           for (sqSrc = 0; sqSrc < 256; sqSrc ++) { // 找到棋盘上的帅(将)，再做以下判断:
7             if (ucpcSquares[sqSrc] != pcSelfSide + PIECE_KING) {
8               continue;
9             }                                           // 1. 判断是否被对方的兵(卒)将军
10            if (ucpcSquares[SQUARE_FORWARD(sqSrc, sdPlayer)] == pcOppSide + PIECE_PAWN)
{
11              return true;                              //是否被过河兵向前走时将军
12            }
13            for (nDelta = -1; nDelta <= 1; nDelta += 2) {//是否被过河兵向左向右走时将军
14              if (ucpcSquares[sqSrc + nDelta] == pcOppSide + PIECE_PAWN) {
15                return true;
16              } }
```

```
17          for (i = 0; i < 4; i ++) {// 2. 判断是否被对方的马将军(以仕(士)的步长当作马腿)
18            if (ucpcSquares[sqSrc + ccAdvisorDelta[i]] != 0) {  //如果马腿处有棋子
19              continue;
20            }
21            for (j = 0; j < 2; j ++) {                  //每个方向有左右两种情况
22              int pcDstt = ucpcSquares[sqSrc + ccKnightCheckDelta[i][j]];
                                                          //得到目标格子处的棋子
23              if (pcDstt == pcOppSide + PIECE_KNIGHT) {//如果该棋子为马，说明被马将军
24                return true;
25              }}}
26          for (i = 0; i < 4; i ++) {  // 3. 判断是否被对方的车或炮将军(包括将帅对脸)
27            nDelta = ccKingDelta[i];                          //得到步长值
28            sqDst = sqSrc + nDelta;                           //得到目标格子
29            while (IN_BOARD(sqDst)) {                         //目标格子在棋盘内
30              pcDst = ucpcSquares[sqDst];                     //得到目标格子处的棋子
31              if (pcDst != 0) {                               //如果目标格子有棋子
32                if (pcDst == pcOppSide + PIECE_ROOK || pcDst == pcOppSide + PIECE_KING)
{
33                  return true;                          //如果目标格子上的棋子是车或者王
34                }
35                break;
36              }                                       //遇到一个棋子后，还未被车或者王将军
37              sqDst += nDelta;                                //目标格子到下一格
38            }                                                 //判断是否被炮将军
39            sqDst += nDelta;                                  //目标格子到下一格
40            while (IN_BOARD(sqDst)) {                         //目标格子在棋盘内
41               pcDst = ucpcSquares[sqDst];                    //得到目标格子上的棋子
42              if (pcDst != 0) {                               //如果不为空
43                if (pcDst == pcOppSide + PIECE_CANNON) {      //如果是炮，则被炮将军
44                  return true;
45                } break;
46              }
47              sqDst += nDelta;                                //目标格子到下一格
48            } }
49          return false;                                       //否则没有被将军
50        } return false;                                       //否则没有被将军
51      }
```

- 第 6～9 行为循环整个棋盘数组，直到找到王为止。找到王之后将王分别按照兵、马、车和炮的走法规则去查看走到的地方是不是对应的棋子，如果是则说明被相应的棋子将军。
- 第 10～16 行为判断是否被对方的兵将军，分两种情况，一种是被兵直走将军，另一种是被兵横走时将军。
- 第 17～25 行为判断是否被对方的马将军，同样将王按照马的走棋规则走棋，不过马脚变换了一下位置。然后查看王走到的位置是否为马，如果是则被马将军。
- 第 26～48 行为判断被炮或者车将军。同上的判断，把王按照车和炮的走棋规则进行走棋，如果能走到的地方是车或者炮，这当中还有一个规则就是，双方的王不能照面，如果照面，则对方的王可以像车一样直走吃自己的王。所以，将车将军情况和王照面情况用同样的规则判断。

（10）判断是否被杀方法 IsMate。该方法在该局面下生成所有的走法，每一个走法走完后，判断是否被将军，如果所有的走法下都被将军，说明该方已经被将死，即被杀。当然，只要遇到一个走法下没有被将军，则说明还没有被将死，即没有被杀。其详细代码如下。

```
1    public static boolean IsMate() {                        // 判断是否被杀（将军）
2          int i, nGenMoveNum, pcCaptured ;
3          Integer mvs[]=new Integer[MAX_GEN_MOVES];         //创建存放所有走法的数组
```

```
4            nGenMoveNum = GenerateMoves(mvs);                    //生成所有的走法并得到走法数
5            for (i = 0; i < nGenMoveNum; i ++) {                 //循环走每一步棋
6                 pcCaptured = MovePiece(mvs[i]);                 //走该走法
7              if (!Checked()) {                                  //如果没有被将军
8                UndoMovePiece(mvs[i], pcCaptured);               //撤销该走法
9                return false;                                    //返回没有被杀
10             } else {                                           //否则为被将军
11               UndoMovePiece(mvs[i], pcCaptured);               //撤销走法
12             } }
13           return true;                                         //如果循环顺利结束了，说明被杀了
14         }
```

说明　该方法首先生成所有走法，然后每走一个走法判断其是否被将军，如果中途遇到一个走法没有被将军，说明还没有被将死，则返回没有被杀。如果循环顺利结束，即所有走法下都是被将军的，则说明被杀。

（11）超出边界（Fail-Soft）的 Alpha-Beta 搜索过程，SearchFull 方法，这是一个 N 叉树的搜索，该方法根据搜索深度，在每一层生成所有走法，循环每一个走法，走一个走法后到下一个局面，再次搜索所有走法，如此递归调用，直到到达最大的搜索深度，然后根据此时的局面价值，将所有走法下的局面价值相互比较得到最好的走法。其详细代码如下。

```
1    public static int SearchFull(int vlAlpha, int vlBeta, int nDepth) {
2            int i=0, nGenMoves=0,pcCaptured=0;
3            int vl, vlBest, mvBest;
4            Integer mvs[]=new Integer[MAX_GEN_MOVES]; //存放走法的数组
5            if (nDepth == 0) {                          // 1. 到达水平线，则返回局面评价值
6              return Evaluate();
7            }                                           // 2. 初始化最佳值和最佳走法
8            vlBest = -MATE_VALUE; // 这样可以知道，是否一个走法都没走过
9            mvBest = 0;       // 这样可以知道，是否搜索到了 Beta 走法或 PV 走法，以便保存到历史表中
10           nGenMoves = GenerateMoves(mvs);             // 3. 生成全部走法，并根据历史表排序
11           Arrays.sort(mvs, 0,nGenMoves,new MyComparator());//对走法进行排序，根据历史表排序
12           for (i = 0; i < nGenMoves; i ++) {              // 4. 逐一走这些走法，并进行递归
13                pcCaptured=ucpcSquares[DST( mvs[i])];     //得到目标格子上的棋子
14             if (MakeMove(mvs[i], pcCaptured)) {      //走该走法，判断是否被将军，如果没有
15               vl = -SearchFull(-vlBeta, -vlAlpha, nDepth - 1);    //搜索下一层
16               UndoMakeMove(mvs[i], pcCaptured); // 5. 进行 Alpha-Beta 大小判断和截断
17               if (vl > vlBest) {    // 找到最佳值(但不能确定是 Alpha、PV 还是 Beta 走法)
18                 vlBest = vl;     // "vlBest"就是目前要返回的最佳值，可能超出 Alpha-Beta 边界
19                 if (vl >= vlBeta) {                       // 找到一个 Beta 走法
20                   mvBest = mvs[i];                        // Beta 走法要保存到历史表中
21                   break;                                  // Beta 截断
22                 }
23                 if (vl > vlAlpha) {                       // 找到一个 PV 走法
24                   mvBest = mvs[i];                        // PV 走法要保存到历史表中
25                   vlAlpha = vl;                           // 缩小 Alpha-Beta 边界
26                 } } } }
27               // 6. 所有走法都搜索完了，把最佳走法(不能是 Alpha 走法)保存到历史表中，返回最佳值
28               if (vlBest == -MATE_VALUE) {
```

```
29            return nDistance - MATE_VALUE;      //如果是杀棋，就根据杀棋步数给出评价
30          }
31              if (mvBest != 0) {  // 如果不是 Alpha 走法，就将最佳走法保存到历史表中
32               nHistoryTable[mvBest] += nDepth * nDepth;
33             // 搜索根节点时，总是有一个最佳走法(因为全窗口搜索不会超出边界)，将这个走法保存下来
34             if (nDistance == 0) {
35             mvResult = mvBest;
36            } }
37          return vlBest;
38        }
```

- 第 5～6 行为当搜索深度达到水平线后就返回局面评价值。
- 第 10～11 行为产生在当前局面下该方的所有走法，然后将所有走法按照历史表排序，排完序后，在以前节点下搜索到的最好的走法排在前面，这样可以被先搜索，从而产生截断等，减少多余的搜索。
- 第 12～30 行为循环走所有走法，每种走法走完后进行界面评估搜索得到界面评价值，找到该局面下最好的走法，其中第 15 行为，通过下一层搜索得到界面评价值，而其参数之所以如此变换，是利用了最好最坏搜索算法，对于一方最好的局面值，那么对另外一方就是最差的值，到下一层后走棋方会交换。所以最好最坏值也交换并取相反数。
- 第 17～26 行为查看该走法下的评价值是否比其他走法的评价值高，比对方的评价值高，如果是则保存该走法。第 25 行为缩小 Alpha-Beta 边界，如果搜索到更好的走法，即更好的局面值，则将搜索到的最好值替换掉。
- 第 28～30 行为在所有走法下都被将军的情况下，说明该局面为杀棋，那么根据搜索深度给出局面评价。
- 第 31～36 行为如果找到一个不是 alpha 走法的，则将其走法保存到历史表中。如果此局面为搜索的根节点，则对于此局面肯定有一个最好的走法，将其保存。

（12）迭代加深搜索过程 SearchMain 方法，该方法对局面进行迭代加深搜索，从一层开始到最大搜索层，这样可以避免那种用搜索固定深度时，在少数深度就可以搜索到杀棋后时，仍然要去搜索剩下的层数的处理是多余的，浪费时间也浪费资源。而用迭代加深搜索有力地避免了这一点。搜索到了杀棋就直接跳出，不再加深搜索深度去搜索。其详细代码如下。

```
1    public static void SearchMain() {                          // 迭代加深搜索过程
2          int i, vl;
3          initHistorytable();                                  // 初始化
4          nDistance = 0;                                       // 初始步数
5          long start=System.nanoTime();
6          for (i = 1; i <= LIMIT_DEPTH; i ++) {                // 迭代加深过程
7            vl = SearchFull(-MATE_VALUE, MATE_VALUE, i);
8            if (vl > WIN_VALUE || vl < -WIN_VALUE) {           // 搜索到杀棋，就终止搜索
9              break;
10           }
11           if((System.nanoTime()-start)/1.e9>ViewConstant.thinkDeeplyTime)
                                                                // 超过时间，就终止搜索
12           break;
13         }}
```

13.7.4　走法的排序规则类 MyComparator

该类为，在进行 Alpha-Beta 搜索过程时，对其局面下的所有走法进行排序，排序依据为历史表，历史表数组的下标代表走法，其值为在该步骤下的深度有关值，所以，每次搜索时就可以先搜索最好的走法。可以减少对后面的搜索次数。其详细代码如下。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 MyComparator.java。

```
1    package com.bn.chess;
2    import static com.bn.chess.LoadUtil.*;
3    import java.util.Comparator;
4    public class MyComparator implements Comparator<Integer>{
5         @Override
6         public int compare(Integer arg0, Integer arg1) {
7              Integer i1=(Integer)arg0;
8              Integer i2=(Integer)arg1;
9              return nHistoryTable[i2.intValue()]-nHistoryTable[i1.intValue()];
10        }}
```

13.7.5　记录走棋步骤类 StackplayChess

该类为记录走法的类，将对弈时，走法和在该走法下目标位置的棋子记录起来，放入栈，在悔棋时取出，撤销该走法即可。

代码位置：见随书光盘中源代码/第 13 章 Chess_DJB/src/com/bn/chess 目录下的 StackplayChess.java。

```
1    package com.bn.chess;
2    public class StackplayChess {
3         int pcCaptured;
4         int mvResult;
5         public StackplayChess(int mvResult,int pcCaptured){
6              this.pcCaptured=pcCaptured;
7              this.mvResult=mvResult;
8         }}
```

13.8　游戏的优化与改进

虽然本游戏基本框架已经开发完毕，但是还有许多优化与改进的地方。

- 游戏界面的美化

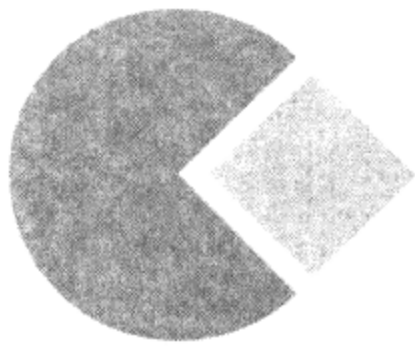
虽然游戏整体场景已经相当绚丽，但是，读者仍然可以对游戏界面做进一步的美化。如游戏中的棋盘，其背景可以做的更加美观，棋盘的颜色搭配也可以重新美化。

- 游戏操作方式的改进

本游戏的操作方式采用屏幕触摸来操控，可能在很多细节方面还不是很到位，坐标计算的精确度还不够好，对按钮的点击效果也不是很绚丽，希望读者能自行对其改善。

- 走法引擎仅是初级的人工智能

虽然该引擎的人工智能已经算不错，但是第一步计算机走棋还是会一如既往的跳马。所以，需要进一步优化。有时会出现重复局面，计算机无止境的将军，需要用到长将判负。没有用到开局库和残局库，没有用置换表技术等，若玩家的棋力比较强,一直和程序对弈到残局阶段，那么本案例用的算法可能就不是十分有效，有能力的读者可以考虑为本应用程序升级，增加开局库、残局库等，进一步优化程序。



第 14 章　物理传感器游戏——小球快跑

本章要介绍的游戏比较特殊，是采用 3D 物理传感器技术来实现的，Android 平台下支持 3D 物理传感器的开发，相关知识在前面的第 4 章已经有所介绍。本章就以小球快跑这款游戏的开发来说明，如何基于物理传感器开发出具有新玩法的手机游戏。

14.1 游戏背景及功能概述

本节将对游戏背景进行简单介绍，然后简要说明一下游戏的功能，这样可以使读者对游戏有一个整体的印象，以便更好地理解游戏的开发过程。

14.1.1　背景概述

单从游戏内容上来看，本游戏是一款简单的移动小球过关的游戏，这属于比较简单的动作类闯关游戏。但从操作方式上来看，本游戏便是一款非常新潮的传感器游戏。

基于传感器技术开发出的游戏，使得游戏的玩法从原来的手机按键或触摸操作改为了更加自由的甩、摇、转等操作方式。这种在游戏操作方式上的创新可以给玩家带来更好的体验。

14.1.2　功能简介

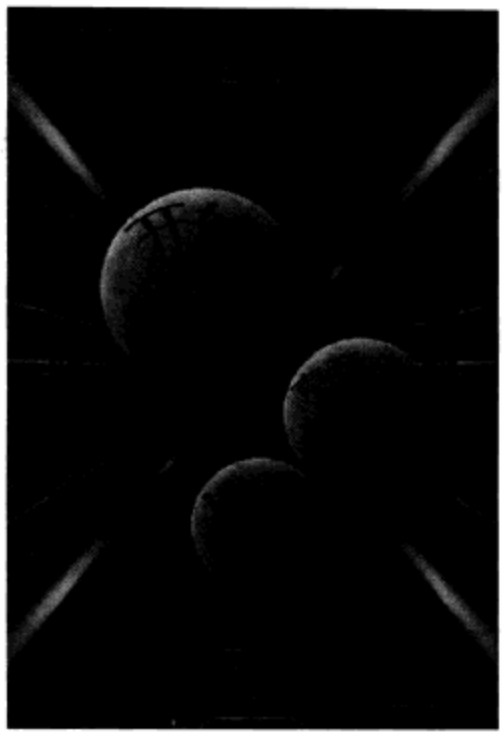
介绍游戏的功能之前有必要先了解一下游戏的玩法，在游戏中，小球受到重力的作用，通过转动手机来让小球沿着游戏中可通过的地方移动，最终达到目的地。期间小球会遇到多种障碍，如陷阱、时通时断的路、大炮等。

（1）运行游戏后，首先进入的是欢迎界面，欢迎界面由 3 个球形按钮组成，背景为穿梭隧道的动画。如图 14-1 所示。玩家可以单击开始、退出和声音按钮，当选择开始后，球形按钮会逐渐旋转变小，同时其他按钮消失，屏幕显示“加载中...”，如图 14-2 所示，加载结束后进入游戏界面，如图 14-3 所示。

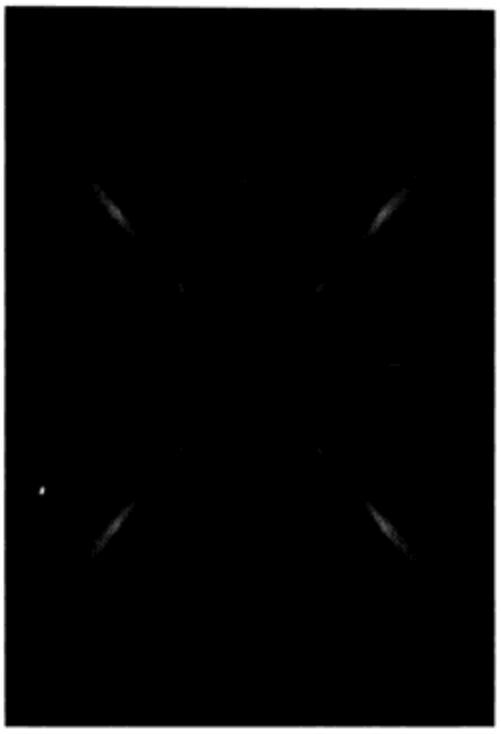
（2）在游戏界面中，玩家可以通过改变手机的方位来控制小球移动。游戏界面的背景为逐渐移动的星空。

当玩家控制小球到达目的地时，游戏会提示“太棒了，单击进入下一关”字样，如图 14-4 所示，单击游戏提示即可进入下一关的冒险。如果玩家成功闯过所有关，游戏会显示“恭喜过关”的提示，如图 14-5 所示，单击“恭喜过关”的游戏提示会返回到主菜单。

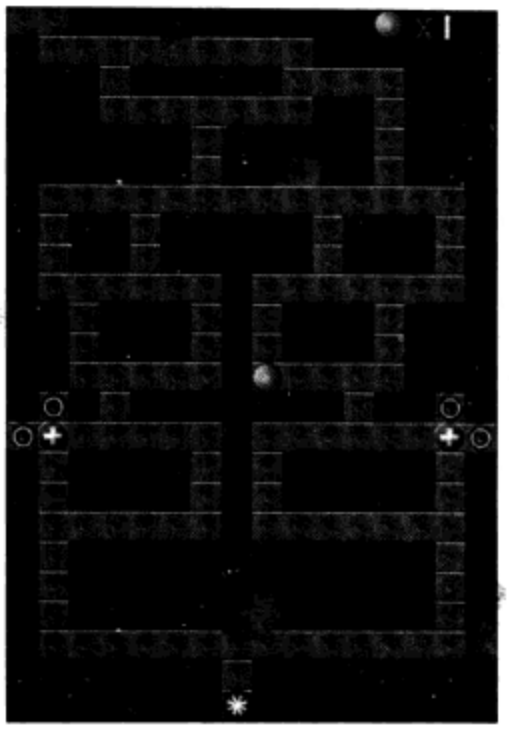
在游戏界面中有很多障碍物，如陷阱、大炮等，当小球不小心遇到这些障碍物时，其生命数会减 1，并且小球从最初的位置重新开始过关。当小球的生命耗尽时，游戏会提示“很遗憾，小球消失在了茫茫宇宙之中...”字样，如图 14-6 所示，单击游戏提示返回主菜单。



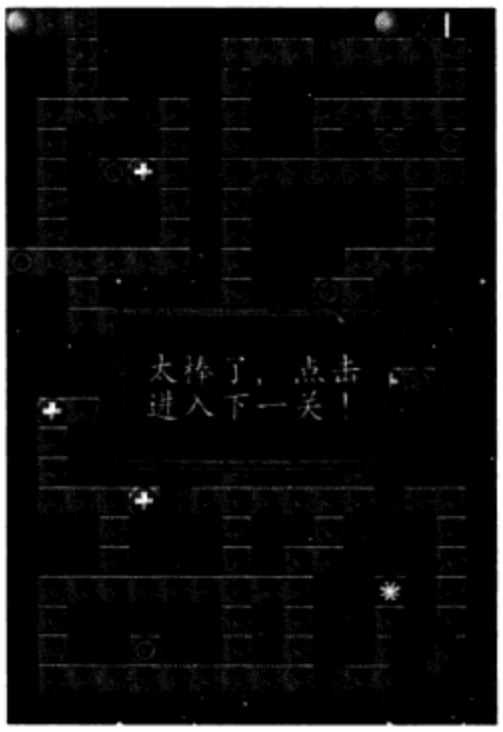
▲图 14-1　游戏欢迎界面



▲图 14-2　加载界面

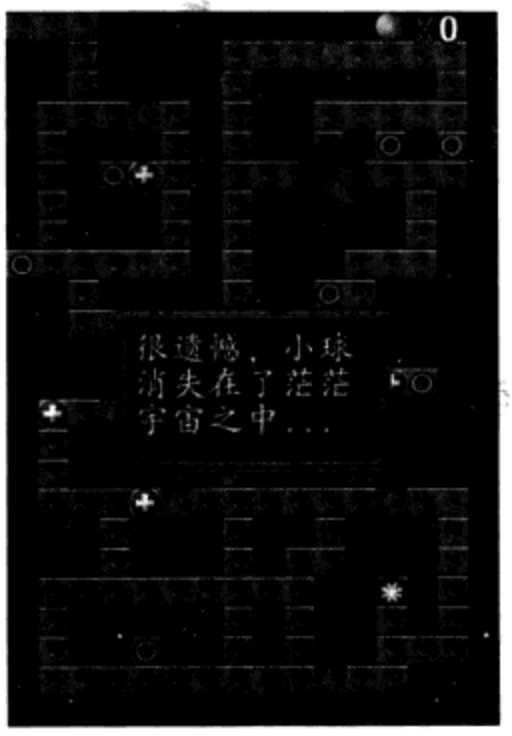


▲图 14-3　游戏界面



▲图 14-4　通过一关示意图

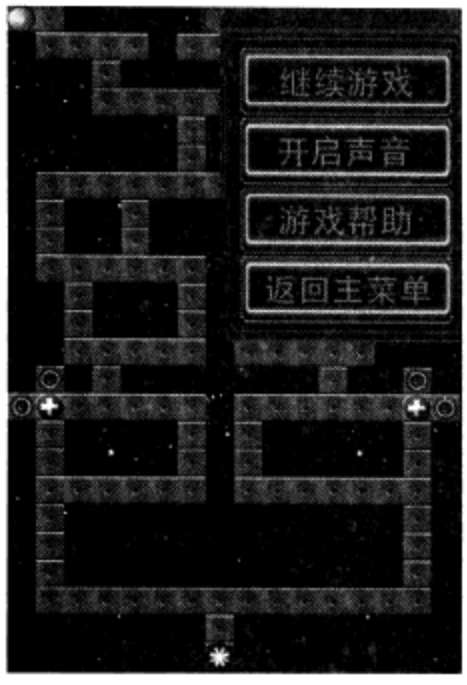
▲图 14-5　打通全部关卡示意图



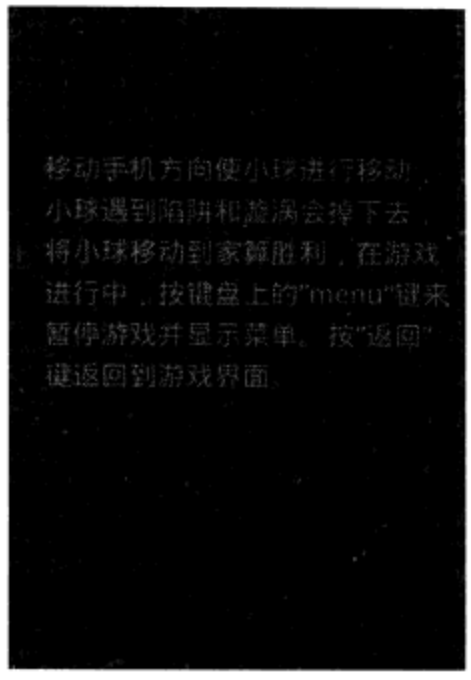
▲图 14-6　游戏失败示意图

（3）在游戏过程中，单击手机键盘上的 MENU 按键会弹出游戏菜单，如图 14-7 所示。在游戏菜单中可以选择继续游戏、开启（关闭）声音、游戏帮助、返回主菜单。如果选择游戏帮助，则会进入如图 14-8 所示的帮助界面。

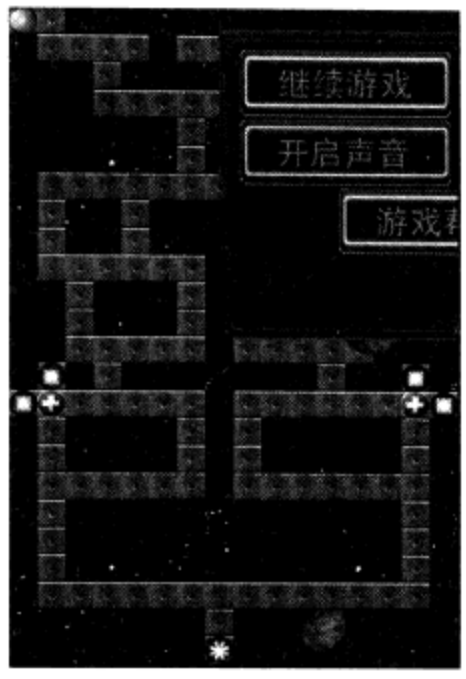
在游戏菜单中单击“继续游戏”按钮将返回到游戏界面，游戏菜单的出现和消失都是从右侧逐行滑入（出）的，如图 14-9 显示的是菜单逐行滑出的效果。



▲图 14-7　游戏菜单示意图



▲图 14-8　游戏帮助界面



▲图 14-9　菜单逐行移出效果

14.2 游戏的策划及准备工作

介绍完游戏的背景和功能概述后，下面运行游戏的策划及准备工作。对游戏进行合理的策划和充分地准备可以使开发过程更加顺利。

14.2.1　游戏的策划

本游戏从类型上来分属于动作类通关游戏，因此，在游戏开发前进行的策划中需要考虑到诸如剧情规划、关卡设计等内容。

- 剧情规划

由于本游戏主要是以玩法取胜，而不是靠游戏情结来吸引玩家，所以游戏剧情的规划可以尽量简单或者干脆没有。

- 关卡设计

关卡设计是本游戏的一个重点，在设计关卡的时候，如何保证难易程度的循序渐进和关卡之间的互异性以及关卡的可玩性是需要重视的问题。本游戏一共分为 5 关，通过设置不同障碍的有无、个数和位置来搭配不同的关卡。

- 目标受众

本游戏的操作方式十分简单，各个年龄阶层的玩家都可以快速掌握。

- 目标平台

本游戏的目标平台为 Android 2.1。

- 呈现技术

本游戏的游戏画面采用了图元技术，所以地图的设计也是一个重点。同时为了增强游戏的空间感，游戏中一共有 4 个图层，各个图层的渲染独立进行。

14.2.2　小球快跑游戏开发的准备工作

完成游戏的策划之后，就要为游戏开发进行一些准备工作了，在准备工作阶段，主要是搜集和制作一些游戏中会用到的图片和声音资源。游戏中会用到的图片资源如表 14-1 所示。

表 14-1　　游戏中用到的图片资源及说明

图 片 名	大小（KB）	像素（w×h）	用　　途	图 片 名	大小（KB）	像素（w×h）	用　　途
ball.png	1.17	20×20	小球图片	back_1～back_9.jpg	109	320×480	欢迎界面背景动画
cannon.png	0.9	20×20	大炮图片	eat_1～eat_4.png	4.44	20×20	漩涡动画帧
home.png	1.23	20×20	小球目的地	game_menu.png	2.97	180×250	游戏菜单背景
missile.png	0.4	10×10	炮弹图片	game_over.jpg	4.98	180×110	游戏结束提示
multiply.png	1.13	15×20	乘号图片	game_pass.png	4.89	180×110	游戏通全关提示
nebula.png	32.7	150×112	星云图片	meteo1～meteo8.png	45.4	65×60	陨石动画帧
plus_life.png	1.17	20×20	加命图片	number0～number9.png	3.65	16×24	数字图片
quit.png	5.32	90×90	退出按钮	trap1～trap8.png	8.09	20×20	陷阱动画帧
sound_off.png	6.87	102×101	关闭声音按钮	menu_back_to_main.png	2.85	180×48	返回主菜单选项
sound_on.png	6.91	103×101	开启声音按钮	menu_continue_game.png	3.12	180×48	继续游戏选项
stars.png	7.73	320×600	背景星空图	menu_help.png	3.0	180×48	帮助选项
start.png	9.9	135×135	开始按钮	menu_sound_off.png	2.75	180×48	关闭声音选项
tile.png	1.27	20×20	地图图元	menu_sound_on.png	2.68	180×48	开启声音选项
game_win.jpg	3.85	180×110	过关胜利提示				

将上述图片资源组织好，放置到项目的 res\drawable-mdpi 目录下。声音资源的搜集和制作工作也是在准备阶段进行的，本游戏会用到的声音资源及说明如表 14-2 所示。

表 14-2　　游戏中用到的声音资源及说明

声音文件	大小（KB）	格式	用　　途	声音文件	大小（KB）	格式	用　　途
Game_lose	7.04	mp3	损失一条命时播放	game_win	4.47	mp3	通关后播放
Game_music	20.08	mp3	游戏背景音乐	missile_hit	5.31	mp3	炮弹打中小球时播放
Break_out	4.08	mp3	菜单滑入、掉入陷阱时播放	plus_life	4.08	mp3	吃到加命奖励时播放

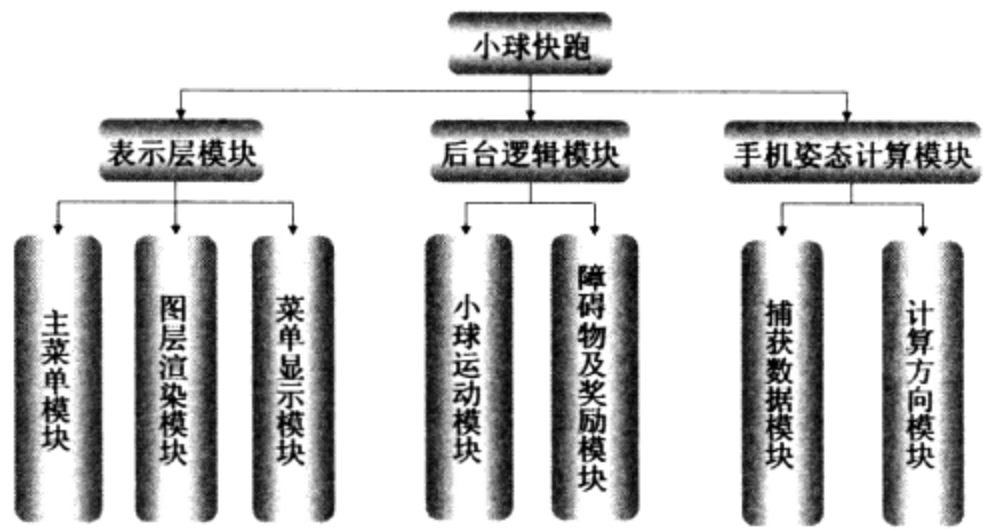
在项目的 res 目录下新建一个文件夹 raw，将上述声音文件放到 raw 目录下。至此，游戏的一些准备工作也已经完成了，下面将会逐步介绍游戏的架构及详细开发步骤。

14.3　游戏的架构

本节将来介绍游戏的架构，首先介绍游戏的总体架构，然后介绍游戏中的类结构。

14.3.1 游戏的总体架构

游戏的总体结构主要是游戏各个功能模块的结构，主要包括表示层模块、后台逻辑模块、手机姿态计算模块，各模块功能结构如图 14-10 所示。



▲图 14-10　游戏的总体架构图

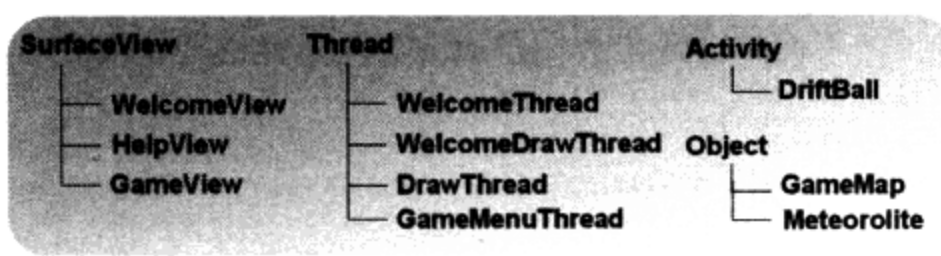
- 表示层模块负责游戏画面渲染工作，包括游戏主菜单的绘制、游戏界面中各个图层的绘制以及游戏中菜单的绘制。
- 后台逻辑模块主要用于对小球的运动情况进行控制，同时负责检测小球是否与障碍物及奖励物发生碰撞并进行相应处理。
- 手机姿态计算模块包括数据捕获模块和计算小球方向模块，捕获数据模块通过监听手机的姿态读取其状态变化，计算小球方向的模块则负责根据读取到的手机姿态进行计算，最后得出小球的运动方向。

14.3.2 游戏的类结构

下面介绍游戏的类结构，游戏的类结构涉及了具体的 Java 类，这些 Java 类将在一个或多个模块中扮演不同的角色。在进行代码开发时，也是大致依照游戏的类结构来进行开发的。下面是不同模块中的类结构。

1. 表示层模块的类结构

表示层模块的类结构如图 14-11 所示。表示层中涉及的类比较多，这些类或者为渲染画面提供数据模型，或者负责修改这些数据模型，或者负责绘制数据模型，下面对这些类进行简单的介绍。



▲图 14-11　表示层模块的类结构

- WelcomeView 类继承自 SurfaceView 类，主要负责播放欢迎动画和显示游戏主菜单。
- WelcomeDrawThread 和 WelcomeThread 均为 WelcomeView 的附属线程类，WelcomeDrawThread 负责定时重绘 WelcomeView，而 WelcomeThread 负责修改 Welcome 的绘制内容。
- HelpView 类比较简单，只负责显示帮助信息。
- GameView 类继承自 SurfaceView，主要负责游戏主画面的渲染，DrawThread 负责定时重绘 GameView。
- GameMenuThread 负责实现游戏中菜单的逐行滑入/滑出的效果，在游戏中每次单击菜单按钮就会启动一个 GameMenuThread。
- GameMap 类负责提供游戏中地图的信息，GameMap 中存储了所有关卡的信息，并对外提供了方法用于获得指定关卡的地图信息。
- Meteorolite 类继承自 Object，每个 Meteorolite 对象代表一个陨石，游戏界面中将会绘制若干个陨石。在 Meteorolite 类中封装了陨石的位置和动画帧索引等信息。
- DriftBall 类继承自 Activity，主要负责在不同的 View 之间进行切换，同时也负责处理用户单击屏幕事件。

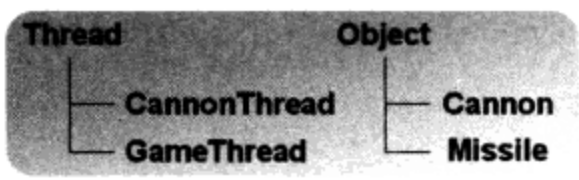
2. 后台逻辑模块

后台逻辑模块的类结构如图 14-12 所示，其主要包括两个继承自 Thread 的线程类和两个自己开发的 Object 的子类。下面将其中涉及的类简单说明如下。

- Cannon 类继承自 Object，其中主要封装了有关大炮的位置、射程等信息以及用于发射炮弹的方法。CannonThread 继承自线程类，其主要的功能是监视小球的位置，当小球进入其射程范围后，向小球发射炮弹。
- Missile 类为炮弹类，炮弹类对象在 CannonThread 中由 Cannon 对象负责创建，每个 Missile 对象中都封装了其方向、坐标等信息，以及负责移动炮弹并进行碰撞检测的成员方法。
- GameThread 主要负责两件工作，第一个是修改 GameView 中陨石和陷阱等动画的帧索引，第二个就是根据小球的方向移动小球，并进行相应的碰撞检测和处理。

3. 手机姿态计算模块

手机姿态计算模块的类结构如图 14-13 所示，该模块只包含继承自 SensorListener 的 BallListener 类和自己开发的 RotateUtil 类，简单介绍如下。



▲ 图 14-12　后台逻辑的模块的类结构

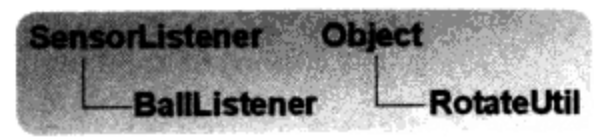


图 14-13　手机姿态计算模块

- BallListener 类继承自 SensorListener 类，其主要负责监听手机姿态的变化并将捕获到的数据传递给 RotateUtile 进行分析计算。
- RotateUtil 类主要负责接收 BallListener 捕获到的手机姿态的变化数据，并根据一系列算法

计算出在这种姿态下小球的运动方向。

14.4 主控制类——DriftBall 类的开发

DriftBall 类继承自 Activity，是本游戏中比较重要的一个类，DriftBall 类的功能在前面的小节中也已经提到过，本节就来具体地介绍 DriftBall 类的开发。

14.4.1 DriftBall 类的代码框架

为让读者对 DriftBall 有一个整体的把握，本小节首先来介绍 DriftBall 类的代码框架，在代码框架中，将主要介绍成员变量，成员方法会在以后遇到的时候详细介绍，DriftBall 类的代码框架如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 DriftBall.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.app.Activity;                              //引入相关类
3    ……此处省略部分引入相关类代码，读者可自行查阅随书光盘
4    import android.view.WindowManager;                        //引入相关类
5    public class DriftBall extends Activity {
6          ……//此处省略游戏常量的声明，将在随后补全
7          int level = 1;                                      //初始状态等级为 1
8          int life = MAX_LIFE;                                //初始状态生命数最大
9          Rect rectStart;                                     //开始圆球按钮的矩形框
10         ……//此处省略部分 Rect 对象的声明，读者可自行查阅随书光盘
11         Rect rectGameMsgBox;                                //屏幕中间提示消息的矩形框
12         MediaPlayer mpGameMusic;                            //游戏背景音乐
13         ……//此处省略部分 MediaPlayer 对象的声明，读者可自行查阅随书光盘
14         MediaPlayer mpBreakOut;                             //显示菜单和掉入陷阱以及被吃掉的音乐
15         boolean wantSound = true;                           //标志位，记录是否播放音乐
16         View currView;                                      //记录当前显示的 View
17         GameView gv;                                        //游戏视图的引用
18         WelcomeView wv;                                     //欢迎视图的引用
19         WelcomeView hv;                                     //帮助视图的引用
20         BallListener bl;                                    //继承自 SensorListener 的监听器
21         SensorManager mySensorManager;                      //SensorManager 对象
22         Handler myHandler = new Handler(){//自定义的 Handler 对象，用于处理收到的消息};
23         public void onCreate(Bundle savedInstanceState) {//重写 onCreate 方法}
24         public void startSensor(){//方法：启动传感器}
25         public void pauseSensor(){//方法：暂停传感器}
26         public void initSound(){//方法：初始化声音资源}
27         public void initRect(){ //方法：初始化矩形区域 Rect}
28         protected void onPause() {//重写 onPause 方法}
29         public boolean onTouchEvent(MotionEvent event) {//重写 onTouchEvent 方法，处理
                                                            //用户单击屏幕事件}
30         public boolean onKeyUp(int keyCode, KeyEvent event) {//重写的 okKeyUp 方法，处
                                                                //理键盘按下事件}
31   }
```

- 代码第 7 行声明了用于记录关卡的成员变量，代码第 8 行声明了用于记录小球生命数的成员变量，MAX_LIFE 是 DriftBall 类静态常量，有关 DriftBall 类的静态常量将在随后介绍。
- 代码第 9～11 行声明了一系列的 Rect 对象，这些对象将用于 onTouchEvent 方法中，

onTouchEvent 在处理用户单击事件时会将单击位置与不同的 Rect 对象进行匹配。

- 代码第 12～14 行声明了游戏中会用到的声音资源 MediaPlayer 对象，这些对象将在 initSound 方法中被初始化。代码第 16 行声明了用于记录当前 View 的成员变量，其值只能是代码第 17～19 行声明的 GameView、WelcomeView 或 WelcomeView。
- 代码第 20 行声明了继承自 SensorListener 的监听器对象，其将会被代码第 31 行的 SensorManager 对象注册或取消注册。
- 代码第 22 行创建了一个 Handler 对象，该对象用于接收消息并进行相应处理，本书将会在 14.5 节游戏主菜单的开发中详细介绍。
- 代码第 23～30 行为 DriftBall 类的所有方法，目前这些方法均为空实现，在后面的小节中会根据需要对这些方法进行介绍。

在上述代码的第 8 行省略了 DriftBall 类的静态常量的声明，这些省略的代码如下。

```
1    public static final int STATUS_PLAY = 0;          //游戏进行中，GameView 的状态
2    public static final int STATUS_PAUSE = 1;         //游戏暂停，GameView 的状态
3    public static final int STATUS_WIN = 2;           //通过一关，GameView 的状态
4    public static final int STATUS_LOSE = 3;          //丢掉一条命，GameView 的状态
5    public static final int STATUS_OVER = 4;          //命数耗尽，游戏结束，GameView 的状态
6    public static final int STATUS_PASS = 5;          //通过全部关卡，GameView 的状态
7    public static final int MAX_LIFE = 5;             //最大生命数
8    public static final int MAX_LEVEL = 5;            //最大关卡数
```

14.4.2　DriftBall 类的主要成员方法的实现

前面的小节介绍了 DriftBall 类的代码框架，下面对主要的方法做一个简单的介绍。有些方法要用到后面的内容，所以会放在合适的位置再介绍。本节要介绍的方法按如下步骤来进行说明。

（1）onCreate 方法，onCreate 方法是 Activity 创建时被调用的，在 onCreate 方法中主要进行各项初始化工作，同时为游戏设定一个当前屏幕，代码如下。

```
1        public void onCreate(Bundle savedInstanceState) {      //重写 onCreate 方法
2            super.onCreate(savedInstanceState);
3            requestWindowFeature(Window.FEATURE_NO_TITLE);      //设置全屏
4            getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
5                                 WindowManager.LayoutParams.FLAG_FULLSCREEN);
6            wv = new WelcomeView(this);                         //创建 WelcomeView 对象
7            setContentView(wv);                                 //设置当前显示的屏幕
8            currView = wv;                                      //记录当前 View
9            bl = new BallListener(this);                        //创建 BallListener
10           mySensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
11           initRect();                                         //初始化矩形区域
12           initSound();                                        //初始化声音
13       }
```

说明　在 onCreate 方法中，首先设置全屏显示，然后先将屏幕设置为 WelcomeView 对象，然后创建和传感器相关的监听器对象和 SensorManager 对象，最后调用初始化方法来完成程序的初始化工作。initRect 和 initSound 方法本身不涉及复杂的逻辑，只是简单地创建对象，故本书在此不再赘述，读者可以自行查阅随书光盘。

（2）startSensor 方法。该方法的功能是使用 SensorManager 对象注册监听器，其代码如下。

```
1        public void startSensor(){                                  //方法：启动传感器
2              mySensorManager.registerListener(bl,
3                                SensorManager.SENSOR_ORIENTATION,
4                                SensorManager.SENSOR_DELAY_GAME);
5        }
```

说明 在前面第 4 章曾经介绍过传感器应用的开发，所以上述代码应该不难理解。由于本游戏使用的传感器类型为姿态传感器，所以 registerListener 方法传入的第二个参数为 SENSOR_ORIENTATION，第一个参数为监听器 BallListener 对象。

（3）有了 startSensor 方法，相应地就该有 pauseSensor 方法，该方法代码如下。

```
1     public void pauseSensor(){                                  //方法：暂停传感器
2          mySensorManager.unregisterListener(bl);
3     }
```

说明 该方法调用 SensorManager 对象的 unregisterListener，取消对 BallListener 对象的监听。在游戏中，startSensor 和 pauseSensor 将会在游戏的多个地方被调用。

（4）onPause 方法。onPause 方法在 Activity 暂停时被调用，在本游戏中其执行的代码为暂停取消监听器，onPause 方法的代码如下。

```
1     protected void onPause() {                                  //重写 onPause 方法
2          pauseSensor();                                         //调用 pauseSensor 方法
3          super.onPause();
4     }
```

说明 在 onPause 方法中主要调用的是 pauseSensor 方法，该方法的功能是取消注册传感器监听器。

（5）onTouchEvent 方法的代码框架，onTouchEvent 主要负责处理用户单击屏幕事件。由于 onTouchEvent 方法的代码涉及了很多后面小节的内容，所以在此只将其代码框架列出。后面对该方法进行完善时再详细介绍。onTouchEvent 方法的代码框架如下。

```
1     public boolean onTouchEvent(MotionEvent event) { //重写的方法，用于接收和处理用户单击屏
                                                       //幕事件
2          if(event.getAction() == MotionEvent.ACTION_UP){ //判断事件类型
3               int x=(int)event.getX();
4               int y=(int)event.getY();
5               if(currView == wv){                          //当前为欢迎界面}
6               else if(currView == gv){                     //当前为游戏界面}
7          }
8          return true;
9     }
```

- 代码第 2 行判断单击事件的类型，即当用户单击屏幕后抬起才会执行 3～6 的代码段。代码第 3～4 行获取用户单击的位置在屏幕上的坐标。
- 代码第 5～6 行对当前 View 进行判断，如果满足条件，则将执行相应逻辑内的代码，在本程序中，代码第 5～6 行的详细部分将在需要的地方进行说明。

说明　本游戏为了调试方便，仍然会使用 SensorSimulator 来模拟手机姿态的变化，关于 SensorSimulator 的知识在第 4 章已有相关介绍。在本章将不再详述如何对程序源代码进行 3 处修改以满足 SensorSimulator 的测试条件，这 3 处修改已在程序的源代码中用注释标明，读者可自行查看。

14.5 游戏主菜单的开发

从本节开始，就来详细介绍本游戏中的各个功能模块是如何实现的。本节将要讲述的是游戏主菜单的开发，其中涉及的类有 WelcomeView、WelcomeThread、WelcomeDrawThread 以及继承自 Activity 类的 DriftBall 类。

14.5.1 欢迎界面类——WelcomeView 类的代码框架

WelcomeView 类主要负责绘制游戏中欢迎界面和主菜单，其继承自 SurfaceView 并实现了 SurfaceHolder.Callback 接口，代码框架如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 WelcomeView.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.content.Context;                           //引入相关类
3    ……//此处省略部分引入相关类代码，读者可自行查阅随书光盘
4    import android.view.SurfaceView;                          //引入相关类
5    //继承自 SurfaceView 的子类
6    public class WelcomeView extends SurfaceView implements SurfaceHolder.Callback{
7         DriftBall father;                                    //Activity 引用
8         WelcomeThread wt;                                    //WelcomeThread 引用
9         WelcomeDrawThread wdt;                               //WelcomeDrawThread 引用
10        Bitmap [] bmpBackScreen;                             //后台背景的动画帧
11        Bitmap [] bmpStartOrQuit;                            //开始和退出的两个球形按钮图片
12        Bitmap [] bmpSoundOption;                        //声音选项的两个球形按钮图片
13        int status = -1;          //状态值，为 0 表示待命,1 表示按钮按下，2 为显示加载中提示
14        int selectedIndex = -1;                          //被选中的命令，只能是开始或退出
15        int backIndex;                                   //后台帧动画的索引
16        Matrix m;                                        //Matrix 对象，负责旋转和缩放图片
17        int [][] planetCoordinate={                      //3 个球的位置数组
18                  {60,120},{120,300},{180,220}
19        };
20        public WelcomeView(DriftBall father) {//构造器，初始化成员变量}
21        public void initBitmap(Context context){//方法：初始化图片资源}
22        public void doDraw(Canvas canvas){//方法：屏幕绘制方法}
23        public void surfaceChanged(SurfaceHolder holder, int format, int width,int
          height) {//重写接口方法}
24        public void surfaceCreated(SurfaceHolder holder) {//重写 surfaceCreated 方法}
25        public void surfaceDestroyed(SurfaceHolder holder) {//重写 surfaceDestroyed 方法}
26   }
```

- 代码第 10～12 行为 WelcomeView 界面所用到的图片，其中 bmpBackScreen 数组中存放的是后台背景的动画帧，bmpStartOrQuit 数组中存放为开始和退出的球形按钮图片，bmpSoundOption 数组中存放声音开和声音关两个按钮图片。
- 代码第 13 行声明了用于记录状态的成员变量 status，Welcome 的绘制方法会根据 status 值

的不同绘制不同的内容，而 WelcomeThread 线程也会根据 status 的不同修改不同的数据。

- 代码第 14 行声明了用于记录被选择的命令的索引，其中 0 代表按下了开始按钮，1 代表按下了退出按钮。该成员变量同时也是代码第 13 行声明的 Bitmap 数组中的索引。
- 代码第 15 行声明的 backIndex 是屏幕背景动画帧数组 bmpBackScreen 的索引。第 16 行声明的 Matrix 对象用于对图片进行旋转和缩放。第 17 行声明的 int 型二维数组用于记录屏幕上 3 个球形按钮所在的位置。
- 代码第 20～25 行为 Welcome 类的构造器、成员方法和接口方法，目前，这些方法都为空实现，在后面的内容中会根据需要为其添加实现代码。

14.5.2 WelcomeView 类主要成员方法

前面小节介绍了 WelcomeView 类的代码框架，本小节将会对其中比较重要的方法进行详细说明，其步骤如下。

（1）doDraw 方法的开发，WelcomeView 中的 doDraw 方法实现的功能主要是根据程序不同的状态，绘制不同的内容到屏幕上，其代码如下。

```
1    public void doDraw(Canvas canvas){                          //方法：绘制屏幕
2         canvas.drawBitmap(bmpBackScreen[backIndex], 0, 0, null);//画背景
3         switch(status){                                        //根据状态绘制不同的内容
4         case 1:                                                //状态为：按钮按下状态
5              Bitmap tmpBmp = bmpStartOrQuit[selectedIndex];   //获取被选中的按钮图片
6              //获得缩小后的图片
7              bmpStartOrQuit[selectedIndex] = Bitmap.createBitmap(tmpBmp, 0, 0,
               tmpBmp. getWidth(),
8                              tmpBmp.getHeight(),m,true);
9              if(tmpBmp.getWidth() <= 5){                        //如果图片已经缩小到一定程度
10                  father.myHandler.sendEmptyMessage(selectedIndex);//向 Activity 发
                    Handler
11                  status = 2;                                   //设置状态为 2，即显示加载中提示
12             }
13        case 0:                                                //状态为：正常待命状态
14             canvas.drawBitmap(bmpStartOrQuit[0], planetCoordinate[0][0],
               planetCoordinate [0][1], null);
15             canvas.drawBitmap(bmpStartOrQuit[1], planetCoordinate[1][0],
               planetCoordinate [1][1], null);
16             int soundIndex = (father.wantSound?1:0);   //获得目前的声音开关状态
17             canvas.drawBitmap(bmpSoundOption[soundIndex], planetCoordinate[2][0],
18                             planetCoordinate[2][1], null);
19             break;
20        case 2:                                                //状态为：显示加载中
21             Paint p = new Paint();
22             p.setTextSize(28);                                //设置字体大小
23             p.setColor(Color.RED);                            //设置画笔颜色
24             p.setAntiAlias(true);                             //设置抗锯齿
25             p.setTextAlign(Paint.Align.CENTER);
26             canvas.drawText("加载中...", 160, 240, p);
27             break;
28        }}
```

- 代码第 2 行为绘制背景图片的代码，不管 WelcomeView 处于何种状态，背景图片帧的轮换是固定不变的，由 WelcomeThread 负责。
- 代码第 11～14 行为检测按照 Matrix 对象被缩小的图片是否已经小于某个阈值，如果小于

该值，则将 WelcomeView 的状态转换为 2，并向 DriftBall 发送 Handler 消息。

- 代码第 12 行向 DriftBall 发送 Handler 时是以被选中的按钮图片的索引来区别消息类型的，即将索引值作为 Message 对象的 what 字段值发送出去。
- 代码第 16～21 行为待命状态下（状态值为 0）执行的绘制代码，待命状态下的绘制内容主要是开始、退出和声音选项 3 个按钮。其中声音选项按钮需要对 DriftBall 类的 wantSound 标志位进行判断后得出。由于待命状态执行的这段代码在按钮按下状态也同样要执行，故在状态 1 时代码第 14 行之后没有 break 语句，而是直接执行到代码第 21 行。
- 代码第 22～29 为加载状态下（状态值为 2）执行的绘制代码，加载状态不需要绘制 3 个按钮，只需要在背景图片的基础上绘制提示文字即可。

（2）上述代码第 12 行用到了 DriftBall 类中成员 myHandler 的 sendEmptyMessage 方法，在 14.4 节中列举 DriftBall 的代码框架时对于 myHandler 对象的创建提供了空实现，下面就来详细介绍 myHandler 对象开发。其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 DriftBall.java。

```
1    Handler myHandler = new Handler(){
2         public void handleMessage(Message msg) {    //重写 handleMessage 方法
3              switch(msg.what){
4              case 0:                                //0 为收到来自 WelcomeView 的开始游戏命令
5                   gv = new GameView(DriftBall.this);
6                   setContentView(gv);               //设置当前 View
7                   currView = gv;                    //记录当前 View
8                   startSensor();                    //开启传感器
9                   wv = null;
10                  break;
11             case 1:                                //1 为收到来自 WelcomeView 的退出游戏命令
12                  System.exit(0);                   //退出程序
13                  break;
14             }}};
```

- 代码第 2～14 行重写了 handleMessage 方法，重写该方法是创建 Handler 工作中一个比较重要的内容，在该方法中，将根据收到的 Message 对象中的 what 等字段信息来进行不同的操作。
- 代码第 4～10 行为当收到 what 值为 0 的消息时执行的代码，主要是创建 GameView 并将屏幕切换到 GameView，最后启动传感器监听器。
- 代码第 11～13 行为当收到 what 值为 1 的消息时执行的代码，即调用 System 的 exit 方法退出。

说明　WelcomeView 中的 initBitmap 以及为实现 SurfaceHolder.Callback 接口而重写的 surfaceChanged、surfaceCreated 和 surfaceDestroyed 方法与前面章节中的同名方法实现的功能相同，代码也十分类似，本书由于篇幅有限就不予列出，读者可以自行查阅随书光盘。

14.5.3　WelcomeThread 类的开发

前面两个小节介绍了 WelcomeView 类的开发，本小节将会介绍 WelcomeThread 类的开发。由于本游戏的 WelcomeView 不十分复杂，所以，负责修改后台数据的 WelcomeThread 的代码也相对

简单一些，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 WelcomeThread. java。

```
1    package wyf.wpf;                                          //声明包语句
2    public class WelcomeThread extends Thread{
3         WelcomeView father;                                  //WelcomeView 引用
4         int sleepSpan = 100;                                 //休眠时间
5         boolean flag = false;
6         public WelcomeThread(WelcomeView father){            //构造器，初始化成员变量
7              this.father = father;
8              this.flag = true;
9         }
10        public void run(){                                   //方法：线程执行方法
11             while(flag){
12                  switch(father.status){
13                  case 2:                                     //加载状态
14                  case 0:                                     //待命状态
15                      father.backIndex = (father.backIndex+1)%father.bmpBackScreen.
                         length;
                                                                //修改帧索引
16                       break;
17                  case 1:                                     //按钮按下状态
18                       father.m.postScale(0.9f, 0.9f);        //对 Matrix 进行缩放
19                       father.m.postRotate(30);               //对 Matrix 进行旋转
20                       break;
21                  }
22                  try{Thread.sleep(sleepSpan);}               //睡眠一段时间
23                  catch(Exception e){e.printStackTrace();}    //捕获并打印异常
24             }}}
```

- 代码第 3～4 行为 WelcomeThread 类的成员变量，这些变量将在构造器中初始化。而代码第 6～9 行便为 WelcomeThread 的构造器。
- 代码第 10～27 行为线程的 run 方法的代码。在 run 方法中，每次循环都要判断 WelcomeView 的 status 的值，如果状态为 0（待命状态）或 2（加载状态），就不断地更换背景图片的帧索引实现动画。如果状态为 1（按钮按下），就对 Matrix 对象进行缩放和旋转，以便在绘制时创建缩小的图片。

说明　WelcomeDrawThread 的实现在前面的章节中已经介绍过，在此不再赘述，读者可以自行查阅随书光盘。

14.5.4　菜单界面的用户交互事件处理

完成了前面几个小节的开发，主菜单界面的绘制工作算是基本完成了，但是还没有添加对处理用户单击屏幕事件的代码。本小节就来进行主菜单界面的最后一步工作，添加用户交互事件代码。

与用户的交互事件的处理是通过 DriftBall 类中的 onTouchEvent 方法来实现的，在 14.4.2 小节介绍 DriftBall 类的主要成员方法时，对于 onTouchEvent 只提供了一个框架，本小节就来完善其中的一部分功能。打开项目文件 DriftBall.java，在 onTouchEvent 方法中添加如下代码。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 DriftBall.java。

```
1    ……//此处省略不相关的代码，读者可自行查阅随书光盘
```

```
2    if(currView == wv){                                    //当前为欢迎界面
3        if(rectStart.contains(x, y)){                      //点击开始的圆球
4            wv.status = 1;                                 //设置状态为 1，有按钮按下
5            wv.selectedIndex = 0;                          //设置被选中的按钮为开始按钮
6        }
7        else if(rectSoundOption.contains(x, y)){           //点击切换声音的圆球
8            wantSound = !wantSound;                        //切换标志位
9            if(!wantSound){                                //如果当前在播放声音，停止
10               if(mpGameMusic.isPlaying()){
11                   try {
12                       mpGameMusic.pause();               //暂停游戏背景音乐
13                   } catch (Exception e) {}
14               }}
15           else if(wantSound){                            //如果当前没有播放声音，播放
16               if(!mpGameMusic.isPlaying()){
17                   try {
18                       mpGameMusic.start();               //播放游戏背景音乐
19                   } catch (Exception e) {}
20               }}}
21       else if(rectQuit.contains(x, y)){                  //点击退出的圆球
22           wv.status = 1;                                 //设置状态为 1，有按钮按下
23           wv.selectedIndex = 1;                          //设置被选中的按钮为退出按钮
24       }}
25   ……//此处省略不相关的代码，读者可自行查阅随书光盘
```

- 代码第 3～6 行为判断并处理用户是否单击了开始按钮的代码，如果点击了开始按钮，则将 WelcomeView 的状态设置为 1（按钮按下状态），并设置 selectedIndex 为 0，用于绘制和发送 Handler 消息。
- 代码第 7～20 行为判断并处理用户是否单击了声音开关按钮的代码，如果用户点击了那个按钮，就将声音开关状态的标志位置反。同时根据用户新设置的声音状态标志位来停止已播放的声音或播放未播放的声音。
- 代码第 21～23 行为判断并处理用户是否单击了退出按钮的代码，如果用户点击了那个按钮，就将状态位设置为 1（有按钮按下），同时设置 selectedIndex 为 1，用于绘制和发送消息。

到 14.5 节为止，游戏的主菜单界面的开发已经基本完成，在模拟器中运行本游戏，其效果如图 14-1 所示。

14.6 游戏界面模块的开发

本书在 14.5 节中介绍了游戏主菜单的开发，在游戏的主菜单中单击开始按钮后，会向 DriftBall 发 Handler 消息通知其创建 GameView，并将屏幕切换到 GameView。在游戏界面中涉及的类有很多，本节将对 GameView、GameThread 和 GameMenu 类进行简单地介绍。

14.6.1 游戏界面类——GameView 的成员变量

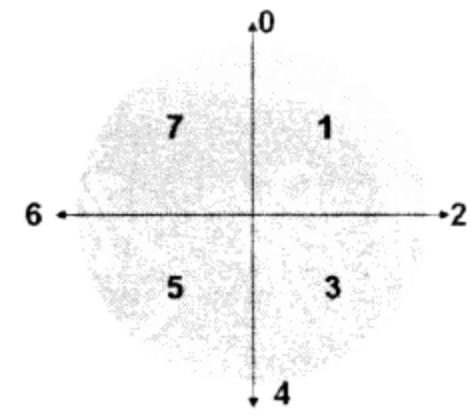
GameView 是游戏画面的核心类，其主要负责游戏的前台显示以及游戏过程控制，本小节将对 GameView 中主要的成员变量做一个简单地介绍。其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameView.java。

```
1    package wyf.wpf;                                       //声明包语句
```

```
2    import java.util.ArrayList;                                    //引入相关类
3    import android.content.res.Resources;                          //引入相关类
4    ……//此处省略部分引入相关类语句
5    import android.view.SurfaceHolder.Callback;                    //引入相关类
6    import static wyf.wpf.DriftBall.*;              //引入 DriftBall 类的静态成员
7    public class GameView extends SurfaceView implement  ts Callback{
8         int screenWidth = 320;                                   //屏幕宽度
9         int screenHeight = 480;                                  //屏幕高度
10        int backY;                                               //背景图片的 Y 坐标
11        int nebulaX;                                             //星云的 x 坐标
12        int nebulaY;                                             //星云的 y 坐标
13        int ballX;                                               //小球的横坐标
14        int ballY;                                               //小球的纵坐标
15        int tileSize = 20;                                       //图元的大小
16        int direction = -1;                             //小球运动方向，上为 0，顺时针依次为 1～7
17        int velocity = 4;                                        //小球运动速度
18        int eatIndex;                                            //吃人动画帧索引
19        int trapIndex;                                           //陷阱动画索引
20        int status;                                              //游戏状态
21        boolean showMenu;                                        //是否显示菜单
22        DriftBall father;                                        //DriftBall 引用
23        DrawThread dt;                                           //绘制线程
24        GameThread gt;                                           //后台数据的修改线程
25        CannonThread ct;                                         //大炮线程
26        GameMenuThread gmt;                                      //游戏菜单线程
27        Meteorolite [] meteoArray;                               //陨石数组
28        ArrayList<Missile> alMissile = new ArrayList<Missile>();//存放导弹集合
29        ArrayList<Cannon> alCannon = new ArrayList<Cannon>();   //存放大炮集合
30        Bitmap bmpNebula;                                        //星云图片
31        Bitmap bmpBall;                                          //小球图片
32        ……//此处省略部分 Bitmap 对象的声明，读者可自行查阅随书光盘
33        Bitmap bmpGameOver;                                      //游戏结束图片
34        Bitmap bmpGamePass;                                      //游戏通关图片
35        byte [][] currMap;    //地图，为 0 则空，为 1 则路,2 为家,6 是吃小球的,7 是陷阱,8 是大炮
36        int [][] menuCoordinate;                                 //菜单及菜单项的位置
37        ……//以下省略 GameView 的成员方法的定义
38   }
```

- 代码第 6 行为引入 DriftBall 类静态成员的语句，这样在 GameView 类的代码中，如果需要引用 DriftBall 类的静态成员变量或方法，可以直接引用而不需要在前面冠以“DriftBall”。
- 代码第 10 行为背景图片的 y 坐标，其 x 坐标一直为 0。
- 代码第 15 行声明了图元的大小，本游戏中的地图由各种图元搭配而成，每个图元的长宽都为 20 个像素值。
- 代码第 16 行声明了用于记录小球方向的成员变量，本游戏的方向系统比较简单，上、下、左、右 4 个方向用 0、2、4、6 来表示，其余部分则用 1、3、5、7 来表示，如图 14-14 所示。



▲ 图 14-14　游戏的方向系统

- 代码第 17 行为小球的速度，即每次移动小球的步进。
- 代码第 20 行声明了用于记录游戏状态的成员变量，其值为 DriftBall 类中静态常量中的一个。
- 代码第 28～29 行声明了用于存放炮弹对象和大炮对象的集合，每一关游戏开始前，GameView 都会检索地图，如果其中有大炮，就将其添加到 alCannon 中。
- 代码第 30～34 行声明了游戏中会用到的图片对象，其中省略了部分对象的声明代码，读者

可自行查阅随书光盘。

- 代码第 35 行声明了用于存放地图数据的二维数组，每一关游戏开始前，都会调用 GameMap 类的静态方法为其赋值。代码第 36 行声明了存放菜单及菜单项坐标的二维数组。

14.6.2　GameView 的成员方法简介

前面的小节介绍了 GameView 类的主要成员变量，本小节将对其成员方法进行简单介绍，有一些成员方法将会在其他小节进行详细说明。

```
1    public GameView(DriftBall father) {//构造器}
2    public void initBitmap(DriftBall father){//方法：初始化图片资源}
3    public void initGame(){//方法：初始化游戏}
4    public void resumeGame(){//方法：恢复游戏}
5    public void doDraw(Canvas canvas){//方法：绘制屏幕}
6    public void drawNumber(Canvas canvas,int life) {//方法：绘制数字}
7    public void drawMap(Canvas canvas){//方法：绘制地图}
8    public boolean checkForCannon(){//方法：检查是否有大炮}
9    public void pauseGame(){//方法：暂停游戏}
10   public void shutAll(){//方法：关闭所有后台线程}
11   public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {//实现接口方法}
12   public void surfaceCreated(SurfaceHolder holder) {//重写 surfaceCreated 方法}
13   public void surfaceDestroyed(SurfaceHolder holder) {//重写 surfaceDestroyed 方法}
```

- 代码第 1 行为 GameView 的构造器，在构造器中主要完成了成员变量的初始化和添加 SurfaceHolder.Callback 接口实现等工作。
- 代码第 3 和 4 以及 9 和 10 行为对游戏过程进行控制的方法，对于控制方法将会在需要的时候进行详细说明，在此只提供空实现。代码第 3 行的 initGame 方法中主要进行关卡地图的加载以及小球、星云等元素位置初始化等工作。
- 代码第 9 行的 pauseGame 方法进行的工作是暂停 DrawThread、GameThread 和 CannonThread 的执行。代码第 4 行的 resumeGame 方法用于恢复暂停后的游戏，即恢复上述线程的执行。代码第 10 行的 shutAll 方法则是停止上述 3 个线程的执行。
- 代码第 5 行为 GameView 的主绘制方法 doDraw，在该方法中调用了第 6 和 7 行的 drawNumber 和 drawMap 方法。
- 代码第 11～13 行定义的 surfaceChanged、surfaceCreated 和 surfaceDestroyed 方法为实现的 SurfaceHolder.Callback 接口方法，这 3 个方法读者应该非常熟悉了，故在此不再详述。

14.6.3　游戏界面绘制线程——GameThread 类的代码框架

GameThread 类的主要功能是修改 GameView 所要绘制的内容。同时 GameThread 还有一个功能是，负责移动小球并进行碰撞检测，这部分内容将在后面小节中详细介绍。本小节将对其代码框架进行简单说明，其代码框架如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameThread.java。

```
package wyf.wpf;                                          //声明包语句
import static wyf.wpf.DriftBall.*;                        //引入 DriftBall 类的静态常量
public class GameThread extends Thread{
      GameView father;                                    //GameView 对象引用
```

```
5          int sleepSpan = 25;                                  //休眠时间
6          int backCounter;                                     //背景帧控制计数器
7          int nebulaCounter;                                   //星云帧控制计数器
8          int eatCounter;                                      //吃掉小球的帧控制计数器
9          int trapCounter;                                     //陷阱帧控制计数器
10         int meteoCounter;                                    //陨石帧控制计数器
11         boolean flag = false;                                //外层循环，线程是否执行
12         boolean isGameOn = false;                            //游戏是否在进行
13         //构造器，初始化主要成员变量
14         public GameThread(GameView father){                  //构造器
15              this.father = father;
16              this.flag = true;
17         }
18         public void run(){//方法：线程执行方法}
19         public void checkAndMoveBall() {//方法：检查移动小球}
20         public boolean checkCollision() {//方法：进行碰撞检测和处理碰撞}
21    }
```

- 代码第 6～10 行分别声明了用于控制背景图片、星云、吃人漩涡、陷阱、陨石动画帧变换的计数器。通过为不同的绘制元素提供单独的计数器，可以在同一个线程中实现不同的帧变换频率、不同的移动速度等效果。
- 代码第 11 和 12 行分别声明了用于表示线程是否执行和游戏是否在进行的标志位，这两个成员变量将在代码第 14～17 行的构造器中进行初始化。
- 代码第 18 行为线程的 run 方法，GameThread 的功能也是通过执行 run 方法来实现的，在随后的小节中将会对 run 方法进行详细地介绍。
- 代码第 19 和 20 行为 checkAndMoveBall 和 checkCollision 方法，这两个方法用于移动小球并进行碰撞检测和处理，此内容将会在后台逻辑模块中进行详细地介绍。

14.6.4 GameMenuThread 类的开发

GameMenuThread 类主要负责在玩家按下手机上“MENU”键和返回键时对菜单背景及 4 个菜单项进行逐行滑入和逐行滑出动画效果的控制，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameMenuThread. java。

```
1     package wyf.wpf;                                          //声明包语句
2     public class GameMenuThread extends Thread{
3          DriftBall father;                                    //DriftBall 类引用
4          boolean flag;                                        //循环控制变量
5          boolean isIn;                                        //是否是滑入菜单
6          boolean isOut;                                       //是否是滑出菜单
7          int sleepSpan = 20;                                  //睡眠时间
8          int [][] menuCoordinate={                            //菜单背景和菜单项的初始位置
9                    {350,0},{400,30},{450,80},{500,130},{550,180}
10         };
11         public GameMenuThread(DriftBall father){             //构造器，初始化成员变量
12              this.father = father;
13              father.gv.menuCoordinate = this.menuCoordinate;
14              flag = true;                                    //将 flag 设置为 true
15              isIn = true;                                    //将 isIn 设置为 true
16         }
17         public void run(){                                   //方法：线程执行方法
18              int inIndex=0;                                  //滑入的索引顺序
19              int outIndex = 4;                               //滑出时的索引顺序
```

```
20            while(flag){
21                if(isIn){                                                    //菜单滑入屏幕
22                    for(int i=inIndex;i<father.gv.menuCoordinate.length;i++){
23                        father.gv.menuCoordinate[i][0]-= 25;//向屏幕内移动菜单项
24                        if(father.gv.menuCoordinate[i][0] == 150){//判断菜单项是否已
                                                                          //到位
25                            inIndex=i+1; //将开始索引下移，即不再改变已到位的菜单项坐标
26                        }}
27                    if(inIndex == 5){                          //如果所有菜单项都移动到位
28                        isIn = false;                          //设置 isIn 标志位为 false
29                    }}
30                else if(isOut){                                //菜单移出屏幕
31                    father.gv.menuCoordinate[outIndex][0] +=10;//向屏幕外移动菜单项
32                    if(father.gv.menuCoordinate[outIndex][0] >=320){//判断是否将菜单项
                                                                          //移出屏幕
33                        outIndex--;                            //改移动前一个菜单项
34                        if(outIndex < 0){                      //判断是否所有菜单项都移出屏幕
35                            if(father.currView == father.gv){
                                                                 //如果当前View是GameView
36                                father.gv.resumeGame();
                                                          //调用 resumeGame 方法恢复游戏
37                            }
38                            flag = false;                      //停止线程的执行
39                        }}}
40                try{Thread.sleep(sleepSpan);}//线程休眠
41                catch(Exception e){e.printStackTrace();}
42            }}}
```

- 代码第 8～10 行定义了用于记录菜单背景及 4 个菜单项的初始坐标的数组，在初始坐标中根据菜单项滑入的先后顺序对其坐标进行了相应设置，如先滑入的坐标比后滑入的要小。
- 代码第 11～16 行为 GameMenuThread 类的构造器，在构造器中将 isIn 设置为 true。
- 代码第 17～42 行为线程的 run 方法，在该方法中，首先判断是滑入菜单还是滑出菜单，如果是滑入菜单，则将菜单背景及菜单项的坐标递减并判断是否已到位，如果已到位就不再修改该菜单的坐标。如果是滑出菜单，则需要让菜单及菜单项逐个移出屏幕，只有前一个移出了屏幕，后一个才开始移出。

14.6.5　用户交互事件处理

在游戏界面中主要的用户交互事件是玩家打开菜单并进行相应的选择，以及对弹出的各种游戏提示进行反应。菜单的打开是通过在 DriftBall 类中重写 onKeyUp 方法来实现的，由于篇幅有限，该方法的代码将不予列出，读者可查阅随书光盘。本小节将主要对 onTouchEvent 方法中涉及游戏界面的用户处理事件进行简单介绍，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 DriftBall.java。

```
1     ……//此处省略不相关的代码，读者可自行查阅随书光盘
2     else if(currView == gv){                                  //当前为游戏界面在显示
3         if(gv.status == STATUS_PAUSE){                        //如果游戏状态为 STATUS_PAUSE
4             if(rectContinue.contains(x,y)){                   //点击继续游戏菜单
5                 gv.gmt.isOut = true;                          //菜单开始退出屏幕
6             }
7             else if(rectSoundAlter.contains(x, y)){           //点击切换声音菜单
```

```
8                   wantSound = !wantSound;
9                   if(!wantSound){                          //判断记录值的真假
10                      if(mpGameMusic.isPlaying()){
11                          try {mpGameMusic.pause();} //暂停声音
12                          catch (Exception e) {}
13                      }}
14                  else if(wantSound){
15                      if(!mpGameMusic.isPlaying()){
16                          try {mpGameMusic.start();}  //播放声音
17                          catch (Exception e) {}
18                      }}
19                  gv.gmt.isOut = true;
20              }
21              else if(rectHelp.contains(x, y)){            //点击帮助菜单
22                  gv.gmt.isOut = true;                     //滑出菜单
23                  gv.pauseGame();                          //暂停游戏
24                  hv = new HelpView(this);                 //创建 HelpView 对象
25                  setContentView(hv);                      //切换屏幕
26                  currView = hv;
27              }
28              else if(rectBackToMain.contains(x, y)){      //点击返回主菜单
29                  gv.gmt.isOut = true;                     //滑出菜单
30                  wv = new WelcomeView(this);              //创建欢迎界面
31                  setContentView(wv);                      //切换屏幕
32                  currView = wv;                           //记录当前 View
33                  pauseSensor();                           //暂停 Sensor
34                  gv.shutAll();                            //关闭 gv 的附属线程
35                  gv = null;
36              }}
37          else if(gv.status == STATUS_WIN){                //闯过一关
38              if(rectGameMsgBox.contains(x, y)){
39                  gv.initGame();
40                  gv.resumeGame();                         //开始下一关的游戏
41              }}
42          else if(gv.status == STATUS_PASS || gv.status == STATUS_OVER){//当前状态为通过全关或生
                                                                          //命耗尽
43              if(rectGameMsgBox.contains(x, y)){           //判断单击的位置
44                  wv = new WelcomeView(this);              //创建欢迎界面
45                  setContentView(wv);                      //切换屏幕
46                  currView = wv;                           //记录 View
47                  pauseSensor();                           //暂停监听器
48                  gv.shutAll();                            //关闭 gv 附属线程
49                  gv = null;
50              }}}
51          ……//此处省略不相关的代码，读者可自行查阅随书光盘
```

- 代码第 3～36 行为游戏状态为 STATUS_PAUSE 时的处理代码，游戏进入该状态时代表用户单击了菜单按钮。在该状态下主要进行的工作是判断单击的是哪个菜单项并进行相应的处理。
- 代码第 37～41 行为游戏状态为 STATUS_WIN 时的处理代码，游戏进入该状态时代表已经闯过了一关，执行的操作是初始化并开始下一关的游戏。
- 代码第 42～50 行为游戏状态为 STATUS_PASS 或 STATUS_OVER 时的处理代码，进入该状态时表示游戏已经失败或者已经胜利，所以进行的操作都是返回欢迎界面。
- 代码第 24 行创建了一个 HelpView 对象，HelpView 继承自 SurfaceView，其主要的功能是显示帮助的内容，实现起来比较简单，读者可自行查阅随书光盘。

14.7 游戏中各个图层的开发

在介绍完 GameView 类的代码框架之后，本节将介绍游戏中各个图层的开发。游戏画面中总共有 4 个图层，分别是游戏地图图层、背景星空图层、陨石图层和星云图层。

14.7.1　地图图层的开发

地图图层位于游戏图层中的最上层（有时会被陨石图层遮挡），地图图层中主要涉及的类是 GameMap，GameMap 类中主要包括所有关卡的地图信息，同时还提供了获取地图信息的成员方法，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameMap.java。

```
1     package wyf.wpf;                                    //声明包语句
2     public class GameMap{
3          static byte [][][] map = {//地图，为 0 则空，为 1 则路,2 为家,3 是加命,4 是吃小球,5 是
                                     //陷阱,6 是大炮
4                    {     //第一关地图数据
5                          {1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
6                          {0,1,1,1,1,5,1,1,1,1,0,0,0,0,0,0},
7                          {0,0,0,1,0,0,0,0,0,1,1,1,1,0,0,0},
8                          ……//此处省略部分地图数据，读者可自行查阅随书光盘
9                          {0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0},
10                         {0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0},
11                         {0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0}
12                   },
13                   {     ……//此处省略第二关地图数据，请查阅随书光盘     },
14                   {     ……//此处省略第三关地图数据，请查阅随书光盘     },
15                   {     ……//此处省略第四关地图数据，请查阅随书光盘     },
16                   {     ……//此处省略第五关地图数据，请查阅随书光盘     }
17         };
18         //方法，返回指定关卡的地图数组
19         public static byte[][] getMap(int level){
20              return map[level-1];
21         }}
```

- 代码第 3 行声明并创建了一个静态的 byte 型三维数组，其中包含 5 个二维数组，每个二维数代表一个关卡的地图数据。
- 代码第 4～12 行为第一关的地图数据，本游戏中图元大小为 20，没有滚动屏幕，故代表地图的二维数组为 24 行、15 列。其中每个图元编号代表不同的内容，见代码第 3 行的注释。
- 代码第 18～21 行为静态方法 getMap，该方法接收一个代表关卡的参数，返回指定的二维数组。

说明，GameMap 类中的 5 个关卡的地图矩阵是通过地图编辑器来设计的，有关地图编辑器的知识在前面的章节中已有介绍，读者可以参考前面的章节。

地图的绘制是通过 GameView 的成员方法 drawMap 来实现的，该方法将会在 doDraw 方法中被调用。在 14.6.2 小节中对于 drawMap 方法只提供了空实现，在此将对其代码进行介绍，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameView.java。

```
1    public void drawMap(Canvas canvas){                    //方法：绘制地图矩阵
2         for(int i=0;i<currMap.length;i++){
3              for(int j=0;j<currMap[i].length;j++){
4                   switch(currMap[i][j]){
5                   case 1:                                 //该位置为可通过的路
6                        canvas.drawBitmap(bmpTile, j*tileSize, i*tileSize, null);
7                        break;
8                   case 2:                                 //该位置为家
9                        canvas.drawBitmap(bmpHome, j*tileSize, i*tileSize, null);
10                       break;
11                  case 3:                                 //该位置为加命奖励
12                       canvas.drawBitmap(bmpPlusLife, j*tileSize, i*tileSize, null);
13                       break;
14                  case 4:                                 //该位置为吃人的漩涡
15                       canvas.drawBitmap(bmpEat[eatIndex], j*tileSize, i*tileSize, null);
16                       break;
17                  case 5:                                 //该位置为陷阱
18                      canvas.drawBitmap(bmpTrap[trapIndex], j*tileSize, i*tileSize,
null);
19                       break;
20                  case 6:                                 //该位置为大炮
21                       canvas.drawBitmap(bmpCannon, j*tileSize, i*tileSize, null);
22                       break;
23                  }}}}
```

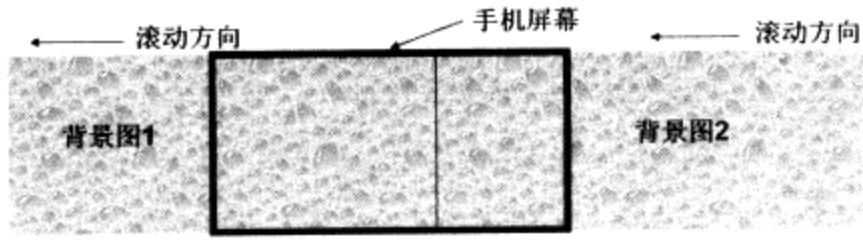
- 上述代码对二维数组 currMap 进行遍历，对于每个位置上的值进行判断，并根据该值绘制相应的地图元素。
- 代码第 15 和 18 行分别为对地图元素为吃人漩涡和陷阱的绘制，由于这两个元素在游戏中是以动画的方式来呈现的，所以每次绘制都根据各自的索引值获取图片帧，这些索引值将在 GameThread 中被修改。

14.7.2 其他图层的开发与实现

除了地图图层，其他 3 个图层的实现较为容易，本小节将会对这 3 个图层的开发进行介绍，其步骤如下。

（1）背景星空图层和星云图层的设计。背景星空图层位于游戏中图层的最底层，呈现的效果是星空不断由下往上飘动，其实现方式是通过绘制两张（或多张）可首尾相接的图片来达到屏幕滚动的效果。

本游戏中背景图层的滚动效果如图 14-15 所示。背景图片的高度为 600，宽度为 320。由于图片的高度大于手机屏幕的高度，所以初始时用一张图片来完成滚动效果。当第一张图片的下边沿要滚入屏幕时，就需要紧随第一张图片绘制第二张图片。当第一张图片完全移除屏幕时，只绘制第二张图片，如此往复，就实现了屏幕的缓缓滚动。



▲图 14-15　滚动星空背景的绘制示意图

星云图层处于地图图层和背景图层之间，其是图层中最简单的一种，星云与背景星空运动方向相反，其实现方式也比较简单。

（2）陨石图层的设计。陨石图层用于显示游戏进行中不断闪过的陨石，本游戏中的陨石图层在整个图层中的顺序是不定的，陨石或者从地图图层上方飞过，或者从其下方飞过。每个陨石都是一个 Meteorolite 对象，该类的代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 Meteorolite.java。

```
1    package wyf.wpf;                                          //声明包语句
2    public class Meteorolite{
3         int x;                                               //陨石 x 坐标
4         int y;                                               //陨石 y 坐标
5         boolean up;                          //为 true 则在地图图层上，为 false 则在地图图层下
6         int index;                           //陨石动画的帧索引
7         public Meteorolite(){                //构造器，调用 init 方法进行初始化
8              init();
9         }
10        public void init(){                                  //方法：初始化成员变量
11             x = (int)(Math.random()*290);                   //随机产生 X 坐标
12             y = -(int)(Math.random()*250);                  //随机产生 Y 坐标
13             up = (Math.random()>0.5?true:false);            //随机设置是否在上层的标志位
14             index = (int)(Math.random()*8);                 //随机产生陨石动画的帧索引
15        }}
```

说明　Meteorolite 类中主要封装了陨石的相关信息，其中代码第 10～15 行定义的 init 方法不仅用于在构造器中进行初始化工作，在陨石飞出屏幕边界后，也会调用该方法重新初始化小球的位置和在上、在下标志位，这样实现了对象的重用。

（3）各图层的实现代码。前面介绍了各个图层的设计，下面介绍具体的实现方法，首先绘制各个图层的工作是放在 GameView 类的 doDraw 方法中进行的。在 14.6.2 小节中对于 doDraw 方法只提供了空实现，下面来对其进行完善，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameView.java。

```
1    public void doDraw(Canvas canvas){                        //屏幕绘制方法
2         canvas.drawBitmap(bmpStar,0,backY,null);             //绘制星空背景
3         if(backY <= -120){                                   //检查是否需要绘制第二张星空背景
4              canvas.drawBitmap(bmpStar, 0, backY+600, null);
5         }
6         canvas.drawBitmap(bmpNebula, nebulaX, nebulaY, null);//绘制星云
7         for(Meteorolite m:meteoArray){                       //画下面的陨石
8              if(!m.up){                                      //检查陨石是否在下面
9                   canvas.drawBitmap(bmpMeteo[m.index], m.x,m.y,null);
10             }}
11        drawMap(canvas);                                     //绘制地图
12        if(status != STATUS_OVER && status != STATUS_PASS){  //检查游戏的状态
13             canvas.drawBitmap(bmpBall, ballX, ballY, null); //绘制小球
14        }
15        for(Meteorolite m:meteoArray){                       //画上面的陨石
16             if(m.up){                                       //检查陨石是否在上面
17                  canvas.drawBitmap(bmpMeteo[m.index], m.x,m.y,null);
18             }}
19        //画还有几条命
20        canvas.drawBitmap(bmpBall, 240, 0, null);
21        canvas.drawBitmap(bmpMultiply, 265, 2, null);
```

```
22          drawNumber(canvas,father.life);
23          try{
24               if(alMissile.size() != 0){                    //检查炮弹集合是否为空
25                    for(Missile m:alMissile){                //遍历炮弹集合
26                         canvas.drawBitmap(bmpMissile, m.x-5, m.y, null);  //绘制炮弹
27                    }}}
28          catch(Exception e){}
29          ……//此处省略绘制游戏提示(如成功过关等)的代码
30     }
```

- 代码第 2 行绘制了星空的背景，在第 3 行检查该背景的纵坐标是否小于 120，即该背景的下边沿是否已经进入屏幕，如果是，就需要绘制第 2 张背景图片了。
- 代码第 6 行为绘制星云图片的代码，代码第 7～10 行为绘制地图图层下方陨石，首先找出陨石列表中 up 属性为 true 的 Meteorolite 对象，然后将其绘制到屏幕上。代码第 15～18 行为绘制地图图层上方的陨石。
- 代码第 11 行为绘制地图图层，通过调用 drawMap 方法来实现。
- 代码第 23～28 行为绘制炮弹，首先检查炮弹列表是否为空，如果不为空则遍历该列表，对每个 Missile 对象进行绘制。
- 代码第 29 行省略了绘制游戏提示的部分，将在后面的小节中进行介绍。

各个图层除了在 GameView 的 doDraw 方法中被绘制，还需要在 GameThread 的 run 方法中被修改以实现动画效果。在 14.6.3 节介绍 GameThread 时，对其 run 方法提供了空实现，现在来对该方法进行初步的完善，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameView.java。

```
1     public void run(){                                        //线程执行方法
2          while(flag){
3               while(isGameOn){
4                    backCounter++;                            //星空的运动
5                    if(backCounter == 50/sleepSpan){          //星空是以 500ms 移动一次
6                         father.backY -=1;                    //每次移动向上 1 个像素
7                         backCounter = 0;
8                         if(father.backY == -600){            //如果该背景图移除屏幕,就复位到 0
9                              father.backY = 0;
10                        }}
11                   nebulaCounter++;                          //星云的运动
12                   if(nebulaCounter == 40/sleepSpan){        //星云是以 40ms 移动一次
13                        father.nebulaY +=1;                  //星云每次向下移动一个像素
14                        nebulaCounter = 0;                   //清空计数器
15                        if(father.nebulaY >= 480){           //如果星云移除屏幕
16                             father.nebulaY = -150;          //设置星云的 Y 坐标
17                             father.nebulaX = (int)(Math.random()*250);
                                                               //随机生成星云的 X 坐标
18                        }}
19                   meteoCounter++;                           //陨石的运动
20                   for(Meteorolite m:father.meteoArray){//遍历陨石列表
21                        m.y +=2;                             //移动陨石的位置
22                        if(m.y >= father.screenHeight){ //超出屏幕边界则重新初始化
23                             m.init();
24                        }}
25                   if(meteoCounter == 150/sleepSpan){//陨石动画的换帧，每 150ms 换一次帧
26                        meteoCounter = 0;                    //清空计数器
27                        for(Meteorolite m:father.meteoArray){     //遍历陨石列表
```

```
28                          m.index = (m.index + 1)%8;      //修改帧索引
29                      }}
30                  eatCounter++;                           //吃人漩涡动画
31                  if(eatCounter == 200/sleepSpan){        //每 200ms 换一帧
32                      father.eatIndex = (father.eatIndex+1)%father.bmpEat.length;
33                      eatCounter = 0;
34                  }
35                  trapCounter++;                          //陷阱动画
36                  if(trapCounter == 300/sleepSpan){       //每 300ms 换一帧
37                      father.trapIndex = (father.trapIndex+1)%father.bmpTrap.length;
38                      trapCounter = 0;
39                  }
40                  try{                                    //炮弹的运动
41                      if(father.alMissile.size() != 0){   //遍历，如果炮弹列表不为空
42                          for(Missile m:father.alMissile){    //遍历炮弹列表
43                              m.moveAndCheck();           //移动炮弹并进行碰撞检测
44                          }}}
45                  catch(Exception e){}
46                  checkAndMoveBall();                     //移动小球
47                  try{Thread.sleep(sleepSpan);}           //休眠一段时间
48                  catch(Exception e){e.printStackTrace();}
49              }
50              try{Thread.sleep(800);}                     //休眠一段时间
51              catch(Exception e){e.printStackTrace();}
52          }}
```

- 代码第 4～18 行为对星空背景和星云的移动控制，通过设置计数器，可以让星空和星云具有不同的移动时间间隔。每次移动后都要对其位置坐标是否超出屏幕进行判断。
- 代码第 19～24 行为对陨石位置的移动控制，每次移动陨石后检查其位置，如果超出屏幕，则调用 Meteorolite 的 init 方法重新初始化陨石的位置。
- 代码第 25～39 行分别为对陨石、吃人漩涡、陷阱的动画变换进行控制，通过设置计数器，可以让不同点绘制元素在同一个线程中享受到不同的动画换帧速率。
- 代码第 41～44 行和第 46 行为对游戏中的炮弹位置和小球位置进行移动的代码，每次移动之后都要对炮弹和小球进行碰撞检测。Missile 类和 GameThread 类的 checkAndMoveBall 方法将在后面讲解游戏后台逻辑时进行介绍。

到此为止，本游戏的前台开发已经基本完成，在模拟器中运行程序，其效果如图 14-3 所示。

14.8 游戏后台逻辑的开发

前面的几节对游戏的表示层进行了介绍，本小节介绍游戏的后台逻辑的实现。本游戏中后台逻辑模块主要是小球和炮弹的运动及碰撞检测模块，同时还包含对游戏过程进行控制的模块。

14.8.1　小球的运动控制

小球的运动控制是通过 GameThread 类的 checkAndMoveBall 方法实现的，该方法在 14.6.3 小节介绍 GameThread 类时只是提供了空实现，本小节将对其进行具体的介绍，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameThread.java。

```
1    public void checkAndMoveBall() {                          //方法：检查和移动小球
2        switch(father.direction){
3        case 0:                                               //方向向上
4            father.ballY -= father.velocity;                  //Y 方向上的移动
5            if(checkCollision()){                             //检测是否发生碰撞
6                father.ballY += father.velocity;              //若发生碰撞就撤销移动
7            }
8            break;
9        case 1:                                               //方向右上
10           father.ballY -= father.velocity;                  //Y 方向上的移动
11           if(checkCollision()){                             //检测是否发生碰撞
12               father.ballY += father.velocity;              //若发生碰撞就撤销移动
13           }
14           father.ballX += father.velocity;                  //X 方向上的移动
15           if(checkCollision()){                             //检测是否发生碰撞
16               father.ballX -= father.velocity;              //若发生碰撞就撤销移动
17           }
18           break;
19       ……//此处省略对方向 2、3、4、5 的处理
20       case 6:                                               //方向向左
21           father.ballX -= father.velocity;                  //X 方向上的移动
22           if(checkCollision()){                             //若发生碰撞就撤销移动
23               father.ballX += father.velocity;              //若发生碰撞就撤销移动
24           }
25           break;
26       case 7:                                               //方向左上
27           father.ballX -= father.velocity;                  //X 方向上的移动
28           if(checkCollision()){                             //若发生碰撞就撤销移动
29               father.ballX += father.velocity;              //若发生碰撞就撤销移动
30           }
31           father.ballY -= father.velocity;                  //Y 方向上的移动
32           if(checkCollision()){                             //若发生碰撞就撤销移动
33               father.ballY += father.velocity;              //若发生碰撞就撤销移动
34           }
35           break;
36       }}
```

说明

checkAndMoveBall 方法的原理就是根据小球的方向进行 x 和 y 方向上的移动，每次对 x 或 y 方向上进行移动之后，都要进行碰撞检测，如果发生碰撞就撤销本次的移动。需要说明的是，对小球进行移动和碰撞检测时 x 方向和 y 方向是独立进行的，因此，如果一个方向上因发生碰撞而需要撤销移动时，不会影响到另一个方向。

14.8.2 小球的碰撞检测

在 checkAndMoveBall 中每次进行碰撞检测时，都会调用 GameThread 的成员方法 checkCollision，该方法返回一个 boolean 值，返回 true 表示发生碰撞，返回 false 表示没有发生碰撞，同时有关碰撞

处理的代码也写在了 checkCollision 方法中。

在 14.6.3 节介绍 GameThread 时对于 checkCollision 方法仅提供了空实现，下面就来具体介绍 checkCollision 代码，其步骤如下。

（1）碰撞检测中，首先判断小球是否与地图矩阵中没有图元的地方以及屏幕的 4 个边界发生碰撞，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameThread.java。

```
1    public boolean checkCollision() {
2        int size = father.tileSize;                    //获取地图图元大小
3        byte [][] map = father.currMap;                //获取地图矩阵
4        int row=0;
5        int col=0;
6        //检查左上角
7        col = father.ballX/size;                       //求出左上角在地图中所占的列数
8        row = father.ballY/size;                       //求出左上角在地图中所占的行数
9        if(col>=0 && col<map[0].length && row>=0&&row<map.length && map[row][col] ==
0){
10           return true;
11       }
12       //检查右上角
13       col = (father.ballX + size-1)/size;            //求出右上角在地图中所占的列数
14       row = father.ballY/size;                       //求出右上角在地图中所占的行数
15       if(col>=0 && col<map[0].length && row>=0&&row<map.length && map[row][col] ==
0){
16           return true;
17       }
18       //检查右下角
19       col = (father.ballX+size-1)/size;              //求出右下角在地图中所占的列数
20       row = (father.ballY+size-1)/size;              //求出右下角在地图中所占的行数
21       if(col>=0 && col<map[0].length && row>=0&&row<map.length && map[row][col] ==
0){
22           return true;
23       }
24       //检查左下角
25       col = father.ballX/size;                       //求出左下角在地图中所占的列数
26       row = (father.ballY+size-1)/size;              //求出左下角在地图中所占的行数
27       if(col>=0 && col<map[0].length && row>=0&&row<map.length && map[row][col] ==
0){
28           return true;
29       }
30       if(father.ballX < 0){                          //检查有没有出左边界
31           return true;
32       }
33       if(father.ballY < 0){                          //检查有没有出上边界
34           return true;
35       }
36       if(father.ballX + size > father.currMap[0].length * size){    //检查有没有出右边界
37           return true;
38       }
39       if(father.ballY + size > father.currMap.length*size){    //检查有没有出下边界
40           return true;
41       }
42       ……//此处省略与地图中陷阱等障碍以及加命奖励的碰撞，将在随后的步骤补全
43       return false;
44   }
```

- 代码第 6～11 行为检测小球的左上角是否遇到了地图矩阵中图元为空的地方，首先计算出

左上角位置在地图矩阵中对应到哪一行哪一列，然后判断该位置的图元类型是否为 0（即什么也没有），如果为 0 说明发生碰撞，返回 true。

- 代码第 12～17 行、第 18～23 行、第 24～29 行分别为检查小球的右上角、右下角和左下角是否与地图中的空白发生碰撞。需要注意的是，在计算其他点如右下角对应的地图图元时，是在 x 和 y 坐标的基础上加上 size-1 个像素，而不是 size 个像素。
- 代码第 30～41 分别为检查小球是否超出了左、上、右和下边界的代码。

（2）检测完小球是否与地图中的空白发生碰撞后，就需要继续检查其是否与游戏中的障碍物或奖励标志发生碰撞了，步骤（1）中代码第 42 行省略的代码框架如下。

```
1     ……
2     col = father.ballX/size;                          //计算小球左上角对应的地图矩阵列数
3     row = father.ballY/size;                          //计算小球左上角对应的地图矩阵行数
4     if(col == (father.ballX + size-1)/size && row==(father.ballY+size-1)/size){
                                                        //判断小球是否正好在某地图格子中
5          switch(father.currMap[row][col]){
6          case 2:                                      //到家了
7               ……//暂时省略处理代码，随后补全
8               break;
9          case 3:                                      //碰到加“命”的标志
10              ……//暂时省略处理代码，随后补全
11              break;
12         case 4:                                      //碰到吃人漩涡
13              ……//暂时省略处理代码，随后补全
14              break;
15         case 5:                                      //碰到陷阱
16              ……//暂时省略处理代码，随后补全
17              break;
18         default:
19              break;
20         }}
21    return false;
```

- 代码第 2 和 3 行首先计算出小球的左上角位于地图矩阵中的列数和行数。
- 代码第 4 行判断小球左上角位于的地图行列数是否与小球右下角位于的地图行列数相同，如果相同，说明小球目前正好完全地位于地图中的某一个格子中。

满足代码第 4 行的条件之后，就需要判断小球位于的格子属于哪种图元了，在游戏中需要处理的图元为家、加命奖励、吃人漩涡和陷阱，上述代码只是列出了处理碰撞的框架，省略了具体的代码。下面就来分别对这几种情况的处理方式进行介绍。

（3）首先是遇到的图元为家的情况，下述的代码来自于步骤（2）代码第 7 行省略的部分。

```
1     father.status = STATUS_WIN;                       //设置游戏状态为 STATUS_WIN
2     if(father.father.wantSound){                      //如有需要播放声音
3             try {
4                  father.father.mpGameWin.start();
5             } catch (Exception e) {}
6     }
7     isGameOn = false;                                 //暂停 GameThread 的执行
8     if(father.father.level == MAX_LEVEL){             //检查是否是最后一关
9             father.status = STATUS_PASS;              //设置游戏状态为 STATUS_PASS
10    }
11    else{
12            father.father.level += 1;                 //如果不是最后一关，就将关卡加 1
```

```
13    }
```

说明　在碰到家标志时，首先将游戏状态设置为 STATUS_WIN，然后根据需要播放声音，随即检查目前是否已经是最后一关，如果是就将游戏状态设置为 STATUS_PASS，否则就将关卡加 1。

（4）接下来是碰到加命奖励时的处理代码，步骤（2）代码第 10 行省略的代码如下。

```
1     if(father.father.wantSound){                          //如有需要播放声音
2             try{
3                 father.father.mpPlusLife.start();
4             }
5             catch(Exception e){}                          //捕获并打印异常
6     }
7     father.ballX = col*size;                              //修正小球的 X 坐标
8     father.ballY = row*size;                              //修正小球的 Y 坐标
9     father.currMap[row][col] = 1;                         //将地图矩阵中该位置改为道路
10    father.father.life += 1;                              //小球的生命数加 1
```

说明　处理与小球与加命奖励的碰撞比较简单，不需要修改游戏的状态位。在碰到加命奖励后先根据需要播放声音，然后将小球的 x、y 坐标进行修正使其刚好处于地图图元中。在将小球的生命数加 1 之后，还需要将地图中该位置的图元类型设置为可通过的路。

（5）接下来是处理碰到陷阱时的情况，本游戏中的陷阱是一条时通时断的机关，所以在小球位于陷阱中时，还需要判断当前的陷阱是通路还是断路。步骤（2）中代码第 16 行省略的代码如下。

```
1     if(father.trapIndex  > 3){                        //判断目前陷阱的动画帧，大于 3 是断路的帧
2          father.status = STATUS_LOSE;                 //设置状态位 STATUS_LOSE
3          if(father.father.wantSound){
4               try {
5                    father.father.mpBreakOut.start();
6               } catch (Exception e) {}
7          }
8          father.father.life -= 1;                          //生命数减 1
9          if(father.father.life<=0){                        //如果生命耗尽
10              father.status = STATUS_OVER;                 //设置状态为 STATUS_OVER
11         }
12         else{                                             //如果还有生命
13              father.ballX = 0;                            //将小球的 X 坐标复位
14              father.ballY = 0;                            //将小球的 Y 坐标复位
15         }}
```

说明　判断当前陷阱为断路后，将游戏状态设置为 STATUS_LOSE，然后根据需要播放声音，并将小球的生命数减 1，随即判断小球的生命数是否已经耗尽，如果耗尽则将状态设置为 STATUS_OVER，否则只是将小球的坐标复位，继续游戏。

处理小球碰到吃人漩涡的代码与陷阱基本类似，惟一的不同是小球碰到吃人漩涡时不需要再进行进一步的判断，直接进行 STATUS_LOSE 状态设置和声音播放等操作。

14.8.3 大炮相关类 Cannon 和 Missile 的开发

游戏中的障碍除了前面小节中介绍的陷阱和吃人漩涡外，还有一个就是大炮。大炮属于比较厉害的障碍物，所以其只在后面的关卡中出现，本小节就来介绍大炮的开发。其中主要涉及的类有 Cannon、Missile、CannonThread，其开发步骤如下。

（1）Missile 类的开发，Missile 类中主要成员变量包含炮弹的位置坐标和移动方向，同时还包括一个成员方法用于与小球进行碰撞检测。Missile 类的代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 Missile.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import static wyf.wpf.DriftBall.*;                        //引入 DriftBall 类的静态成员
3    public class Missile{
4         float k;                                             //斜率
5         int span=2;                                          //炮弹步进
6         int x;                                               //导弹中心 x 坐标
7         int y;                                               //导弹中心 y 坐标
8         GameView father;                                     //GameView 引用
9         //构造器，初始化小球的位置、速度和方向
10        public Missile(int x,int y,float k,int span,GameView father){
11             this.x = x;
12             this.y = y;
13             this.k = k;
14             this.span = span;
15             this.father = father;
16        }
17        //方法：移动导弹并检测是否与小球相撞
18        public void moveAndCheck(){
19             //移动导弹
20             x = x+span;                                      //根据步进移动 X 坐标
21             y = (int)(y+ k*span);                            //根据一定斜率移动 Y 坐标
22             //检测是否碰到目标
23             int bx = father.ballX+10;                        //中心 x 坐标
24             int by = father.ballY+10;                        //中心 y 坐标
25             if((bx-x)*(bx-x)+(by-y)*(by-y) <= 15*15){
26                  father.father.life -=1;                     //小球生命数减 1
27                  if(father.father.life <=0){                 //所有命都已经耗完
28                       father.gt.flag = false;                //停止 GameThread 的执行
29                       father.gt.isGameOn = false;
30                       father.status = STATUS_OVER;           //设置游戏状态为 STATUS_OVER
31                  }
32                  else{                                       //还有生命，那就从头开始
33                       father.ballX = 0;                      //复位小球 X 坐标
34                       father.ballY = 0;                      //复位小球 Y 坐标
35                  }
36                  father.alMissile.remove(this);              //从 GameView 的炮弹列表中移除自己
37             }
38             //检测是否碰到边界
39             if(x > father.screenWidth || x<0 || y>father.screenHeight || y<0){
40                  father.alMissile.remove(this);              //从 GameView 的炮弹列表移除自己
41             }}}
```

- 代码第 4 行声明的变量 k 用来表示炮弹的方向，由于炮弹的运动轨迹为一条直线，故其运动方向可用其斜率来代表。
- 代码第 10～16 行为 Missile 类的构造器，自其中进行位置坐标和斜率及步进的初始化工作。

- 代码第 18～41 行为 moveAndCheck 方法，该方法负责移动炮弹并随即检测是否与小球发生碰撞。
- 代码第 20～21 行为移动炮弹的代码，x 方向上的移动根据步进来计算，y 方向上的移动还需要利用斜率来计算。
- 代码第 25 行为检测炮弹是否与小球发生碰撞的代码，检测的依据是二者的中心点之间的距离是否小于二者的半径之和。
- 代码第 26～36 行为发生碰撞后的处理代码。首先将小球的生命数减 1，然后判断小球的生命数是否已经耗尽，如果耗尽则停止 GameThread 的执行并设置游戏状态为 STATUS_OVER。如果小球的生命数还没耗尽，则让小球复位重新开始过关。
- 代码第 39～41 行为检测炮弹是否超出边界的代码，如果小球超出边界，将其从 GameView 中的 Missile 对象列表中移除。

（2）Cannon 类的开发，Missile 类开发完成后，下面来开发 Cannon 类，每个 Cannon 对象代表一门大炮，炮弹也是通过 Cannon 类的成员方法创建的，Cannon 类的代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 Cannon.java。

```
1    package wyf.wpf;                                          //声明包语句
2    public class Cannon{
3         GameView father;                                     //GameView 对象引用
4         int row;                                             //大炮在地图上的行数
5         int col;                                             //大炮在地图上的列数
6         int range;                                           //射程
7         //构造器，初始化成员变量
8         public Cannon(int row,int col,GameView father){
9              this.row = row;                                 //初始化大炮的行数
10             this.col = col;                                 //初始化大炮的列数
11             this.father = father;                           //初始化 GameView 引用
12             this.range = 840;                               //初始化大炮的射程
13        }
14        //方法：产生一个 Missile 对象
15        public Missile fire(){
16             int missileX = col*father.tileSize + 10;        //确定炮弹的起始 X 坐标
17             int missileY = row*father.tileSize + 10;        //确定炮弹的起始 Y 坐标
18             float k = (float)(missileY-father.ballY)/(missileX-father.ballX);
                                                               //计算炮弹的斜率
19             int span = missileX>father.ballX?-2:2;          //判断是向左运动还是向右移动
20             return new Missile(missileX,missileY,k,span,father);
                                                               //创建 Missile 对象并返回
21        }}
```

- 代码第 4 和 5 行声明了记录大炮在地图中的行数和列数的成员变量，该值将在代码第 8～13 行的构造器中被初始化。
- 代码第 14～21 行为 fire 方法的代码，代码第 16 和 17 行分别计算出炮弹初始时刻的 x、y 坐标，代码第 18 行根据小球的位置和大炮的位置确定二者之间直线的斜率；代码第 19 行根据炮弹和小球的 x 坐标的对比确定炮弹在坐标系中的运动时沿 x 轴负向还是正向。
- 代码第 20 行根据之前计算出的数据创建一个 Missile 对象并将其返回。

（3）在 14.6.1 小节介绍 GameView 中的成员变量时，曾对 alCannon 列表做过说明，该列表中存放游戏地图中的大炮，向该列表添加内容是通过 GameView 类的 checkForCannon 方法完成的，

下面就来介绍 checkForCannon 的具体开发，其代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 GameView.java。

```
1    public Boolean checkForCannon(){                    //方法：检查地图中是否有大炮
2         boolean result = false;
3         for(int i=0;i<currMap.length;i++){             //对地图进行遍历
4              for(int j=0;j<currMap[i].length;j++){
5                   if(currMap[i][j] == 8){               //如果地图图元为大炮
6                        Cannon c = new Cannon(i,j,this);//根据其行列值创建 Cannon 对象
7                        alCannon.add(c);                //将 Cannon 对象添加到 GameView 的列表中
8                        result = true;                  //将 result 赋值为 true
9                   }}}
12        return result;                                 //返回 result
13   }
```

说明　checkForCannon 方法在每次加载新关卡后调用，其主要进行的工作是遍历整个地图矩阵，如果碰到图元为大炮的，就根据其行列值创建一个 Cannon 对象，并将其添加到 GameView 的存放 Cannon 对象的列表 alCannon 中。

（4）CannonThread 类的开发，前面的步骤介绍了如何创建 Cannon 和 Missile 对象，而真正控制大炮向小球发射炮弹的是 CannonThread 类，该类的代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 CannonThread.java。

```
1    package wyf.wpf;                                          //声明包语句
2    public class CannonThread extends Thread{
3         GameView father;                                     //GameView 类引用
4         int sleepSpan = 2000;                                //休眠时间
5         boolean flag = false;                                //主循环变量
6         boolean isGameOn = false;                            //游戏是否进行中
7         //构造器
8         public CannonThread(GameView father){
9              this.father = father;
10             if(father.alCannon.size()!=0){          //检查 Cannono 对象列表是否为空
11                  flag = true;
12                  isGameOn = true;
13             }
14        }
15        //线程的执行方法
16        public void run(){
17             while(flag){
18                  while(isGameOn){
19                       //遍历 Cannon 对象集合
20                       for(Cannon c:father.alCannon){
21                            //求出大炮和小球之间的距离
22                            int distance = Math.abs(c.col*father.tileSize-father.ballX)
23                                 + Math.abs(c.row*father.tileSize-father.ballY);
24                            if(distance <= c.range){          //如果小球处于射程之内
25                                 Missile m = c.fire();        //创建炮弹对象
26                                 father.alMissile.add(m);
                                                          //将炮弹对象添加到炮弹对象的集合中
27                            }
28                       }
29                       try{Thread.sleep(sleepSpan); }        //线程休眠
32                       catch(Exception e){e.printStackTrace();}
35                  }
36                  try{Thread.sleep(1000);}                   //外层循环休眠 1s
39                  catch(Exception e){e.printStackTrace();}
```

```
41                  }}}
```

- 代码第 5 和 6 行声明了两个 boolean 变量，分别为线程是否执行标志位和游戏是否在进行标志位，采用这两个标志位可以实现线程暂停的效果。
- 代码第 8～14 行为 CannonThread 类的构造器，在其中根据 GameView 中的 Cannon 对象列表的大小进行初始化的工作。
- 代码第 16～41 行为线程的执行方法，其主要进行的工作是遍历 Cannon 对象列表，对每个 Cannon 对象计算其与小球之间的距离。如果小球和大炮的距离在大炮的射程之内，则通过 Cannon 对象的 fire 方法发射一发炮弹并将该炮弹对象添加到 GameView 的炮弹列表中。

14.9 传感器计算模块的开发

本节将会介绍最后一个模块——开发传感器计算模块，该模块的功能十分简单，就是捕获手机姿态，并根据收到的数据计算出小球的方向，但实现起来比较复杂。该模块涉及的类主要为 BallListener 和 RotateUtil。

14.9.1 手机姿态变化监听器——BallListener 类的开发

BallListener 类继承自 SensorListener，其主要的功能是对手机的姿态变化进行监听，BallListener 类的代码如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 BallListener.java。

```
1    package wyf.wpf;                                          //声明包语句
2    import android.hardware.SensorListener;                   //引入相关类
3    import android.hardware.SensorManager;                    //引入相关类
4    public class BallListener implements SensorListener{
5         DriftBall father;                                    //DriftBall 引用
6         int timeSpan = 500;                                  //500MS 检查一次
7         long startTime;                                      //记录开始的时间
8         //构造器，初始化成员变量
9         public BallListener(DriftBall father){
10             this.father = father;
11             startTime = System.currentTimeMillis();         //记录系统时间作为当前时间
12        }
13        //重写 onAccuracyChanged 方法
14        public void onAccuracyChanged(int sensor, int accuracy) {}
15        //重写 onSensorChanged 方法
16        public void onSensorChanged(int sensor, float[] values) {
17             long now = System.currentTimeMillis();          //获取系统当前时间
18             if(now - startTime >= timeSpan){                //判断是否走过指定的时间间隔
19                  if(sensor == SensorManager.SENSOR_ORIENTATION){
20                       analysisData(values);                 //调用 analysisData 方法分析数据
21                  }
22                  startTime = now;                            //重新计时
23             }
24        }
```

```
25        public void analysisData(float[] values) {//对姿态数据进行分析，调用 RoateUtil 的
                                                    //静态方法计算出方向
26            double[] valuesTemp=new double[]{values[0],-values[1],values[2]};
                                                          //对 y 轴进行修正
27            father.gv.direction = RotateUtil.getDirectionCase(valuesTemp);
                                                          //计算出小球方向
28        }}
```

- 代码第 6 行声明了计算的时间间隔，在监听器监听过程每隔固定的时间获取手机姿态进行计算。代码第 7 行声明了用于记录开始时间的成员变量，当系统时间与开始时间的差等于 timeSpan 时，获取手机姿态进行计算。
- 代码第 9～12 行为 BallListener 类的构造器，在其中初始化主要的成员变量。
- 代码第 13 行为重写的 onAccuracyChanged 方法，在此为空实现。
- 代码第 16～24 行为重写的 onSensorChanged 方法，在该方法中每隔固定的时间获取手机姿态，并调用 analysisData 方法进行计算。在每次计算结束后，将当前系统时间记录到 startTime 成员变量中重新计时。
- 代码第 25～28 行为 analysisData 方法的代码，在该方法中将获取到的 values 数组进行修正后，调用 RotateUtil 类的 getDirectionCase 方法计算出小球的方向。

BallListener 对象的创建是在 DriftBall 类的 onCreate 方法中进行的，而注册和取消注册监听器的工作则放在 startSensor 和 pauseSensor 方法中进行，这两个方法在 14.4.1 小节已经做过介绍。

14.9.2 工具类——RotateUtil 类的代码框架

前面在 BallListener 类的 analysisData 方法中，提到了 RotateUtil 类的 getDirectionCase 方法。RotateUtil 类的主要工作是对捕获的手机姿态数据进行计算得出小球的运动方向，这些计算涉及一些数学和物理的知识，也是本游戏中比较难理解的部分，下面就来介绍 RotateUtil 的计算原理。

第 4 章介绍手机姿态传感器相关知识时曾提到，手机的姿态传感器捕获的数据为手机沿 Yaw 轴、Pitch 轴和 Roll 轴转过的角度。本游戏中的小球是受重力而运动的，不管手机如何翻转，小球都是沿着重力加速度在手机屏幕上的分量进行运动。所以只需要求出重力加速度向量在手机屏幕上的投影，就可以求得小球的运动方向。

在手机姿态系统中，重力加速度向量的方向总是与 Yaw 轴相反，竖直向下。而手机屏幕则根据旋转位于任意的平面，因此计算重力加速度向量在手机屏幕上的投影将会比较困难。本游戏中，采用逆转的方式来求向量投影，其步骤如下。

- 初始情况下，手机已经进行了任意地翻转，此时重力加速度向量与手机屏幕平面呈现一定夹角。
- 保持重力加速度向量与手机屏幕平面的相对关系不变，即相当于将重力加速度向量焊死到手机屏幕所在的平面上。将手机沿着刚才转过的轨迹转回，例如，原先绕 Yaw 轴逆时针转过 30°，现在沿 Yaw 轴顺时针转回 30°。
- 将手机沿 3 个方向的旋转都恢复之后，手机将处于初始状态，即 Yaw、Pitch 和 Roll 轴分别可对应于空间坐标系中的 z、x 和 y 轴。

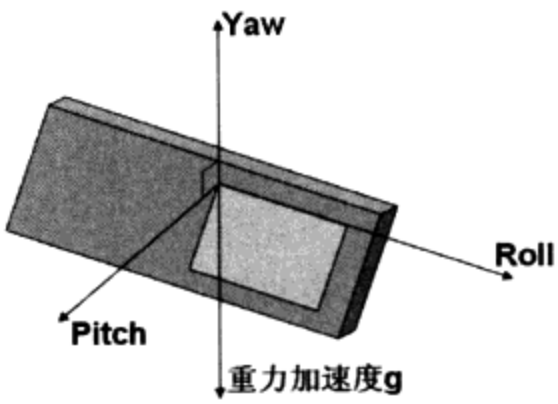
- 此时重力加速度向量将会是一个与手机屏幕（即 x、y 轴确定的平面），相交的向量，将该向量投影到 xoy 平面上，根据其投影向量的方向即可得出小球的运动方向。

上述算法并不是将固定不变的重力加速度向量投影到变化莫测的手机屏幕平面上，而是通过转回手机姿态将虚拟出来的重力加速度（即焊死在手机屏幕上的）投影到已处在空间坐标系中 xoy 平面上的手机屏幕上，这样主要的计算就是如何把手机沿各方向将转过的角度转回了。

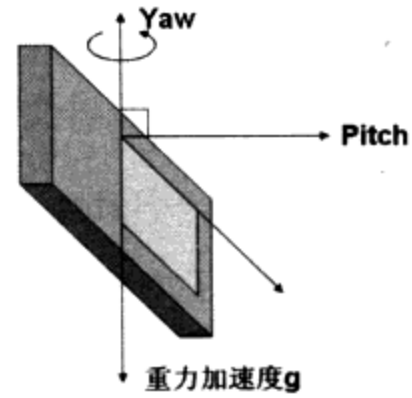
说明　上述算法其实就是把向量投影到任意平面上的复杂计算简化为旋转平面，再将向量投影到坐标平面上的计算。实际游戏开发中经常需要通过采用一些方法将复杂计算简化，读者可以多总结这方面的经验。

这里需要特别注意沿 3 个空间方向 Yaw、Pitch 和 Roll 轴所旋转角度的恢复顺序。本算法中，其顺序必须为 Yaw、Pitch、Roll。下面通过图示对这 3 个轴的角度恢复进行说明，起始状态如图 14-16 所示，此时手机已经经过了 3 个轴的旋转，位于任意平面上。

- 由于 Yaw 轴始终指向竖直向上方向（重力加速度反向），和标准的空间坐标系的 z 轴一样，所以可以通过负向旋转直接进行角度恢复，如图 14-17 所示。
- 沿 Yaw 轴将转过的角度恢复后，此时的 Pitch 轴就变成了空间坐标系中的 x 轴，如图 14-17 所示，同样对 Pitch 轴负向旋转进行角度恢复。

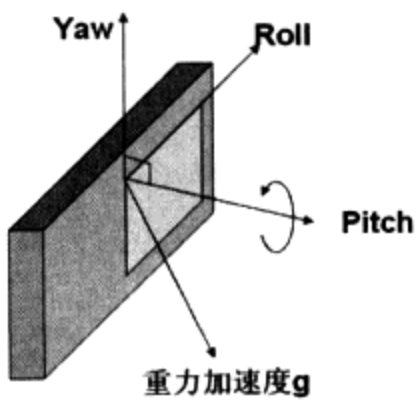


▲图 14-16　手机旋转后示意图

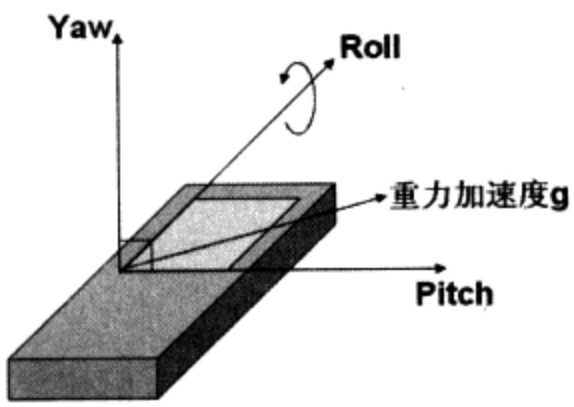


▲图 14-17　进行 Yaw 轴恢复后

沿 Pitch 轴（此时已经为空间坐标系的 x 轴）将转过的角度恢复后，此时的 Roll 轴就修正为了空间坐标系中的 y 轴，如图 14-18 所示，最后按照 y 轴将转过的角度恢复，则此时手机屏幕所在的平面就变成了空间坐标系中 x，y 平面，而附着于手机平面上的重力加速度则是一个与手机平面相交的向量，如图 14-18 所示。



▲图 14-18　进行 Pitch 轴恢复后



▲图 14-19　进行 Roll 轴恢复后

最后将该向量投影到手机平面，通过投影点就可以计算出小球要滚动的方向。如果不按照上述顺序进行角度恢复，则空间坐标的计算转换将会非常复杂。而上述方法中每一步的角度恢复都是基于标准的空间坐标系轴，而对标准坐标轴的转换在计算机图形学中已经有很成熟的实现，直接采用即可。

下面通过介绍 RotateUtile 类的代码来进一步理解各个方向轴是如何旋转和恢复的，其代码框架如下。

代码位置：见随书光盘中源代码/第 14 章/ DriftBall/src/wyf/wpf 目录下的 RotateUtil.java。

```
1    package wyf.wpf;                                          //声明包语句
2    public class RotateUtil{
3          //指定向量沿 X 轴旋转指定角度，返回值为旋转后的向量
4          public static double[] XRotate(double angle,double[] gVector){}
5          //指定向量沿 Y 轴旋转指定角度，返回值为旋转后的向量
6          public static double[] YRotate(double angle,double[] gVector){}
7          //指定向量沿 Z 轴旋转指定角度，返回值为旋转后的向量
8          public static double[] ZRotate(double angle,double[] gVector){}
9          //方法：调用 XRotate、YRotate 和 ZRotate 方法计算小球的运动方向
10         public static int getDirectionCase(double[] values){}
11   }
```

- RotateUtile 类中主要定义了用于向量旋转计算的静态工具方法，其中代码第 4～8 行定义了用于恢复 3 个方向上旋转角度的方法。
- 代码第 10 行为 getDirectionCase 方法，该方法将调用 XRotate、YRotate 和 ZRotate 方法计算小球的方向，并将结果以代表方向的 0～7 八个数值的形式返回。

14.9.3 工具类——RotateUtil 类的开发

上一小节介绍了通过手机姿态计算小球方向的原理及 RotateUtil 类的代码框架，本节就来对 RotateUtil 类中的方法开发进行详细介绍，其步骤如下。

（1）开发沿 3 个方向轴进行角度恢复的方法 XRotate、YRotate 和 ZRotate 的代码，这 3 个方法分别负责某一个坐标轴上的角度恢复，具体代码如下。

```
1    //指定向量沿 X 轴旋转指定角度，返回值为旋转后的向量
2    public static double[] XRotate(double angle,double[] gVector){
3          double[][] matrix={//绕 x 轴旋转变换矩阵
4             {1,0,0,0},                                     //此矩阵的元素是固定的
5             {0,Math.cos(angle),Math.sin(angle),0},         //来自于数学家的研究成果
6             {0,-Math.sin(angle),Math.cos(angle),0},        //其中 angle 表示要旋转的角度
7             {0,0,0,1}
8          };
9          double[] tempDot={gVector[0],gVector[1],gVector[2],gVector[3]};
                                                             //复制旋转前的向量到临时向量中
10         for(int j=0;j<4;j++){                             //将向量与旋转矩阵相乘，得到
11               gVector[j]=(tempDot[0]*matrix[0][j]+tempDot[1]*matrix[1][j]+
                                                             //旋转后的向量
12                         tempDot[2]*matrix[2][j]+tempDot[3]*matrix[3][j]);
13         }
14         return gVector;
15   }
16   //指定向量沿 Y 轴旋转指定角度，返回值为旋转后的向量
17   public static double[] YRotate(double angle,double[] gVector){
18         double[][] matrix={//绕 y 轴旋转变换矩阵
```

```
19            {Math.cos(angle),0,-Math.sin(angle),0},        //此矩阵的元素是死的
20            {0,1,0,0},                                     //来自于数学家的研究成果
21            {Math.sin(angle),0,Math.cos(angle),0},         //其中 angle 表示要旋转的角度
22            {0,0,0,1}
23        };
24        double[] tempDot={gVector[0],gVector[1],gVector[2],gVector[3]};
                                                             //拷贝旋转前的向量到临时向量中
25        for(int j=0;j<4;j++){                              //将向量与旋转矩阵相乘，得到
26             gVector[j]=(tempDot[0]*matrix[0][j]+tempDot[1]*matrix[1][j]+ //旋转后的向量
27                         tempDot[2]*matrix[2][j]+tempDot[3]*matrix[3][j]);
28        }
29        return gVector;
30    }
31    //指定向量沿 z 轴旋转指定角度，返回值为旋转后的向量
32    public static double[] ZRotate(double angle,double[] gVector){
33        double[][] matrix={//绕 z 轴旋转变换矩阵
34            {Math.cos(angle),Math.sin(angle),0,0},         //此矩阵的元素是死的
35            {-Math.sin(angle),Math.cos(angle),0,0},        //来自于数学家的研究成果
36            {0,0,1,0},                                     //其中 angle 表示要旋转的角度
37            {0,0,0,1}
38        };
39        double[] tempDot={gVector[0],gVector[1],gVector[2],gVector[3]};
                                                             //复制旋转前的向量到临时向量中
40        for(int j=0;j<4;j++){                              //将向量与旋转矩阵相乘，得到
41             gVector[j]=(tempDot[0]*matrix[0][j]+tempDot[1]*matrix[1][j]+
                                                             //旋转复制后的向量
42                         tempDot[2]*matrix[2][j]+tempDot[3]*matrix[3][j]);
43        }
44        return gVector;
45    }
```

- 代码第 2～15 行为 XRotate 方法的代码，该方法功能为指定向量沿 x 轴旋转指定角度，返回值为旋转后的向量。
- 代码第 17～30 行为 YRotate 方法的代码，该方法功能为指定向量沿 y 轴旋转指定角度，返回值为旋转后的向量。
- 代码第 32～45 行为 ZRotate 方法的代码，该方法功能为指定向量沿 z 轴旋转指定角度，返回值为旋转后的向量。

提示　上述 3 个方法的代码可能读者不太容易理解，读者可以直接使用，不必关心细节。如果想搞清楚其所以然可以参看计算机图形学的相关资料，这部分内容大大超出本书讨论的范围，这里不再赘述。

（2）开发 getDirectionCase 方法，步骤（1）中开发的 3 个方法将被 getDirectionCase 调用以计算小球的方向，其代码如下。

```
public static int getDirectionCase(double[] values){ //方法：计算小球的方向
     double zAngle=-Math.toRadians(values[0]);       //获取 z 轴旋转角度弧度
     double xAngle=-Math.toRadians(values[1]);       //获取 x 轴旋转角度弧度
     double yAngle=-Math.toRadians(values[2]);       //获取 y 轴旋转角度弧度
     double[] gVector={0,0,-100,1};                  //虚拟一个重力向量
     gVector=RotateUtil.ZRotate(zAngle,gVector);     //yaw 轴恢复
     gVector=RotateUtil.XRotate(xAngle,gVector);     //pitch 轴恢复
     gVector=RotateUtil.YRotate(yAngle,gVector);     //roll 轴恢复
     double mapX=gVector[0];              //mapX 为重力向量在 X、Y 轴平面上的投影的 X 坐标
```

```
10        double mapY=gVector[1];                    //mapY 为重力向量在 X、Y 轴平面上的投影的 Y 坐标
11        if(mapX==0){                                          //如果 mapX 等于 0
12             if(mapY>0){                                      //mapY 大于 0
13                  return 0;                                   //返回方向为 0，向上
14             }else{                                           //mapY 小于 0
15                  return 4;                                   //返回 4，方向为向下
16             }
17        }else{
18             if(mapY==0){                                     //如果 mapY 等于 0
19                  if(mapX>0){                                 //如果 mapX 大于 0
20                       return 2;                              //返回 2，方向向右
21                  }else{                                      //如果 mapX 小于 0
22                       return 6;                              //返回 6，方向向左
23                  }
24             }else{
25                  if(mapX>0&&mapY>0){                         //如果向量位于第一象限
26                       return 1;                              //返回 1，方向右上
27                  }else if(mapX>0&&mapY<0){                   //如果向量位于第四象限
28                       return 3;                              //返回 3，方向右下
29                  }else if(mapX<0&&mapY>0){                   //如果向量位于第二象限
30                       return 7;                              //返回 7，方向左上
31                  }else{                                      //向量位于第 3 象限
32                       return 5;                              //返回 5，方向左下
33                  }}}}
```

- 代码第 2～4 行将监听器捕获的数据从角度值转换为弧度值。
- 代码第 5 行虚拟出一个从坐标系原点出发的重力加速度向量，其中{0,0,−100,1}代表该向量终点的坐标为(0,0,−100)并且缩放比例为 1。
- 代码 6～8 行对虚拟的重力向量，分别进行 Yaw、Pitch 和 Roll 轴的恢复。经过代码 6～8 行的计算之后，此时的重力向量将不再是竖直向下，而是一个与手机屏幕（xoy 平面）有一定夹角的向量。实际计算时只需要计算重力向量的旋转即可，并不需要去想手机的问题。
- 代码第 9 和 10 行分别获取了旋转后的重力向量在手机屏幕上的投影，由于此时的手机处于标准的空间坐标系中，故在 xoy 平面上的投影即是空间向量坐标去掉 z 轴分量后的向量。
- 代码第 11～33 行为根据获取的重力向量投影点在 xoy 平面上的位置来得出小球的运动方向。当投影点位于坐标轴上时，其方向为 0、2、4、6 中的一个。而当投影点位于其他位置时，其将根据投影点所在的象限返回不同的方向。

14.10 游戏的优化与改进

到此为止，“小球快跑”游戏已经基本开发完成，也实现了最初设计的功能，但是游戏中仍然还存在一些可以优化和改进的地方，现将这些可提升的内容列举如下。

- 适当添加故事情节

虽然说本游戏的亮点不在于此，而是在于新奇的玩法和快节奏的冒险。但是如果条件允许，适当添加一些不影响玩家玩游戏的故事内容，或许会收到更好的效果。例如，为每一关卡设定背景，用故事将关卡与关卡连接起来等。

- 增加障碍物和奖励

本游戏中的障碍物只有吃人漩涡、陷阱和大炮，可以适当增加更多的障碍物，例如，增加一种

反向障碍物，当小球遇到该障碍物时，小球的方向将会与根据手机姿态计算出的方向相反，这样会大大增加可玩性。同时本游戏中的奖励物品也可以在增加生命数之外多设计几个。

- 增加 AI

游戏中除了漂移的小球，其他的游戏元素均为静态的，可以在游戏中添加一些巡逻的机器人，并赋予这些机器人简单的人工智能，这样可以提高游戏的可玩性。

第 15 章　塔防游戏——精灵塔防

当前移动互联网快速发展，人们生活逐渐趋于多元化，以智能手机为代表的电子产品逐渐受到人们的追捧，手机游戏渐渐成为了人们休闲娱乐的一种方式。本软件是一款基于 Android 系统下采用 2.5D 技术开发的休闲益智类塔防游戏，下面就将带领读者详细地讲解该塔防游戏的开发过程。

15.1 游戏的背景及功能概述

本小节将对精灵塔防游戏进行简单的介绍，使读者对本软件的开发有一个整体的认知，方便读者快速理解并掌握本软件的开发技术。

15.1.1　背景概述

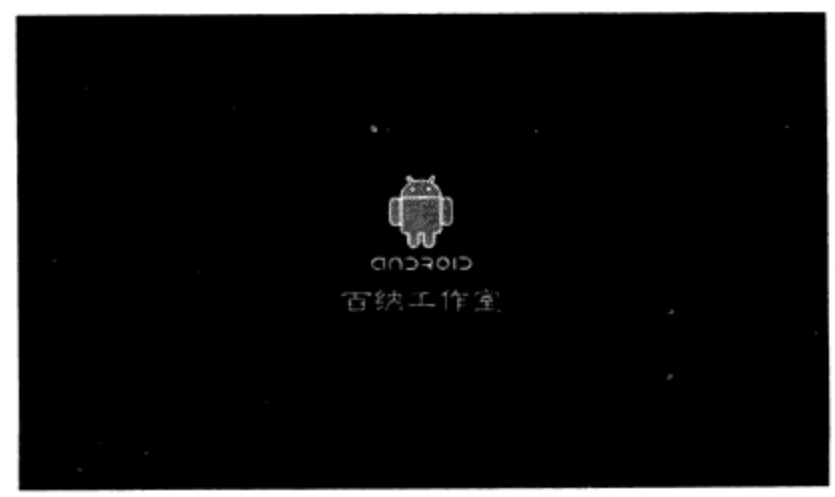
塔防类游戏是目前市场上比较流行的一类游戏，本章介绍的这款塔防游戏沿用了比较流行的守城玩法。本游戏要守的城池有自己的血量，通过铸造箭塔阻挡不断侵袭的怪物精灵，城池血量为零时守城游戏失败。

15.1.2　功能介绍

本节将对精灵塔防游戏的功能以及操作方法进行简单介绍，使读者对本塔防游戏有一个整体的了解，方便读者对后面章节知识的深入学习。下面将分步骤介绍该游戏的简单玩法。

（1）运行该游戏，首先进入欢迎界面，由两张动态的图片组成，效果如图 15-1 和图 15-2 所示。

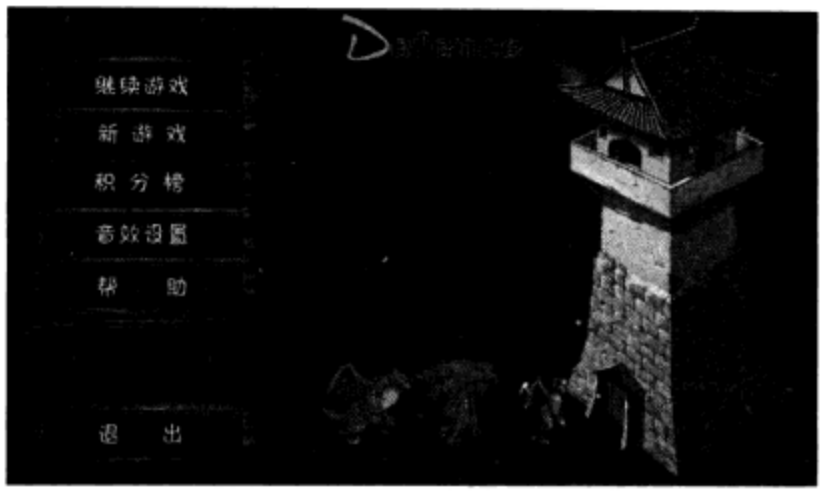
▲图 15-1　欢迎动画 1



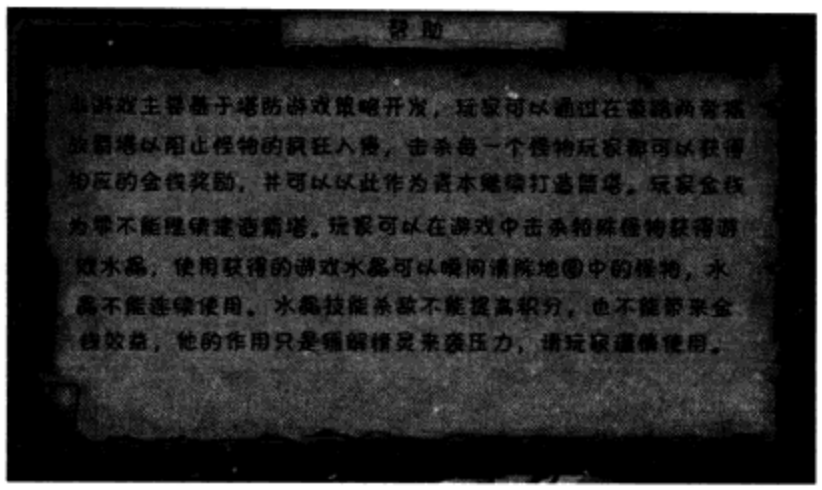
▲图 15-2　欢迎动画 2

（2）欢迎界面完毕以后就进入了该游戏软件的主菜单，效果如图 15-3 所示。

（3）主菜单的左半部分由一组操作菜单组成，单击帮助菜单按钮系统就会进入软件的帮助界面，将会对该游戏软件的玩法做一个简单介绍，帮助界面的效果如图 15-4 所示。



▲图 15-3　主菜单界面



▲图 15-4　帮助界面

（4）在主界面中单击音效设置按钮，可以对游戏中的背景音乐以及炮塔音效等进行开启和关闭操作，音效设置界面的效果如图 15-5 所示。

（5）在主菜单界面中单击积分榜按钮，进入游戏的积分榜界面，里面记载了每一次玩的得分记录并进行高分排序，效果如图 15-6 所示。

▲图 15-5　音效设置界面

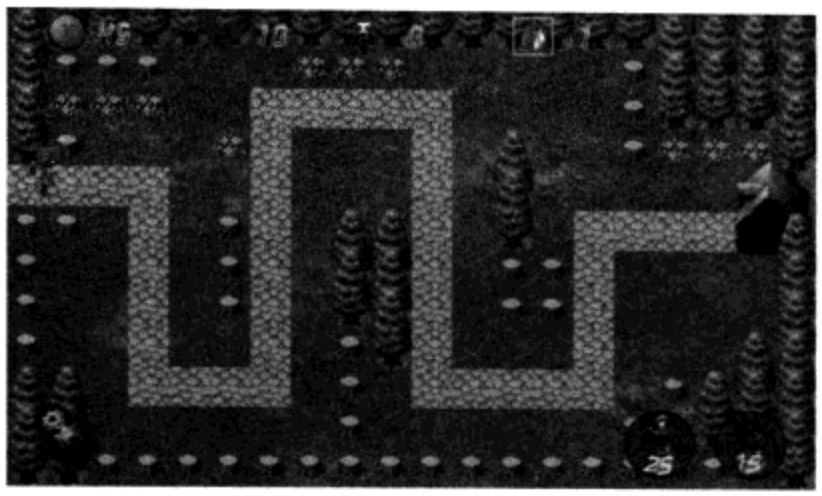
▲图 15-6　积分榜界面

（6）在主界面中单击新游戏就可以进行本软件的游戏过程了。游戏界面的上方包括金钱数、游戏中家的生命数、游戏的杀敌数和游戏中得到的水晶数。界面的下方包含菜单选项和两种可拖动箭塔的图标。游戏界面的初始化效果如图 15-7 所示。

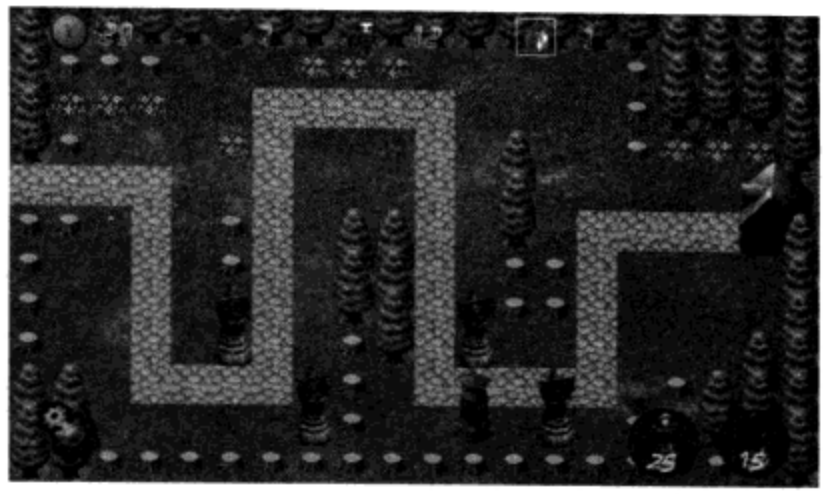
（7）本游戏主要是通过建塔来阻止怪物精灵对家的侵犯，游戏中箭塔的效果如图 15-8 所示。

（8）本游戏采用用手拖动放置箭塔，在有树木公路的地方不准放塔，在可以放箭塔的地方系统将会以红色的圆圈提示玩家此处可以放置箭塔，操作过程的效果如图 15-9 所示。

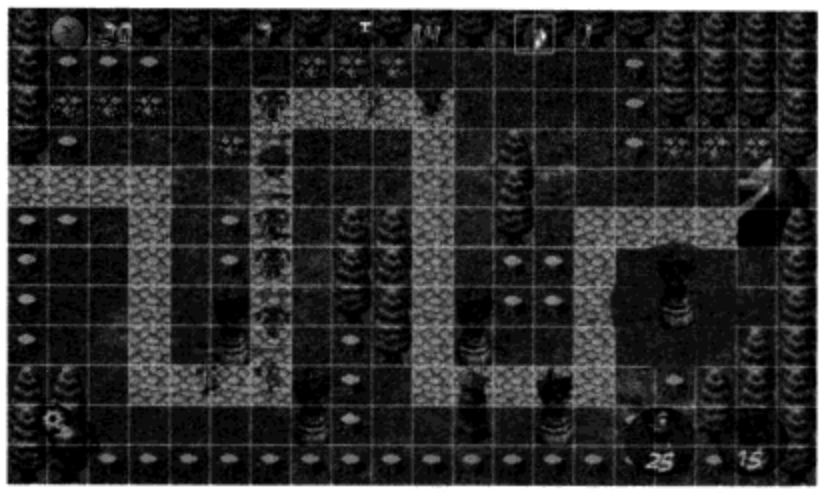
（9）本游戏采用 2.5D 的游戏开发技术，相比较 2D 有一种立体的感觉，游戏中可以看到根据位置的不同怪物精灵可以挡住箭塔，箭塔也可以挡住精灵，游戏运行效果如图 15-10 所示。



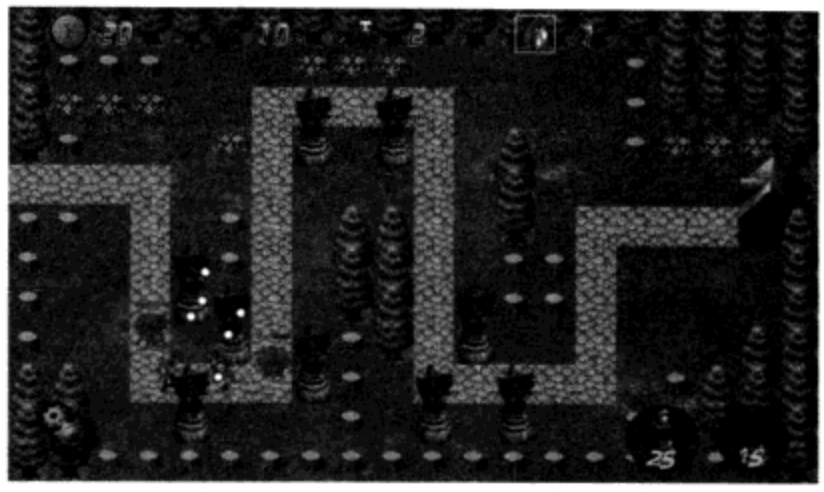
▲图 15-7　游戏初始化界面



▲图 15-8　游戏界面中的箭塔



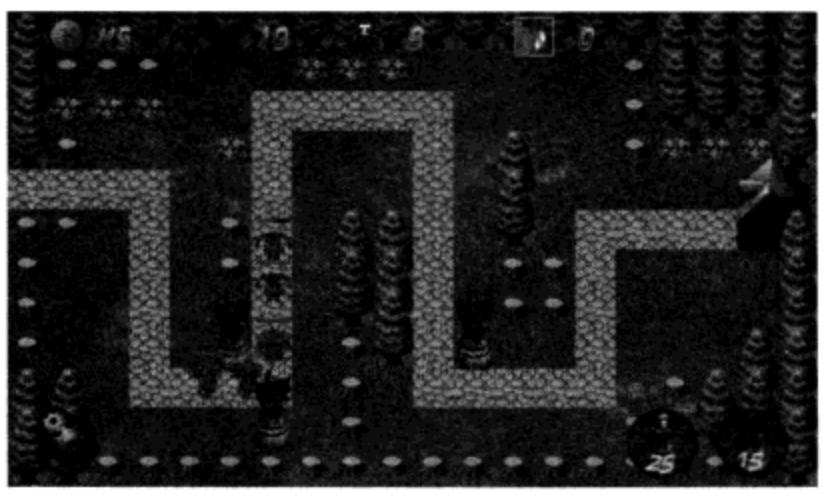
▲图 15-9　箭塔拖动界面



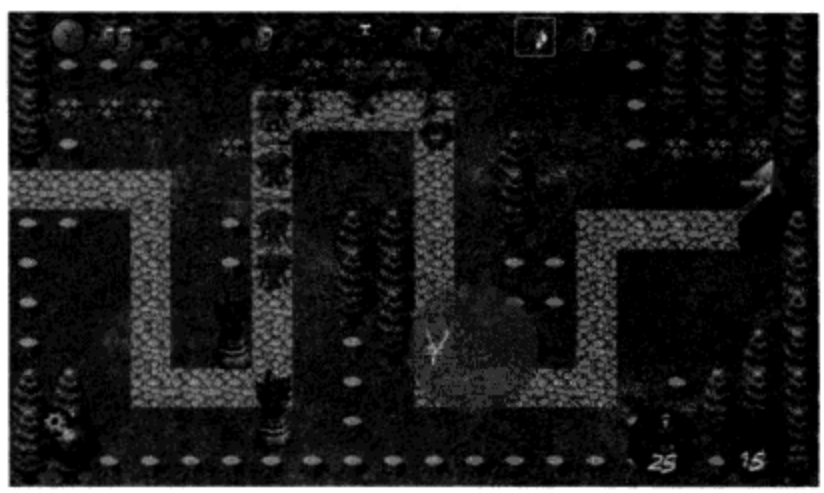
▲图 15-10　游戏界面 2.5D 效果

（10）游戏中每击杀 25 个怪物精灵将会获得系统赠送的 1 个水晶，使用它可以击杀地图上的所有怪物精灵，同时该水晶功能不能连续使用，隔一段时间才能再次使用，效果如图 15-11 所示。

（11）游戏中还可以对放置的箭塔进行操作，触摸当前箭塔出现箭塔操作图层，再次触摸图层就可以去掉当前箭塔。效果如图 15-12 所示。



▲图 15-11　水晶功能的灰色效果

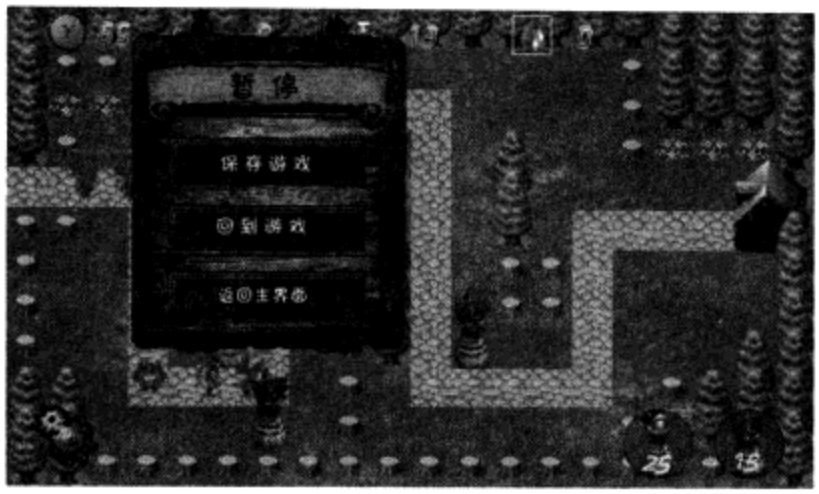


▲图 15-12　箭塔操作界面

（12）触摸界面左下方的菜单按钮可以暂停游戏，进行游戏的一些操作，效果如图 15-13 所示。

（13）单击弹出的菜单返回到游戏选项可以继续进行当前的游戏，单击返回到主菜单则会跳转

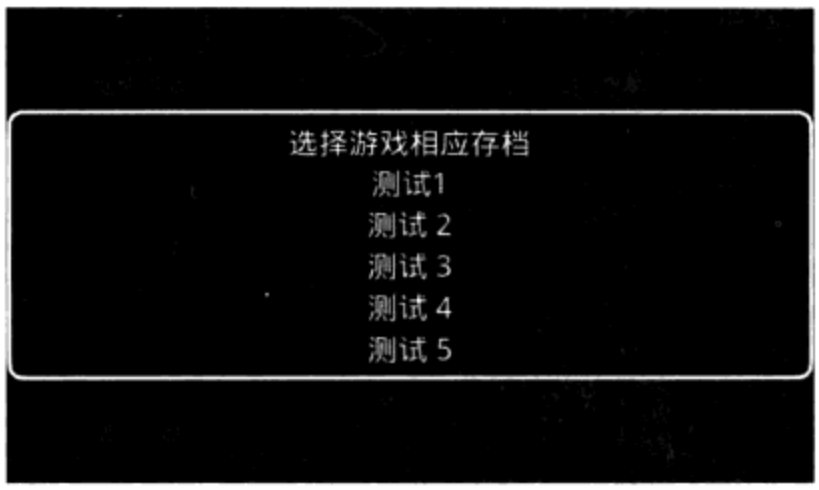
到游戏软件的主菜单界面，单击保存游戏会弹出对话框提示玩家存储当前游戏进度，效果如图 15-14 所示。



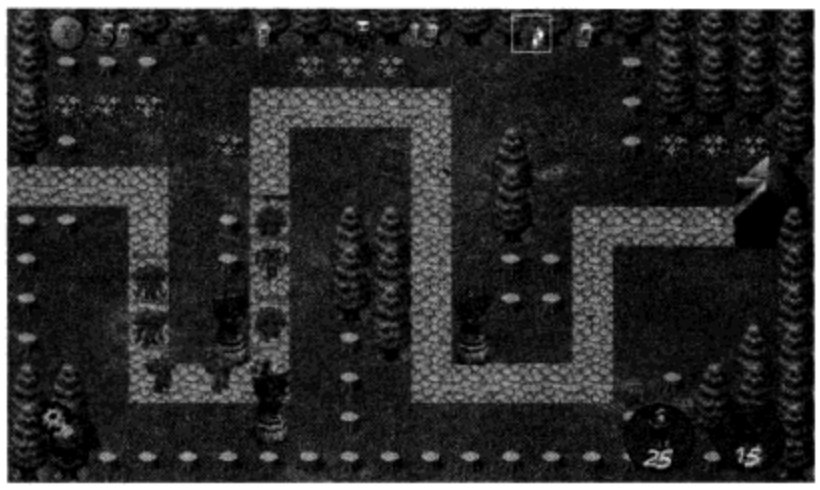
▲图 15-13　弹出暂停菜单

▲图 15-14　游戏存档界面

（14）返回到主界面单击继续游戏，系统会弹出玩家的所有存档记录，供玩家选择，效果如图 15-15 所示。当单击相应的存档记录，玩家可以继续上次存档时的游戏进度继续游戏，效果如图 15-16 所示。



▲图 15-15　读取存档记录界面



▲图 15-16　读取游戏进度界面

15.2 游戏的策划及准备工作

在介绍了游戏的背景和基本功能以后，本节将着重讲解游戏的前期准备工作，这里主要包含游戏的策划和游戏中资源的前期准备。

15.2.1　游戏的策划

本游戏的策划主要包含：游戏类型定位、呈现技术以及目标平台的确定等工作。

- 游戏类型

该游戏的所有操作都为触屏完成，通过摆放箭塔阻止精灵来袭，属于休闲益智类游戏。

- 运行目标平台

游戏目标平台为 Android 2.2，向下兼容 Android 2.1、Android 2.0、Android 1.6。

- 操作方式

本游戏所有关于游戏的操作都是触屏操作，玩家可以拖动箭塔进行塔的放置，同时触摸当前箭塔显现箭塔操作图层，可以进行箭塔操作。

- 呈现技术

相比于 2D 游戏，本游戏采用 2.5D 技术开发，可以有一种立体的效果，随着位置的不同怪物精灵可以遮挡箭塔，同时也可被遮挡。

15.2.2 Android 平台下游戏开发的准备工作

下面做一些开发前的准备工作，包括搜集和制作图片、声音等资源，其详细步骤如下。

（1）首先为读者介绍的是本游戏中除游戏界面外的所要用到的图片资源，系统将所有图片资源都放在项目文件下的 res/drawable-nodpi 文件夹下，如表 15-1 所示。

表 15-1　　其他界面图片清单

图 片 名	大小（KB）	像素（w×h）	用　　途
icon.png	6.75	48×48	游戏的图标
backmusicoff.png	6.57	220×50	控制游戏背景音乐开
dialog.9.png	4.0	150×80	读档对话框背景
dukea.png	37.1	480×320	欢迎界面图片 1
gang.png	2.83	20×20	表示时间的横杠图片
help.png	683	800×480	帮助界面
jifenbackground.png	688	800×480	游戏积分榜的背景图片
mainbackground.png	684	800×480	主界面的底层背景
musicbackground.png	661	800×480	音效设置背景图片
select.png	119	269×465	主界面的选项图片
yinxiaooff.png	6.29	220×50	控制游戏音效开
yinxiaoon.png	5.98	220×50	控制游戏音效关
gameover.png	35.8	800×480	游戏结束图片

（2）接下来为读者介绍的图片资源为游戏中所要见到的图片资源，系统将该部分图片资源放在项目文件下的 res/drawable-nodpi 文件夹下，如表 15-2 所示。

表 15-2　　游戏界面图片清单

图 片 名	大小（KB）	像素（w×h）	用　　途
back.png	738	800×480	游戏界面的底层背景
b01~b12.png	4.02（12 张）	40×60	怪物精灵 1 的各种形态图片
c 01~c12.png	6.02（12 张）	40×60	怪物精灵 2 的各种形态图片
d01~d12.png	6.04（12 张）	40×60	怪物精灵 3 的各种形态图片

续表

图 片 名	大小（KB）	像素（w×h）	用　　途
d0~d9.png	1.20（10 张）	16×23	数字图片
biao1.png	10.2	70×70	箭塔 1 图片
biao2.png	9.52	70×70	箭塔 2 图片
blood.png	3.63	45×40	游戏界面中用于表示当前血量的图片
caidan.png	10.2	80×80	游戏界面中的菜单图片
caidanbig.png	147	280×350	弹出暂停菜单图片
cando.png	5.46	130×130	可以操作箭塔图层图片
canlocate.png	4.7	130×130	箭塔可以建造的标识图片
dazhao.png	138	800×480	使用水晶的效果图片
flower01.png	5.30	40×40	花草图片
gonglu.png	2.09	40×40	公路图片
money.png	6.07	40×40	游戏界面中的记录当前钱数的图片
shidishu.png	4.11	40×40	用于标识杀怪物数量的图片
shuijing.png	4.33	40×40	游戏中的表示水晶的图片
shuijingcover.png	2.78	40×40	水晶技能使用时覆盖的图片
tree01.png	2.08	40×40	树桩图片
tree02.png	7.38	40×80	树木图片
home.png	20.9	75×106	大本营的图片
jianta.png	10.0	40×80	游戏箭塔 1 图片
jianta1.png	11.0	40×80	游戏箭塔 2 图片
tubiao.png	2.92	11×11	箭塔发射的炮弹图片

（3）本游戏用到的声音资源，将声音资源放在项目目录中的 res/raw 文件夹下，其详细情况如表 15-3 所示。

表 15-3　　声音清单

声音文件名	大小（KB）	格　　式	用　　途
background.mp3	6.03	mid	背景音乐
explode.ogg	5.68	ogg	炮弹击中精灵怪物的音效

15.3 游戏的架构

上一小节实现了游戏的策划和前期准备工作，本节将对该游戏的架构进行简单介绍，方便读者对游戏开发过程有一个深的认识。

15.3.1 各个类的简要介绍

为了让读者能够更好地理解各个类的作用，下面将其分成 4 部分进行介绍，而各个类的详细代码将在后面的章节中相继给出。

1. 公共类

- Activity 的实现类 TafangGameActivity

该类是通过扩展 Activity 得到的，是整个游戏的控制器，也是整个游戏的程序入口。

- 数据库类 `DBUtil`

该类包含系统用到的数据库查询、插入等数据库操作方法。

2. 界面相关类

- 欢迎界面类 WelcomeView

该类为软件启动后要出现的欢迎界面，通过改变图片的透明度实现图片的渐变，运行期间屏蔽任何触屏以及键盘操作。

- 主操作界面类 MainMenuSurfaceView

该类为本软件的主界面，可以进行游戏音效的设定、跳转帮助界面、查看游戏积分榜、运行新游戏以及读取游戏存档继续游戏等操作。

- 排行榜界面类 HighJifenSurfaceView

该类为游戏积分排行榜的实现类，该类数据库查询中应用了分页技术，通过向上或者向下滑动屏幕实现积分榜信息的分页显示。

- 帮助界面类 HelpView

该类主要是静态地向读者介绍精灵塔防的玩法。

- 音效设置界面类 MusicSurfaceView

该类主要负责游戏中的背景音乐和炮塔的射击音效的开启和关闭。

- 游戏结束界面类 GameOverView

该类为游戏失败后出现的结束界面，该类和 WelcomeView 类相似，也是通过改变图片的透明度来实现图片的渐变。

- 游戏界面类 GameView

该类为游戏程序中最主要的类，主要负责游戏画面的绘制。

3. 辅助线程类

- GameViewDrawThread 类

继承自 Thread 类，重绘 GameView 的 SurfaceView 对象。

- IfGameOverThread 类

继承自 Thread 类，用来判断游戏何时结束。

- ShellNumThread 类和 ShellThread 类

二者均继承自 Thread 类，用来控制炮弹的产生和炮弹的发射路线。

- TargetNumThread 类和 TargetThread 类

二者均继承自 Thread 类，分别用来控制怪物的产生和怪物的行走。

- SelectThread 类

继承自 Thread 类，用来控制主界面中下拉选项组下拉的线程。

- TimeThread 类和 ShuiJingThread 类

继承自 Thread 类，用来控制游戏界面中水晶物品操作的线程。

4. 实体对象类

- Shell 类和 Target 类

Shell 类是为描述炮塔射击炮弹开发的对象类，Target 为描述怪物精灵开发的对象类。

- SingleJianta 类和 SingleGoJianta 类

前者为描述游戏界面中箭塔开发的对象类，后者为描述游戏中拖动箭塔过程中的箭塔类。

15.3.2　游戏框架简介

上一小节已经对该游戏中所用到的类进行了简单介绍，可能读者还没有理解游戏的架构以及游戏的运行过程，本节将从游戏的整体架构上进行介绍，使读者对本游戏有更好地理解，其框架如图 15-17 所示。

▲图 15-17　游戏框架图

接下来按照程序运行的顺序介绍各个类的作用以及整体的运行框架，其详细步骤如下。

（1）启动游戏，首先创建的是 TafangGameActivity，在 TafangGameActivity 中首先将用户界面显示成欢迎界面 WelcomeView。

（2）欢迎界面播放完毕，游戏将切换到主菜单界面 MainMenuSurfaceView。

（3）在主菜单界面下，界面中有继续游戏、新游戏、积分榜、音效设置和帮助选择按钮，玩家单击新游戏进入游戏界面 GameView；玩家单击积分榜按钮就进入游戏的积分榜界面 HighJifenSurfaceView；当玩家单击帮助按钮，系统进入帮助界面 HelpView；如果单击继续游戏界

面，弹出游戏读档对话框，选择相应的游戏档就进入游戏界面 GameView，接着上次玩的游戏继续。

（4）当进入游戏界面时，玩家可以通过建造箭塔用以阻挡怪物精灵的侵袭，当主城血量为 0 时，塔防游戏失败，系统会跳转到游戏失败界面 GameOverView。

15.4 主控制类 TafangGameActivity 和数据库类 DBUtil

从本节开始将进入游戏的开发过程，首先介绍的是游戏的控制器 TafangGameACtivity 类，该类主要作用是，在适当的时间初始化相应的界面，并根据其他界面发送回来的消息切换到用户需要的界面。

15.4.1 主控制类——TafangGameActivity 主要框架

（1）下面先简单介绍 TafangGameActivity 的框架代码，使读者对代码有一个整体了解，其中省略部分模块代码，将在后面再进行详细讲解，其框架代码如下。

代码位置：见随书光盘中源代码/第 11 章/MapMaset/src/com/bn/map 目录下的 TafangGameActivity.java。

```
1     package com.bn.tag;
2     import java.util.HashMap;
3     //此处省略了部分类的引入代码，读者可以自行查阅随书光盘的源代码
4     enum WhichView                                          //
5          {WELCOME_VIEW,GAME_VIEW,MUSIC_VIEW,HELP_VIEW,JIFEN_VIEW,MAINMENU_VIEW};
6     public class TafangGameActivity extends Activity {
7          GameView gameView;                                 //GameView 的引用
8          WelcomeView welcomeview;                           //欢迎界面
9          MainMenuSurfaceView MainMenuView;                  //主界面
10         HighJifenSurfaceView ScoreView;                    //积分界面
11         MusicSurfaceView musicView;                        //音乐设置界面
12         GameOverView gameoverView;                         //游戏结束界面
13         HelpView helpview;                                 //帮助界面
14         static SQLiteDatabase sld;                         //数据源
15         Dialog Inputdialog;                                //游戏存档对话框
16         Dialog GoonGamedialog;                             //继续游戏对话框
17         String name;                                       //继续游戏对话框中选择的 name
18         int screenWidth;                                   //手机屏幕的宽度
19         int screenHeight;                                  //手机屏幕高度
20         List<String> nearlist=new Vector<String>();
21         private boolean backGroundMusicOn=true;            //背景音乐是否开启的标志
22         private boolean soundOn=true;                      //音效是否开启的标志
23         boolean cundang_Flag=false;
24         Vibrator mVibrator;                                //声明振动器
25         boolean shakeflag=true;                            //是否振动
26         WhichView curr;
27         //此处省略了 Handler 具体处理的代码，将在后面给出
28       @Override
29       public void onCreate(Bundle savedInstanceState) {
30              super.onCreate(savedInstanceState);          //调用父类
31              DisplayMetrics dm=new DisplayMetrics();      //新建 DisplayMetrics 对象
32              getWindowManager().getDefaultDisplay().getMetrics(dm);
                                                             //动态得到手机屏幕的尺寸
33              screenWidth=dm.widthPixels;                  //屏幕的宽度
34              screenHeight=dm.heightPixels;                //屏幕的高度
35              requestWindowFeature(Window.FEATURE_NO_TITLE); //全屏
36              getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN ,
```

```
37                          WindowManager.LayoutParams.FLAG_FULLSCREEN);
38
    this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                                                        //设置为横屏
39              collisionShake();                       //初始化
40              gotoWelcomeView();                      //开机动画,首先进入WelcomeView
41        }
42          private void gotoWelcomeView(){             //欢迎界面
43              if(welcomeview==null){
44                  welcomeview = new WelcomeView(this); //新建WelcomeView
45              }
46              this.setContentView(welcomeview);     //设置当前WelcomeView显示于界面
47              curr=WhichView.WELCOME_VIEW;
48        }
49          private void gotoGameOverView(){          //游戏结束界面
50              if(gameoverView==null){
51                  gameoverView=new GameOverView(this); //新建GameOverView
52              }
53              this.setContentView(gameoverView);
54        }
55          private void gotoGameView(){                //游戏界面
56              gameView=new GameView(this);            //新建GameView
57              this.setContentView(gameView);          //设置当前界面显示
58              curr=WhichView.GAME_VIEW;
59        }
60        private void gotoMainMenuView(){              //主界面
61              MainMenuView=new MainMenuSurfaceView(this);//新建MainMenuSurfaceView
62              this.setContentView(MainMenuView);      //设置当前界面显示
63              curr=WhichView.MAINMENU_VIEW;
64        }
65        public void gotoHelpView(){                   //帮助界面
66              if(helpview==null){
67                  helpview=new HelpView(this);        //新建HelpView
68              }
69              this.setContentView(helpview);          //设置当前界面显示
70              curr=WhichView.HELP_VIEW;
71        }
72          public void gotoJifenView(){                //积分界面
73              if(ScoreView==null){
74                  ScoreView=new HighJifenSurfaceView(this); //新建HighJifenSurfaceView
75              }
76              this.setContentView(ScoreView);           //设置当前界面显示
77              curr=WhichView.JIFEN_VIEW;
78          }
79          public void gotoMusicView(){                //音乐设置界面
80              if(musicView==null){
81                  musicView=new MusicSurfaceView(this);//新建MusicSurfaceView
82              }
83              this.setContentView(musicView);         //设置当前界面显示
84              curr=WhichView.MUSIC_VIEW;
85          }
86        public void gototachu(){                      //弹出对话框
87              showDialog(CUN_DANG_DIALOG_ID);         //显示对话框
88        }
89        public void gotoTaChuGame(){                  //继续游戏弹出对话
90              nearlist=DBUtil.getcundang();           //数据库查询语句
91              showDialog(GO_ONGAME_DIALOG_ID);        //显示对话框
92        }
93        public void sendMessage(int what){            //传播信息
94              Message msg1 = myHandler.obtainMessage(what);
95              myHandler.sendMessage(msg1);            //发送消息
96        }
```

```
97        public boolean isBackGroundMusicOn() {                //获得声音播放标志位的函数
98             return backGroundMusicOn;
99        }
100       public void setBackGroundMusicOn(boolean backGroundMusicOn) {     //背景音乐
101            this.backGroundMusicOn = backGroundMusicOn;     //背景音乐开启标志位
102       }
103       public boolean isSoundOn() {                          //音效
104            return soundOn;                                  //返回
105       }
106       public void setSoundOn(boolean soundOn) {             //设置音效标志位
107            this.soundOn = soundOn;
108       }
109     public void collisionShake(){                           //手机振动
110            mVibrator=(Vibrator)getApplication().getSystemService
111            (Service.VIBRATOR_SERVICE);
112     }
113       //设置振动信息
114       public void shake(){                                  //振动方法
115            if(shakeflag){
116                 mVibrator.vibrate( new long[]{0,80},-1);  //振动信息
117            }
118       }
119       @Override
120       public boolean onKeyDown(int keyCode,KeyEvent e) {   //重写手机返回键的事件监听
121            if(keyCode==4){                                  //调制上一个界面的键
122                 if(curr==WhichView.HELP_VIEW){
123                      gotoMainMenuView();                    //返回主界面
124                      return true;                           //返回
125                 }
126                 if(curr==WhichView.JIFEN_VIEW||curr==WhichView.MUSIC_VIEW){
127                      gotoMainMenuView();                    //返回主界面
128                      return true;
129                 }
130            }
131            return false;
132       }
133       //省略部分数据库查询、插入模块，后面会对此进行详细介绍
134       //省略对话框相关知识模块，后面将会对此进行详细讲解
135  }
```

- 第 4～5 行为声明一个枚举类型，用来标识当前玩家所在的界面，当按下手机返回按钮时，系统会根据当前所在界面的枚举标识，跳转到相应的前一界面。
- 第 7～26 行为成员变量的声明。
- 第 31～34 行新建 DisplayMetrics 对象，动态获取当前屏幕的分辨率，dm.widthPixels 为系统得到屏幕的宽度，dm.heightPixels 为系统得到的高度。
- 第 35～38 行为对手机屏幕进行设置，这里设置为手机屏幕全屏显示，并且强制为横屏。
- 第 42～85 行为切换屏幕到欢迎界面、主界面、游戏界面、帮助界面、音效设置界面、积分界面以及游戏结束界面的方法。
- 第 86～92 行为游戏存档对话框和继续存档游戏对话框的方法代码。
- 第 93～96 行传播消息发送消息的方法。
- 第 97～108 行分别为背景音乐开启标志位的设定以及游戏音效标志位的设定方法。
- 第 109～118 行为通过方法设定手机振动信息。本游戏手机振动主要用于怪物进入防御方营

地时，系统会触发手机振动，表明已有怪物侵袭到防御方营地中。

- 第 119～132 行为重写返回键的事件监听，根据当前的界面，单击返回键会跳转运行界面的函数。

（2）接下来将介绍的是 TafangGameActivity 类中的 Handler 消息处理器，其方法代码如下。

```
1    Handler myHandler = new Handler(){                    //处理各个 SurfaceView 发送的消息
2         @Override
3         public void handleMessage(Message msg) {         //根据出入不同的 ID 选择切换到不同界面
4         switch(msg.what){
5              case 0:
6                   gotoGameView();                         //切换到游戏界面
7                   break;
8              case 1:
9                   gotoMainMenuView();                     //切换到主界面
10                  break;
11             case 2:
12                  gotoHelpView();                         //切换到帮助界面
13                  break;
14             case 3:
15                  gotoJifenView();                        //切换到积分界面
16                  break;
17             case 4:
18                  gotoMusicView();                        //切换到音乐设置界面
19                  break;
20             case 5:
21                  gotoGameOverView();                     //切换到游戏结束界面
22                  break;
23   }}};
```

说明　本段代码是 Handler 根据传入不同的消息 ID 判断系统要跳转到的界面，例如，当 Handler 消息处理器收到消息 0 时，就会调用 gotoGameView 函数，跳转到游戏界面，其他操作类似。

15.4.2　主控制类——TafangGameActivity 中部分数据库的实现

TafangGameActivity 类中的数据库主要是针对游戏的积分记录而开发设计的，系统会将玩家的游戏结果通过数据库查询语句插入到数据库中，然后玩家可以通过软件的积分榜查看游戏的积分列表，当然，这里只介绍这部分的数据库实现，积分界面的设计将在后面详细介绍。

（1）创建数据库的代码如下。

```
1    public static void openOrCreateDatabase(){                    //数据库创建函数
2         try{                                                     //异常处理
3              sld=SQLiteDatabase.openDatabase                     //声明数据源
4              (
5                   "/data/data/com.bn.tag/ttdb",                  //数据库所在路径
6                   null,                                          //CursorFactory
7                   SQLiteDatabase.OPEN_READWRITE|SQLiteDatabase.CREATE_IF_NECESSARY
8              );
```

```
9               String sql="create table if not exists highScore" +//创建表的数据库语句
10                      "( " +
11                      "score integer," +                          //得分
12                      "date varchar(20)" +                        //时间
13                      ");";
14              sld.execSQL(sql);                                   //创建表
15          }
16          catch(Exception e){                                     //打印异常信息
17              e.printStackTrace();
18      }}
```

- 第 3～8 行为创建 SQLite 数据库的方法。
- 第 9～14 行为在创建的数据库中建表 highScore，记录游戏中的得分记录。

（2）在数据库中插入数据的方法 insert()，其代码如下。

```
1     public void insert(int score,String date){                //数据库插入记录方法
2         try{
3             openOrCreateDatabase();                           //打开数据源
4             String sql="insert into highScore values("+score+",'"+date+"');";
                                                                //插入记录的 SQL 语句
5         sld.execSQL(sql);                                     //插入记录
6         sld.close();                                          //关闭数据源
7         }
8         catch(Exception e){                                   //打印异常信息
9             e.printStackTrace();
10    }}
```

说明 数据库查询、插入方法完成以后要记住关闭数据源，否则会影响数据库查询性能。

（3）数据库查询方法 query()，方法代码如下。

```
1     public String query(int posFrom,int length){              //开始的位置，要查寻的记录条数
2         StringBuilder sb=new StringBuilder();                 //要返回的结果
3         Cursor cur=null;                                      //声明游标
4         openOrCreateDatabase();                               //打开数据源
5         String sql="select score,date from highScore order by score desc;";
                                                                //查询的 SQL 语句
6         cur=sld.rawQuery(sql, null);
7         try{
8             cur.moveToPosition(posFrom);                      //将游标移动到指定的开始位置
9             int count=0;                                      //当前查询记录条数
10            while(cur.moveToNext()&&count<length){
11                int score=cur.getInt(0);                      //积分
12                String date=cur.getString(1);                 //时间
13                sb.append(score);                             //添加积分
14                sb.append("/");                               //添加斜杠
15                sb.append(date);
```

```
16                  sb.append("/");                          //将记录用"/"分隔开
17                  count++;                                 //记录当前信息条数
18              }
19          }
20          catch(Exception e){                              //异常信息打印
21              e.printStackTrace();
22          }
23          finally{
24              cur.close();                                 //关闭游标
25              closeDatabase();                             //关闭数据源
26          }
27          return sb.toString();
28      }
```

- 第 2～5 行为一些变量的声明以及打开数据库声明游标。
- 第 6～18 行为根据查询的位置以及一组查询结果数量的限制对积分表进行查询。
- 第 23～26 行为数据库查询语句查询完毕，关闭数据游标，关闭数据源。

（4）数据库中记录数量查询方法 getRowcount()，实现代码如下。

```
1     public int getRowCount(){                                  //记录数量查询方法
2         int result=0;                                          //声明结果变量
3         Cursor cur=null;                                       //声明游标
4         openOrCreateDatabase();
5         try{
6             String sql="select count(score) from highScore;";//得到记录数量的 SQL 语句
7             cur=sld.rawQuery(sql, null);                       //游标
8             if(cur.moveToNext()){
9                 result=cur.getInt(0);                          //得到记录数量
10            }
11        }
12        catch(Exception e){
13            e.printStackTrace();                           //打印异常信息
14        }
15        finally{
16            cur.close();                                   //关闭游标
17            closeDatabase();                               //关闭数据库
18        }
19        return result;                                     //返回结果
20    }
```

- 第 2～3 行为声明结果变量，声明游标，打开数据库。
- 第 5～11 行为通过数据库查询得到当前数据库中记录的数量。
- 第 15～18 行为关闭游标、关闭数据库操作。

15.4.3　主控制类——TafangGameActivity 中对话框的开发

游戏中多次用到了对话框的开发知识，有不同类型的，不同用法的，包括提示玩家存档的对话框以及继续游戏加载文档的对话框，在这里将分步骤详细介绍这两方面的知识。

（1）判断并创建不同类型的对话框函数 onCreateDialog()，函数代码如下。

```
1    public Dialog onCreateDialog(int id){                    //创建对话框
2         Dialog result=null;
3         switch(id){
4              case CUN_DANG_DIALOG_ID:                       //存档输入对话框
5                   Inputdialog=new MyDialogue(this);  //创建对话框
6                   result=Inputdialog;                       //返回结果
7                   break;
8              case GO_ONGAME_DIALOG_ID:                      //游戏读档对话框
9                   AlertDialog.Builder b2=new AlertDialog.Builder(this);
                                                              //通过 AlertDialog.Builder 方法创建
10                  b2.setItems(                              //类型
11                       null,
12                       null
13                  );
14                  GoonGamedialog=b2.create();               //创建 Dialog（对话框）
15                  result=GoonGamedialog;                    //返回结果
16                  break;
17        }
18        return result;
19   }
```

- 第 4～7 行为通过判断传入的 ID 创建游戏存档的对话框，其中用到了类 MyDialogue，该类的具体实现代码会在接下来的步骤里详细介绍。
- 第 8～16 行为通过判断传入的 ID 创建游戏读档的对话框。

说明 通过重写这个方法，就可以通过传入不同的 ID 创建相应类型的对话框，当然要想完成对话框的最后创建还需重写 onPrepareDialog 函数，具体实现代码将在后面给出。

（2）MyDialogue 类的实现，具体代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGame/src/com/bn/tag 目录下的 MyDialogue.java。

```
1    package com.bn.tag;                                      //引用的包名
2    import android.app.Dialog;
3    //此处省略了部分类的引入代码，读者可以自行查阅随书光盘的源代码
4    public class MyDialogue extends Dialog{                  //继承 Dialog
5         public MyDialogue(TafangGameActivity tafangGameActivity) {
6              super(tafangGameActivity,R.style.FullHeightDialog);//调用父类
7         }
8         @Override
9         public void onCreate (Bundle savedInstanceState){
10             this.setContentView(R.layout.dialog_name_input);  //设置对话框样式
11        }
12        @Override
13        public String toString(){                            //重写 toString 方法
14             return "MyDialog";
15   }}
```

- 第 5～7 行为 MyDialogue 构造器，调用父类。
- 第 8～11 行为重写 onCreate 方法，设置对话框样式，相应的 XML 布局文件见随书光盘中源代码/第 15 章/TafangGame/res/layout 目录下的 dialog_name_input.xml。
- 第 12～15 行为重写 toString 方法。

说明　该类定义了游戏存档对话框的样式，这里只是讲解了一种对话框开发的方法，读者以后进行相应对话框的开发完全可以借鉴、重写。

（3）onPrepareDialog 函数的开发，具体实现代码如下。

```
1    public void onPrepareDialog(int id, final Dialog dialog){
2         switch(id){
3              case CUN_DANG_DIALOG_ID:                          //存档名输入对话框
4                   Button bjhmcok=(Button)Inputdialog.findViewById(R.id.bjhmcOk);
                                                                  //确定按钮
5                   Button bjhmccancel=(Button)Inputdialog.findViewById(R.id.bjhmcCancle);
                                                                  //取消按钮
6                   bjhmcok.setOnClickListener(                   //设置监听事件
7                   new OnClickListener(){
8                   @Override
9                   public void onClick(View v){                  //重写 onClick 方法
10                       EditText et=(EditText)Inputdialog.findViewById(R.id.etname);
                                                                  //得到文本框
11                       String name=et.getText().toString();//得到文本框中输入的内容
12                       String currentDate=DateUtil.getCurrentDate();     //得到时间
13                       String sql1="insert into save values('"+name+"','"+current-
                         Date+"')";                                       //SQL 语句
14                       DBUtil.updatetable(sql1);                 //执行插入语句
15                       for(int i=0;i<gameView.jiantaList.size();i++){
                                                                  //遍历箭塔列表
16                            String sql="insert into jiant values ('"+name+"',
                                                                  //得到当前箭塔的定位信息
17                            '"+gameView.jiantaList.get(i).clo+"','"+gameView.
                              jiantaList.get(i).row+
18                            "','"+gameView.jiantaList.get(i).state+"')";
19                            DBUtil.updatetable(sql);    //执行插入
20                       }
21                       for(int i=0;i<gameView.alTarget1.size();i++){
                                                                  //遍历游戏中的怪物精灵列表
22                            String sql="insert into guaiw values ('"+name+"',
                                                                  //得到定位精灵的信息
23                            '"+gameView.alTarget1.get(i).ballx+"','"+gameView.
                              alTarget1.get(i).bally
24                            +"','"+gameView.alTarget1.get(i).bloodsumNO+"','"+
25                            gameView.alTarget1.get(i).direction*180/Math.PI+"',
26                            '"+gameView.alTarget1.get(i).ii+"')";
27                            DBUtil.updatetable(sql);    //执行数据库插入语句操作
28                       }
29                       String sql="insert into nochange values ('"+name+"',
                                                                  //插入游戏中数字类金钱等
30                       '"+gameView.doller+"','"+gameView.bloodNUM+"','"+
31                       gameView.shaNUM+"','"+gameView.shuijingMiddleNum+"')";
32                       DBUtil.updatetable(sql);          //执行数据库插入操作
```

```
33                      Inputdialog.cancel();            //取消对话框
34                  } });
35                  bjhmccancel.setOnClickListener(      //为取消按钮设定监听事件
36                      new OnClickListener(){           //内部类
37                          @Override
38                          public void onClick(View v){    //重写 onClick 方法
39                              Inputdialog.cancel();       //取消对话框，不做任何操作
40                  } });
41                  break;
42              case GO_ONGAME_DIALOG_ID:                   //继续游戏读档对话框
43                  LinearLayout ll=new LinearLayout(TafangGameActivity.this);
44                  ll.setOrientation(LinearLayout.VERTICAL); //设置朝向
45                  ll.setGravity(Gravity.CENTER_HORIZONTAL); //这时对齐
46                  ll.setBackgroundResource(R.drawable.dialog);  //设定背景
47                  LinearLayout lln=new LinearLayout(TafangGameActivity.this);
48                  lln.setOrientation(LinearLayout.HORIZONTAL);  //设置朝向
49                  lln.setGravity(Gravity.CENTER);               //对齐方式
50                  TextView tvTitle=new TextView(TafangGameActivity.this);
51                  tvTitle.setText("请选择游戏相应的存档");          //设置标题
52                  tvTitle.setTextSize(20);                      //设置字体大小
53                  tvTitle.setTextColor(Color.WHITE);            //设置字体颜色
54                  lln.addView(tvTitle);
55                  ll.addView(lln);                              //添加控件 View
56                  ListView lv=new ListView(this);               //新建 ListView
57                  lv.setChoiceMode(ListView.CHOICE_MODE_SINGLE);//设置为单选模式
58                  BaseAdapter ba=new BaseAdapter(){             //新建适配器
59                  @Override
60                  public int getCount() {                       //返回数量
61                      return nearlist.size();                   //总共几个选项
62                  }
63                  @Override
64                  public Object getItem(int arg0) { return null; }
65                  @Override
66                  public long getItemId(int arg0) { return 0; }
67                  @Override
68                  public View getView(int arg0, View arg1, ViewGroup arg2) {
69                      TextView tv=new TextView(TafangGameActivity.this);
70                      tv.setGravity(Gravity.CENTER);  //对齐方式
71                      tv.setText(nearlist.get(arg0)); //设置内容
72                      tv.setTextSize(20);             //设置字体大小
73                      tv.setTextColor(Color.WHITE);   //设置字体颜色
74                      tv.setPadding(0,0,0,0);         //设置四周留白
75                      return tv;
76                  }
77              };
78              lv.setAdapter(ba);                          //为 ListView 设置内容适配器
79              lv.setOnItemClickListener(
80                  new OnItemClickListener(){              //事件监听
81                      @Override
```

```
82                          public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
83                                          long arg3) {      //重写选项被单击事件的处理方法
84                               name=nearlist.get(arg2);     //得到读档中的名称
85                               cundang_Flag=true;           //设定读档标志位游戏是继续游戏
86                               gotoGameView();              //切换到游戏界面
87                               GoonGamedialog.cancel();     //取消当前对话框
88                }});
89                ll.addView(lv);                             //添加控件 View
90                dialog.setContentView(ll);
91     }}
```

- 第 4～5 行为得到存档对话框的确定和取消操作按钮。
- 第 10～14 行根据在存档对话框中输入的名称进行名称事件存档。
- 第 15～20 行遍历箭塔列表得到当前箭塔的位置坐标、箭塔的攻击范围等记录
- 第 21～28 行为遍历怪物列表获得当前怪物精灵的种类、位置坐标、当前种类精灵的血量以及总血量等，用于继续游戏中怪物的加载。
- 第 29～34 行为获得当前游戏进度下的金钱数、当前家的血量、杀敌数以及水晶数量，用于继续游戏的金钱、血量、杀敌以及水晶物品的加载。
- 第 35～40 行为对话框中取消按钮的监听设置。
- 第 43～57 行为继续游戏加载文档对话框的界面设置。
- 第 58～77 行将数据通过 listView 加载进文档进度选择对话框内。
- 第 78～90 行选择不同 ListView 中的元素，加载相应的游戏进度。

15.4.4　数据库类 DBUtil 的开发

上面介绍的类 TafangGameActivity 中涉及了积分榜的数据库开发，本节将详细讲解关于游戏存档和读档的相关数据库的开发，相关的执行操作将在后续章节进行详细讲解。类 DBUtil 代码位于随书光盘中源代码/第 15 章/TafangGame/src/com/bn/tag 目录下的 DBUtil.java

（1）存档、读档游戏数据库的建立。方法代码如下。

```
1     public static void createOrOpenDatabase() throws Exception{
2          sld=SQLiteDatabase.openDatabase
3          (
4               "/data/data/com.bn.tag/tffadb",                    //数据库所在路径
5               null,                                              //CursorFactory
6               SQLiteDatabase.OPEN_READWRITE|SQLiteDatabase.CREATE_IF_NECESSARY
7          );
8          String sql1="create table if not exists save(savename varchar2(20),savetime
           varchar2(20))";
9          sld.execSQL(sql1);                                     //创建表 save
10         String sql2="create table if not exists nochange(savename varchar2(20),
11         money Integer(4),blood Integer(4),shdi Integer(4),shuijing Integer(4))";
12         sld.execSQL(sql2);                                     //创建表记录数量类
13         String sql3="create table if not exists guaiw(savename varchar2(20),
14         gw_x INTEGER(4),gw_y INTEGER(4),gw_state INTEGER(2),direction INTEGER(3),ii
           INTEGER(2))";
15         sld.execSQL(sql3);                                     //创建怪物表
16         String sql4="create table if not exists jiant(savename varchar2(20),
17         jt_x INTEGER(4),jt_y INTEGER(4),gt_state INTEGER(2))";
```

```
18          sld.execSQL(sql4);                                    //创建箭塔表
19    }
```

- 第 8～9 行为创建 save 表，用于记录游戏存档的名称和时间。
- 第 10～12 行为创建 nochange，用于记录当前游戏下的金钱数、血量、杀敌数以及水晶物品的数量。
- 第 13～15 行为创建表 guaiw，用于记录当前怪物精灵的位置、血量以及怪物的种类。
- 第 16～118 行为创建表 jiant，用于记录当前箭塔的位置、射程等。

（2）获得当前游戏的金钱数等的查询方法 searchnochange()，实现代码如下。

```
1     public static int[] searchnochange(String savename){
2          int[] result=new int[4];                                //结果数组
3          try{
4               createOrOpenDatabase();                            //打开数据源
5               String sql="select nochange.money,nochange.blood,  //查询 SQL 语句
6               nochange.shdi,nochange.shuijing from nochange where
7               nochange.savename='"+savename+"'";
8               Cursor cur=sld.rawQuery(sql, new String[]{});      //结果游标
9               if(cur.moveToNext()){
10                    result[0]=cur.getInt(0);                     //金钱物品
11                    result[1]=cur.getInt(1);                     //生命
12                    result[2]=cur.getInt(2);                     //杀敌数
13                    result[3]=cur.getInt(3);                     //水晶物品数
14              }
15              cur.close();                                       //关闭游标
16              closeDatabase();                                   //关闭数据库
17         }
18         catch(Exception e){
19         e.printStackTrace();                                    //异常信息打印
20         }
21         return result;
22    }
```

说明

此方法用于获得数据库中的某一游戏存档中的金钱数、血量、杀敌数以及水晶物品数，当游戏加载过程中，会调用这个方法。

（3）怪物精灵信息表、箭塔信息表以及读档查询信息表的查询和上面的方法类似，这里就不再赘述了，读者可参见随书光盘相应版块的代码。

11.5 界面相关类

前面的章节介绍了主控制类 TafangActivity 和公共类中的 DBUtil，本节将对游戏软件的界面进行分别介绍，讲解的界面主要是：欢迎界面 WelcomeView、主界面 MainMenuSurfaceView、积分榜界面 HighJifenSurfaceView、帮助界面 HelpView、音效设置界面 MusicSurfaceView 以及游戏结束界面 GameOverView，下面将分别介绍这几个界面。

15.5.1 欢迎界面 WelcomeView 类的介绍

本系统的欢迎界面主要是通过对两张图片的可见度进行设置，用以产生一种动态的效果，下面

分步讲解这方面内容。

（1）首先介绍的是欢迎界面的框架，其框架代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 WelcomeView.java。

```
1    package com.bn.tag;
2    import android.graphics.Bitmap;
3    //此处省略了部分类的引入代码，读者可以自行查阅随书光盘的源代码
4    public class WelcomeView extends SurfaceView
5    implements SurfaceHolder.Callback{                          //实现生命周期回调接口
6         TafangGameActivity activity;
7         Paint paint;                                           //画笔
8         int currentAlpha=0;                                    //当前的不透明值
9         float screenWidth=Constant.SCREEN_HEIGHT;              //屏幕宽度
10        float screenHeight=Constant.SCREEN_WIDTH;              //屏幕高度
11        int sleepSpan=50;                                      //动画的时延 ms
12        Bitmap[] logos=new Bitmap[2];                          //Logo 图片数组
13        Bitmap currentLogo;                                    //当前 Logo 图片引用
14        int currentX;
15        int currentY;
16        public WelcomeView(TafangGameActivity activity) {
17             super(activity);
18             this.activity = activity;
19             this.getHolder().addCallback(this);               //设置生命周期回调接口的实现者
20             paint = new Paint();                              //创建画笔
21             paint.setAntiAlias(true);                         //打开抗锯齿
22             logos[0]=BitmapFactory.decodeResource(activity.getResources(),
               R.drawable.dukea);
23             logos[1]=BitmapFactory.decodeResource(activity.getResources(),
               R.drawable.dukeb);
24        }
25        public void onDraw(Canvas canvas){
26             paint.setColor(Color.BLACK);                      //设置画笔颜色
27             paint.setAlpha(255);                              //设置可见度
28             canvas.drawRect(0, 0, screenWidth, screenHeight, paint);
29             if(currentLogo==null)return;
30             paint.setAlpha(currentAlpha);                     //设置可见度
31             canvas.drawBitmap(currentLogo, currentX, currentY, paint);
32        }
33        public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {}
34        public void surfaceCreated(SurfaceHolder holder) {        //创建时被调用
35             //省略了改变图片 Alpha 的方法，后面将详细讲解
36        }
37        public void surfaceDestroyed(SurfaceHolder arg0) { }      //销毁时被调用
38   }
```

- 第 6～15 行为声明了一些需要用到的成员变量。
- 第 18～23 行为构造器，创建了画笔，加载了两张用到的图片。
- 第 25～32 行为绘制函数，设置画笔颜色同时设置可见度，画当前图片。
- 第 34～36 行，此函数为在界面创建的时候被调用，函数启用一个线程，通过改变图片的可见度来实现图片的渐变效果。

（2）接下来将介绍的是在欢迎界面类中的 surfaceCreated 方法中省略的部分代码，其详细代码如下。

```
1    new Thread(){
2         public void run(){                                        //重写 run 方法
```

```
3          for(Bitmap bm:logos){                                          //遍历当前的图片数组
4               currentLogo=bm;                                           //当前图片
5               currentX=activity.screenWidth/2-bm.getWidth()/2;          //绘制的起始位置
6               currentY=activity.screenHeight/2-bm.getHeight()/2;        //绘制的起始位置
7               for(int i=255;i>-10;i=i-10){                              //改变图片的 Alpha
8                    currentAlpha=i;
9                    if(currentAlpha<0){                                  //当前可见度为负
10                        currentAlpha=0;                                 //设置当前可见度为 0
11                   }
12              SurfaceHolder myholder=WelcomeView.this.getHolder();
13              Canvas canvas = myholder.lockCanvas();                    //获取画布
14              try{
15                   synchronized(myholder){
16                        onDraw(canvas);                                 //绘制
17                   }
18              }
19              catch(Exception e){                                       //打印异常信息
20                   e.printStackTrace();
21              }
22              finally{
23                   if(canvas != null){                                  //如果不为空
24                        myholder.unlockCanvasAndPost(canvas);           //进行锁处理
25                   }
26              }
27              try{
28                   if(i==255){                                          //当前可见度为 255
29                        Thread.sleep(1000);                             //休眠
30                   }
31                   Thread.sleep(sleepSpan);
32              }
33              catch(Exception e){
34                   e.printStackTrace();                                 //打印异常信息
35              }}}
36              activity.sendMessage(1);
37         }
38    }.start();
```

说明 该线程通过强制性 for 循环，第一次循环，对欢迎界面图片 1 进行不透明度的变化，每变化一次画一次，就达到了渐变的效果。当第一张图片完全消失后，线程等待一小段时间，切换到第二幅图片，通过同样的处理，就构成了整个欢迎界面的动态效果。详细情况请见注释。绘制界面，首先锁定画布，然后调用 onDraw 函数进行界面绘制。

15.5.2 主界面 MainMenuSurfaceView 类的介绍

上面讲解了欢迎界面的实现过程，当欢迎界面动画完成以后，随即就进入到了游戏的主界面，下面将分步骤介绍主界面的开发过程。

（1）首先介绍 MainMenuSurfaceView 类的框架，后面将会对其进行详细讲解。框架代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGame/src/com/bn/tag 目录下的 MainMenuSurfaceView.java。

```
1    package com.bn.tag;                                                //引用的包名
2    import android.graphics.Bitmap;
3    //此处省略了部分类的引入代码，读者可以自行查阅随书光盘的源代码
4    public class MainMenuSurfaceView extends SurfaceView implements SurfaceHolder.
     Callback{
```

```
5          TafangGameActivity activity;                          //Activity 的引用
6          Paint paint;                                          //画笔引用
7          DrawThread drawThread;                                //绘制线程引用
8          SelectThread selectthread;                            //弹出菜单线程
9          Bitmap mainbackgroundpic;                             //背景图片
10         Bitmap selectpic;                                     //菜单图片
11         float ballX=0;
12         float ballY=-400;
13         public MainMenuSurfaceView(TafangGameActivity activity) {    //构造器
14              super(activity);                                        //调用父类
15              this.activity=activity;
16              this.requestFocus();
17              this.setFocusableInTouchMode(true);
18              getHolder().addCallback(this);                          //注册回调接口
19              initBitmap();
20         }
20         @Override
21         protected void onDraw(Canvas canvas) {                      //绘制函数
22              super.onDraw(canvas);                                   //调用父类
23              try{
24                   canvas.drawBitmap(mainbackgroundpic,0,0, null);    //绘制背景图
25                   canvas.drawBitmap(selectpic, ballX, ballY, null);  //绘制弹出菜单
26              }catch(Exception e){
27                   e.printStackTrace();
28              }
29         }
30         @Override
31         public boolean onTouchEvent(MotionEvent event) {         //触摸事件
32              //此处省略了部分的触摸引发的事件，后面将详细介绍
33         }
34         @Override
35         public void surfaceChanged(SurfaceHolder holder, int format, int width,
36                   int height) {
37         }
38         @Override
39         public void surfaceCreated(SurfaceHolder holder)
40         {
41              paint=new Paint();                                  //创建画笔
42              paint.setAntiAlias(true);                           //打开抗锯齿
43              selectthread=new SelectThread(this);                //新建线程
44              selectthread.start();                               //开启线程
45              createAllThreads();                                 //创建线程
46              startAllThreads();
47         }
48         @Override
49         public void surfaceDestroyed(SurfaceHolder holder) {     //销毁时触发
50                boolean retry = true;
51                stopAllThreads();
52         }
53         public void initBitmap(){                                //初始化图片资源
54              mainbackgroundpic=BitmapFactory.decodeResource(this.getResources(),
                                                                    //背景图片
55                                   R.drawable.mainbackground);
56              selectpic=BitmapFactory.decodeResource(this.getResources(),
                                                                    //菜单图片
57                                   R.drawable.select);
58         }
59         void createAllThreads(){
60              drawThread=new DrawThread(this);                    //创建绘制线程
61         }
62         void startAllThreads(){                                  //开启所有线程
63              drawThread.setFlag(true);
```

```
64                drawThread.start();
65            }
66            void stopAllThreads(){                                    //停止所有线程
67                drawThread.setFlag(false);
68            }
69            //此处省略了绘制画面的线程，后面将会给出详细讲解
70    }
```

- 第 5～12 行为相关成员变量的声明。
- 第 13～19 行本类的有参构造器。
- 第 21～29 行为绘制背景图片和菜单图片。
- 第 53～58 行为初始化加载主界面中用到的图片资源。
- 第 59～68 行为创建线程、开启线程以及开启工作线程的方法。

（2）onTouchEvent 触摸方法的代码实现，通过触摸不同的选项图标执行不同的事件响应，具体实现代码如下所示。

```
1     public boolean onTouchEvent(MotionEvent event) {          //触摸事件
2           int x = (int) event.getX();                         //当前触摸的位置坐标
3           int y = (int) event.getY();
4           switch(event.getAction()){
5           case MotionEvent.ACTION_DOWN:                       //按下事件
6           if(x>=GOON_X&&x<=GOON_X+MAIN_LENGTH&&y>=GOON_Y&&y<=GOON_Y+MAIN_WEIGHT)
7                 {
8                       activity.gotoTaChuGame();               //游戏加载界面
9                 }
10                if(x>=NEW_X&&x<=NEW_X+MAIN_LENGTH&&y>=NEW_Y&&y<=NEW_Y+MAIN_WEIGHT)
11                {
12                      activity.myHandler.sendEmptyMessage(0);   //游戏界面
13                }
14                if(x>=JIFEN_X&&x<=JIFEN_X+MAIN_LENGTH&&y>=JIFEN_Y&&y<=
15                          JIFEN_Y+MAIN_WEIGHT)
16                {
17                      activity.myHandler.sendEmptyMessage(3);   //积分查看界面
18                }
19                if(x>=YINXIAO_X&&x<=YINXIAO_X+MAIN_LENGTH&&y>=YINXIAO_Y&&
20                      y<=YINXIAO_Y+MAIN_WEIGHT)
21                {
22                      activity.myHandler.sendEmptyMessage(4);   //切换到音效设置界面
23                }
24                if(x>=HELP_X&&x<=HELP_X+MAIN_LENGTH&&y>=HELP_Y&&y<=HELP_Y+MAIN_WEIGHT)
25                {
26                      activity.myHandler.sendEmptyMessage(2);   //切换到帮助界面
27                }
28                if(x>=EXIT_X&&x<=EXIT_X+MAIN_LENGTH&&y>=EXIT_Y&&y<=EXIT_Y+MAIN_WEIGHT)
29                {
30                      System.exit(0);                           //退出游戏
31                }
32                break;
33          }
34                return true;
35    }
```

- 第 2～3 行为获得当前触摸位置的坐标。
- 第 5～33 行为触摸不同菜单中的选项进行不同界面的切换，主界面中包含：继续游戏、新游戏、积分榜查询、背景音效设置、帮助以及退出等几个选项。

（3）本软件的选项组是一个可以自上而下滑动的模块，进入主界面时会在界面的左上方滑下一个选项组，其实是包含主界面中所有按钮的图片，通过对其线程的控制，使其产生动画效果，接下来将介绍该线程 SelectThread 的开发过程，代码如下所示。

代码位置：见随书光盘中源代码/第 15 章/TafangGame/src/com/bn/tag 目录下的 SelectThread.java。

```
1    package com.bn.tag;                                          //声明包语句
2    import com.bn.tag.MainMenuSurfaceView;;                      //引入包名
3    public class SelectThread extends Thread{
4         MainMenuSurfaceView father;                             //MainMenuSurfaceView 的引用
5         boolean flag;                                           //线程启动标志位
6         int sleepSpan = 100;                                    //休眠时间
7         public SelectThread(MainMenuSurfaceView father){
8              this.father = father;
9         }
10        public void run(){                                      //重写 run 方法
11             while(true){
12                  father.ballY+=30;                              //弹出菜单向下移动
13                  if(father.ballY>=0){                           //都显示在界面中
14                       father.ballY=0;
15                       break;                                    //停止向下移动
16                  }
17                  try{
18                       Thread.sleep(sleepSpan);                  //线程休眠
19                  }
20                  catch(Exception e){
21                       e.printStackTrace();                      //打印异常信息
22        }}}
23        public void setFlag(boolean flag) {                     //设置线程开启标志位
24             this.flag = flag;
25   }}
```

- 第 4～9 行为相关变量和该类构造器的声明。
- 第 10～22 行动态改变弹出菜单的 y 坐标，不断绘制该弹出菜单，使玩家感觉到每次进入到主界面，弹出菜单都是从顶部滑落下，效果非常炫酷。
- 第 23～25 行设置线程开启标志位的方法。

（4）绘制画面线程 DrawThread，实现代码如下。

```
private class DrawThread extends Thread{                          //绘制画面线程
     private boolean flag = true;                                 //线程开启标志位
     private int sleepSpan = 100;                                 //睡眠时间
     MainMenuSurfaceView fatherView;                              //MainMenuSurfaceView 的引用
     SurfaceHolder surfaceHolder;
     public DrawThread(MainMenuSurfaceView fatherView){   //构造器
          this.fatherView = fatherView;
          this.surfaceHolder = fatherView.getHolder();
     }
     public void run(){                                           //重写 run 方法
          Canvas c;
          while (this.flag) {
          c = null;
          try {
               c = this.surfaceHolder.lockCanvas(null);
               synchronized (this.surfaceHolder) {
               fatherView.onDraw(c);                              //绘制
               }
          } finally {
```

```
20                if (c != null) {
21                     this.surfaceHolder.unlockCanvasAndPost(c);
22                }}
23                try{
24                     Thread.sleep(sleepSpan);                    //睡眠指定毫秒数
25                }
26                catch(Exception e){
27                     e.printStackTrace();                        //打印堆栈信息
28                }}}
29                public void setFlag(boolean flag) {             //设置线程开启标志位
30                     this.flag = flag;
31        }}
```

说明 该线程主要用于相应 SurfaceView 类的重绘工作。

15.5.3 音效设置界面 MusicSurfaceView 类的介绍

本游戏软件的音效包含游戏的背景音乐和游戏的炮弹音效，音效界面中可以通过控制背景音乐和游戏音效控制按钮控制背景音乐的播放和游戏音效的使用，界面的开发和前面的相似，整体上都是通过线程重绘相应的 SurfaceView 对象，音效设置界面的开发也遵循这个原则，所以相同的地方这里就不再介绍，读者可参见随书光盘中源代码/第 15 章/TafangGame/src/com/bn/tag 目录下的 MusicSurfaceView.java。

下面简单介绍音效设置界面中的按钮控制代码，实现代码如下所示。

```
1     public void onDraw(Canvas canvas){
2          paint.setColor(Color.BLACK);                           //设置画笔颜色
3          paint.setAlpha(255);                                   //设置可见度
4          canvas.drawBitmap(musicback, 0, 0, paint);             //绘制背景图片
5          if(backmusicFlag01)                                    //绘制背景音乐开关
6          {
7               canvas.drawBitmap(backMusicoff, pic_x, pic_y, paint);
8          }
9          else if(!backmusicFlag01)                              //绘制背景音乐开关
10         {
11              canvas.drawBitmap(backMusicon, pic_x, pic_y, paint);
12         }
13         if(yinxiaoFlag01)                                      //绘制游戏音效开关
14         {
15              float x1=pic_x;                                    //绘制起始点坐标
16              float y1=pic_y+MUSIC_HEIGHT+25;
17              canvas.drawBitmap(Yinxiaooff, x1, y1, paint);
18         }
19         else if(!yinxiaoFlag01)                                //绘制游戏音效开关
20         {
21              float x1=pic_x;                                    //绘制起始点坐标
22              float y1=pic_y+MUSIC_HEIGHT+25;
23              canvas.drawBitmap(Yinxiaoon, x1, y1, paint);
24         }    }
25         @Override
26         public boolean onTouchEvent(MotionEvent event) {       //触摸事件
27              int x = (int) event.getX();                        //当前触摸位置坐标
28              int y = (int) event.getY();
29              switch(event.getAction()){
30              case MotionEvent.ACTION_DOWN:                      //按下事件
31              if(x>pic_x&&x<pic_x+MUSIC_WEIGHT&&y>pic_y&&y<pic_y+MUSIC_HEIGHT){
                                                                   //背景音乐
```

```
32                  backmusicFlag02=!backmusicFlag02;                //背景音乐标志位置反
33                  activity.setBackGroundMusicOn(backmusicFlag02); //设置背景音乐标志位
34                  backmusicFlag01=!backmusicFlag01;                //绘制标志位
35              }
36              if(x>pic_x&&x<pic_x+MUSIC_WEIGHT&&y>pic_y+MUSIC_HEIGHT+25&& //游戏音效
37                  y<pic_y+MUSIC_HEIGHT+25+MUSIC_HEIGHT){
38                  yinxiaoFlag02=!yinxiaoFlag02;                    //游戏音效标志位置反
39                  activity.setSoundOn(yinxiaoFlag02);              //设置游戏音效
40                  yinxiaoFlag01=!yinxiaoFlag01;
41              }
42              break;
43          }
44              return true;                                         //返回
45      }
```

- 第 2～4 行为绘制界面的背景图片。
- 第 5～12 行根据当前的背景音乐是否开启标志位，进行按钮开启、关闭的相应绘制。
- 第 13～24 行根据当前的游戏音效是否开启标志位，进行按钮开启、关闭的相应绘制
- 第 31～35 行触摸背景音乐设置按钮，进行背景音乐开启、关闭操作。
- 第 36～41 行触摸游戏音效设置按钮，进行游戏音效的开启、关闭操作。

15.5.4　游戏结束界面 GameOverView 类和帮助界面 HelpView 类的介绍

当玩家在玩游戏时，如果玩家的生命值为 0，则游戏结束，系统会调用闪屏图片作为游戏结束的标志，闪屏过后，系统将自动进入游戏的主界面。这里用到的图片闪屏技术和欢迎界面的闪屏技术是一样的，这里就不再介绍游戏结束界面的代码开发，读者可参见随书光盘中源代码/第 15 章/TafangGame/src/com/bn/tag 目录下的 GameOverView.java。

帮助界面 HelpView 类的实现更加简单，界面中就是一张静态的背景图，所以就不再讲解，读者可参见随书光盘中源代码/第 15 章/TafangGame/src/com/bn/tag 目录下的 HelpView.java。

15.5.5　积分榜界面 HighJifenSurfaceView 类的介绍

该游戏的积分榜记录了玩家游戏的积分过程，玩家可以上下触摸滑动查看不同的积分记录，下面将分步骤详细讲解界面的开发过程。

（1）下面首先介绍积分榜界面的框架代码，具体模块实现代码后面将详细给出，框架代码如下。

```
package com.bn.tag;                                          //引入的包名
import android.graphics.Bitmap;
//此处省略了部分类的引入代码，读者可以自行查阅随书光盘的源代码
public class HighJifenSurfaceView extends SurfaceView implements SurfaceHolder.
Callback{
    TafangGameActivity activity;                             //Activity 的引用
    Paint paint;                                             //画笔引用
    DrawThread drawThread;                                   //绘制线程引用
    Bitmap bgBitmap;                                         //背景图片
    Bitmap[] numberBitmaps;                                  //数字图片
    Bitmap gangBitmap;                                       //符号"-"对应的图片
    int bmpx;                                                //文字位置
    String queryResultStr;                                   //查询数据库的结果
    String[] splitResultStrs;                                //将查询结果切分后的数组
    private int numberWidth;                                 //数字图片的宽度
    private int posFrom=-1;                                  //查询的开始位置
```

```
16          private int length=6;                                   //查询的最大记录个数
17          int downY=0;                                            //按下和抬起时的 y 坐标
18          int upY=0;
19          public HighJifenSurfaceView(TafangGameActivity activity) {   //构造器
20               super(activity);
21               this.activity=activity;
22               this.requestFocus();
23               this.setFocusableInTouchMode(true);
24               getHolder().addCallback(this);                         //注册回调接口
25          }
26          @Override
27          protected void onDraw(Canvas canvas) {                      //绘制函数
28               super.onDraw(canvas);
29               canvas.drawColor(Color.WHITE);                         //设置画笔颜色
30               canvas.drawBitmap(bgBitmap, 0, 0, paint);              //绘制背景图片
31               int x,y;
32               for(int i=0;i<splitResultStrs.length;i++){
33                    if(i%2==0){                                       //计算得分的位置

34                         x=400;
35                    }
36                    else{                                             //计算时间的位置
37                         x=230;
38                    }
39                    y=20+40+20+(numberBitmaps[0].getHeight()+10)*(i/2+1)+10;
40                    drawDateBitmap(splitResultStrs[i],x,y,canvas,paint);//绘制
41          }}
42               //此处省略画日期图片的方法，下面步骤将详细给出
43          @Override
44          public boolean onTouchEvent(MotionEvent event) {
45               //此处省略触摸引发的事件，后面步骤将给出解决过程
46          }
47          @Override
48          public void surfaceChanged(SurfaceHolder holder, int format, int width,
49                    int height) {
50          }
51          @Override
52          public void surfaceCreated(SurfaceHolder holder){
53               paint=new Paint()                                   //创建画笔
54               paint.setAntiAlias(true);                           //打开抗锯齿
55               createAllThreads();                                 //创建所有线程
56               initBitmap();                                       //初始化位图资源
57               numberWidth=numberBitmaps[0].getWidth()+3;          //得到数字图片的宽度
58               posFrom=-1;                                         //查询的开始位置置-1
59               queryResultStr=activity.query(posFrom,length);      //得到数据库中的数据
60               splitResultStrs=queryResultStr.split("/", 0);       //用"/"切分，且舍掉空串

61               startAllThreads();                                  //开启所有线程
62          }
63          @Override
64          public void surfaceDestroyed(SurfaceHolder holder) {     //销毁时触发
65                 boolean retry = true;
66                   stopAllThreads();                                //停止线程
67                   while (retry) {
68                     try {
69                     drawThread.join();
70                       retry = false;
71                     }
72                     catch (InterruptedException e) {e.printStackTrace();}
                                                        //不断地循环，直到刷帧线程结束
73                   }
74          }
```

```
75          public void initBitmap(){                          //初始化加载图片资源
76               //此处省略图片初始化加载的过程，后面将给出初始化过程
77          }
78          void createAllThreads(){
79               drawThread=new DrawThread(this);              //创建绘制线程
80          }
81          void startAllThreads(){                            //开启线程
82               drawThread.setFlag(true);                     //设置线程标志位
83               drawThread.start();
84          }
85          void stopAllThreads(){                             //结束线程
86               drawThread.setFlag(false);
87          }
88          private class DrawThread extends Thread{
89          //省略用于重绘 Surfaceview 的线程 DrawThread，前面多次进行讲解，这里就不再赘述
90     }
```

- 第 5～18 行为相应变量的声明。
- 第 19～25 行类 HinghJifenSurfaceView 的构造器。
- 第 52～62 行为完成一些初始化工作，包含画笔的设置、积分的查询等操作。
- 第 78～87 行为对该类用到的线程进行的统一操作，包含创建线程、开启线程以及停止线程。

（2）日期绘制的方法 drawDateBitmap，实现代码如下。

```
1     public void drawDateBitmap(                                //日期绘制方法
2          String numberStr,float endX,float endY,
3          Canvas canvas,Paint paint){                           //函数的参数
4          for(int i=0;i<numberStr.length();i++){                //遍历字符串
5               char c=numberStr.charAt(i);                      //得到当前字符
6               if(c=='-'){                                      //绘制横杠
7                    canvas.drawBitmap(gangBitmap,endX-numberWidth*(numberStr.length
                     ()-i), endY, paint);
8               }
9               else{
10                   canvas.drawBitmap                            //绘制数字
11                   (
12                        numberBitmaps[c-'0'],
13                        endX-numberWidth*(numberStr.length()-i),
14                        endY,
15                        paint
16                   );
17    }}}
```

说明　通过数据库的查询得到游戏的积分以及相对应的玩家游戏时间，该方法为绘制时间的方法，遍历时间字符串遇到横杠画横杠，遇到数字绘制数字，这样就完成了事件的绘制。

（3）触摸事件的方法实现，详细代码如下。

```
1     public boolean onTouchEvent(MotionEvent event) {           //触摸事件
2          int y = (int) event.getY();                           //当前触摸位置的 y 坐标
3          switch(event.getAction()){
4          case MotionEvent.ACTION_DOWN:                         //按下设置事件
5               downY=y;
6               break;
7          case MotionEvent.ACTION_UP:                           //抬起事件
8               upY=y;                                           //位置 y 坐标
```

```
9                if(Math.abs(downY-upY)<20){                          //在域值范围内，不翻页
10                    return true;
11               }
12               else if(downY<upY){                                  //往下拖动
13                    if(this.posFrom-this.length>=-1){               //下面还有积分数据,没有到头
14                         this.posFrom-=this.length;
15               }}
16               else{                                                //往上拖动
17                    if(this.posFrom+this.length<activity.getRowCount()-1){
18                         this.posFrom+=this.length;
19               }}
20               queryResultStr=activity.query(posFrom,length);      //得到数据库中的数据
21                    splitResultStrs=queryResultStr.split("/", 0);  //用"/"切分,且舍掉空串
22               break;
23          }
24          return true;
25     }
```

说明 通过对这个触摸事件的开发，可以实现积分榜查询动态翻页的效果，主要技术是数据查询的知识，通过记录每个页面数据的条数，记录当前查询游标的位置，动态实现信息的查询。

（4）图片资源加载的方法 initBitmap()，实现代码如下。

```
1     public void initBitmap(){                                        //图片加载
2          bgBitmap=BitmapFactory.decodeResource(this.getResources(), R.drawable.
           jifenbackground);
3          numberBitmaps=new Bitmap[]{
4               BitmapFactory.decodeResource(this.getResources(), R.drawable.d0),
                                                                        //数字图片
5               BitmapFactory.decodeResource(this.getResources(), R.drawable.d1),
6               BitmapFactory.decodeResource(this.getResources(), R.drawable.d2),
7               BitmapFactory.decodeResource(this.getResources(), R.drawable.d3),
8               BitmapFactory.decodeResource(this.getResources(), R.drawable.d4),
9               BitmapFactory.decodeResource(this.getResources(), R.drawable.d5),
10              BitmapFactory.decodeResource(this.getResources(), R.drawable.d6),
11              BitmapFactory.decodeResource(this.getResources(), R.drawable.d7),
12              BitmapFactory.decodeResource(this.getResources(), R.drawable.d8),
13              BitmapFactory.decodeResource(this.getResources(), R.drawable.d9),
14              BitmapFactory.decodeResource(this.getResources(), R.drawable.d0),
15         };
16        gangBitmap=BitmapFactory.decodeResource(this.getResources(), R.drawable.gang);
                                                                        //横杠图片
17    }
```

- 第 2 行为积分榜界面的背景图片。
- 第 3～15 行为积分榜界面中用到的数字图片组，包含 0～9 十张图片。
- 第 16 行为积分榜界面中事件绘制时连接数字的横杠图片。

15.6 游戏界面 GameView 及相关类

前面已经介绍了除游戏界面的其他所有界面，使读者对界面的开发有了一个深入的了解，从本节开始将讲解游戏开发的模块，逐步介绍塔防的实现过程。

15.6.1　精灵怪物 Target 类

本游戏主要任务是守住精灵怪物不让其接近主城，所以，怪物精灵是游戏的主体，为了实现怪物精灵的动态效果，开发了怪物的行走、辨别方向等模块，下面将分步骤详细介绍这方面知识。

（1）首先介绍 Target 类实现的整体框架代码，具体各模块功能的实现，将陆续给出，代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 Target.java。

```
1    package com.bn.tag;
2    import java.util.List;
3    //此处省略部分引入包类，读者可自行参见随书光盘代码
4    public class Target {
5         GameView gameView;                                  //GameView 引用
6         Bitmap bitmap;                                      //用于绘制的图片
7         float xspan=SINGLE_FOOT;                            //目标步进
8         float yspan=SINGLE_FOOT;                            //目标步进
9         double direction;                                   //怪物精灵行走的方向
10        float ballx=20;                                     //精灵出现的起始位置的 X 坐标
11        float bally=180;                                    //精灵出现的起始位置的 Y 坐标
12        int ii=0;
13        int jj=0;
14        Bitmap bitmapma;                                    //图片引用
15        static List<Target> ta=new Vector<Target>();          //最终进入营地的怪物列表
16        double num=0;                                       //标志位
17        boolean workflag=true;                              //换帧标志位
18        int bloodsum;                                       //当前血量
19        int bloodsumNO;                                     //不变的总血量
20        int state;                                          //0 代表怪物 1，1 代表怪物 2，2 代表怪物 3
21        public Target(GameView gameView,Bitmap bitmap,float ballx,float bally,
22             int bloodsum,int bloodsumNO,int state,double direction,int ii){
23             this.gameView=gameView;                        //GameView
24             this.bitmap=bitmap;                            //当前怪物精灵图片
25             this.ballx=ballx;                              //当前怪物精灵坐标
26             this.bally=bally;
27             this.bloodsum=bloodsum;                        //当前血量
28             this.bloodsumNO=bloodsumNO;                    //当前怪物种类的总血量
29             this.state=state;                              //当前怪物的种类
30             this.direction=direction;                      //方向
31             this.ii=ii;                                    //方向索引
32        }
33        public void drawSelf(Canvas canvas,Paint paint){     //绘制方法
34             Bitmap[] bit=new Bitmap[4];                    //新建 Bitmap 数组
35             bit=getbitmap(direction,state);                //得到当前状态下的状态图片组
36             bitmapma=getbitmapma(bit);                     //得到当前的状态图片
37             float x1=ballx-19+(SINGLE_RODER-GUAI_BLOOD_WEIGHT)/2;
                                                              //怪物精灵绘制起点 x 坐标
38             float y1=bally-19-10-GUAI_BLOOD_HEIGHT-5;        //怪物精灵绘制起点 y 坐标
39             canvas.drawBitmap(bitmapma, ballx-19,bally-19-10,paint); //绘制怪物精灵
40             if(bloodsum>0){                                  //怪物没有阵亡绘制声明
41                  canvas.drawBitmap(gameView.shengming01,x1,y1,paint); //生命图片的底色
42                  canvas.save();
43                  canvas.clipPath(makePathDashTarget(bloodsumNO,bloodsum));
                                                              //得到当前血量纹理路径
44                  canvas.drawBitmap(gameView.shengming, x1,y1, null); //绘制当前的血量
45                  canvas.restore();                         //防止对后面的绘制产生影响
46             }
47        }
48        public void goRode(){
49             //省略方法本体，具体实现代码后面将详细介绍
50        }
```

```
51          public void setWorkFlag(boolean workflag) {
52               this.workflag = workflag;
53          }
54          public double CallDirection(float x,float y){
55               //此处省略求解方向的方法代码，将在后面进行详细介绍
56          }
57          public Bitmap[] getbitmap(double direction,int state){
58               //省略方法体，具体代码将在后面进行详细讲解
59          }
60          public Bitmap getbitmapma(Bitmap[] bitmap){
61               //和上一个方法共同实现不同状态下怪物精灵图片的求解，后面将详细讲解
62          }
63          private  Path makePathDashTarget(float sumblood,float currentblood) {
64            //根据总的声明图片得到当前声明图片的四边形纹理路径
65     }}
```

- 第 5～20 行为一些变量的声明。
- 第 21～32 行代码为 Target 类的构造器。
- 第 33～47 行为绘制当前的怪物精灵以及当前怪物精灵的血量，这里的怪物精灵的血量采用的是绘制了红色的底色，然后用绿色图片纹理绘制表示当前血量，这样绘制更加美观合理。
- 第 48～65 行为怪物行走、方向设置、状态换帧以及生命血槽路径获得等方法，但此处已省略方法体，将在后面进行分步详细讲解。

（2）上面实现了怪物精灵的框架代码，接下来将讲解精灵怪物的行走。为此开发了方法 goRode()，通过这个方法可以实现怪物精灵的行走，方法代码如下。

```
1     public void goRode(){                                          //怪物行走方法体
2          float x1=(float) (ballx+xspan*Math.cos(direction)); //行走一步后的怪物位置坐标
3          float y1=(float)(bally+xspan*Math.sin(direction));
4          boolean backFlag=false;                                   //判断是否走出路径的标志位
5          float tempX=(float) (x1-SINGLE_PIC/2+0.01);               //怪物的左上侧坐标
6          float tempY=(float) (y1-SINGLE_PIC/2+0.01);
7          int tempCol=(int)((tempX/SINGLE_RODER>=0)?(tempX/SINGLE_RODER):-1);
                                                                     //左上侧坐标位置
8          int tempRow=(int)((tempY/SINGLE_RODER>=0)?(tempY/SINGLE_RODER):-1);   //行数
9          if(tempCol==-1||
10              tempRow==-1||                                        //左上侧位置超出了地图
11              MAP[tempRow][tempCol]==0){                           //当前位置没有路径
12                   backFlag=true;                                  //标志位为真
13         }
14         tempX=(float) (x1+SINGLE_PIC/2-0.01);                     //怪物精灵右上侧坐标
15         tempY=(float) (y1-SINGLE_PIC/2+0.01);
16         tempCol=(int)((tempX/SINGLE_RODER>=0)?(tempX/SINGLE_RODER):-1);//右上侧点行列
17         tempRow=(int)((tempY/SINGLE_RODER>=0)?(tempY/SINGLE_RODER):-1);
18         if(tempCol==-1||
19              tempRow==-1||
20              MAP[tempRow][tempCol]==0) {                          //当前位置没有路径
21                   backFlag=true;
22         }
23         tempX=(float) (x1-SINGLE_PIC/2+0.01);                     //左下侧 XY 坐标
24         tempY=(float) (y1+SINGLE_PIC/2-0.01);
25         tempCol=(int)((tempX/SINGLE_RODER>=0)?(tempX/SINGLE_RODER):-1);//左下侧点行列
26         tempRow=(int)((tempY/SINGLE_RODER>=0)?(tempY/SINGLE_RODER):-1);
27         if(tempCol==-1||
28              tempRow==-1||
```

```
29              MAP[tempRow][tempCol]==0){                          //当前位置没有路径
30                   backFlag=true;
31         }
32         tempX=(float) (x1+SINGLE_PIC/2-0.01);                    //右下侧 XY 坐标
33         tempY=(float) (y1+SINGLE_PIC/2-0.01);
34         tempCol=(int)((tempX/SINGLE_RODER>=0)?(tempX/SINGLE_RODER):-1);//右下侧点行列
35         tempRow=(int)((tempY/SINGLE_RODER>=0)?(tempY/SINGLE_RODER):-1);
36         if(tempCol==-1||
37              tempRow==-1||
38              MAP[tempRow][tempCol]==0){                     //右下侧所处位置没有路径
39                   backFlag=true;                            //标志位设置为真
40         }
41         if(tempCol==homeLocations[0][1]&&tempRow==homeLocations[0][0]){
                                                               //判断是否最终到达营地
42              gameView.homeTh.setFlag(true);                 //设置营地被侵袭，抖动线程开启
43              gameView.activity.shake();                     //设置振动
44              ta.add(this);                                  //当前怪物从列表中删除
45              if(gameView.bloodNUM==0){                      //当前营地的血量为 0
46                   gameView.bloodNUM=0;
47              }
48              else if(gameView.bloodNUM>=1){                 //如果营地的血量不为零
49                   gameView.bloodNUM-=1;                     //怪物侵袭，血量减 1
50         }}
51         if(!backFlag){                                      //怪物前进之后还是有路径
52              ballx=x1;                                      //怪物精灵行走
53              bally=y1;
54              num++;                                         //怪物精灵走的步数
55         }
56         else if(backFlag){                                  //不能沿原方向继续行走
57              ii++;
58              direction=CallDirection(MIDDLE_MAP[ii+1][0]-MIDDLE_MAP[ii][0], //改变方向
59              MIDDLE_MAP[ii+1][1]-MIDDLE_MAP[ii][1]);
60    }}
```

- 第 5～13 行得到当前怪物精灵的左上侧坐标，通过坐标计算得出当前位置所处游戏地图上的行列数，判断此位置是否有路径，并设置可行走的标志位。
- 第 14～22 行得到当前怪物精灵的右上侧坐标，通过坐标计算得到当前位置所处游戏地图上的行列数，判断此位置是否有路径，并设置可行走的标志位。
- 第 23～31 行得到当前怪物精灵的左下侧坐标，通过坐标计算得到当前位置所处游戏地图上的行列数，判断此位置是否有路径，并设置可行走的标志位
- 第 32～40 行得到当前怪物精灵的右下侧坐标，通过坐标计算得到当前位置所处游戏地图上的行列数，判断此位置是否有路径，并设置可行走的标志位
- 第 41～50 行为判断怪物精灵是否侵袭到营地，如果是，开启营地抖动线程，同时设置手机振动，表明营地有精灵袭击，同时将该精灵从怪物列表中删除。
- 第 51～55 行表明当前怪物可以顺着当前方向行走。
- 第 56～60 行表明怪物不能沿当前方向继续行走，要改变方向。

（3）怪物的行走问题解决了，下面要解决的问题是，怪物精灵的行走方向，要使怪物在转弯的时候会自动转弯，控制方向的函数方法 CallDirection()，实现代码如下。

```
1    public double CallDirection(float x,float y){          //怪物精灵方向设置方法
2         double result = 0.0;                              //结果变量
3         if(x==0&&y>0)
```

```
4           {
5                result = (0.5*Math.PI);                    //90°
6           }
7           else if(x==0&&y<0)
8           {
9                result=(1.5*Math.PI);                      //270°
10          }
11          else if(y==0&&x>0)
12          {
13               result=(0.0*Math.PI);                      //0°
14          }
15          else if(y==0&&x<0)
16          {
17               result=(1*Math.PI);                        //180°
18          }
19          return  result;
20     }
```

说明 通过这个方法得出了怪物精灵处于不同位置时，通过计算得出下一步怪物精灵要走的方向，这样就可以看到怪物精灵当行走到弯道时可以自动转身寻找正确的方向。

（4）上面的几个步骤实现了怪物行走和怪物精灵行走的方向问题，但是方向变了，怪物没有转身，所以，要加载不同方向的怪物精灵图片，刷新不同的方向图片，这样就实现了怪物的转身问题。同理在同一方向上刷新不同行走状态的图片就形成了怪物的动态行走，方法实现代码如下。

```
1     public Bitmap[] getbitmap(double direction,int state){
                                                   //获取同一方向上的怪物精灵的图片数组
2          Bitmap[] result=new Bitmap[4];          //结果数组的声明
3          if(direction==0.5*Math.PI)              //怪物方向为 90° 行走，即向上行走
4          {
5               result=gameView.targetbottom[state];  //获得当前一组图片
6          }
7          if(direction==1.5*Math.PI)              //怪物方向为 270° 行走即向下行走
8          {
9               result=gameView.targettop[state];     //获取图片组
10         }
11         if(direction==0.0*Math.PI)              //怪物水平向右行走
12         {
13              result=gameView.targetstraight[state];//获取图片组
14         }
15         return result;
16    }
17    public Bitmap getbitmapma(Bitmap[] bitmap){      //获得同一方向上不同状态的图片
18         Bitmap result = null;
19         if(workflag)                            //工作标志位
20         {
21              result=bitmap[(jj++)%bitmap.length];  //得到该状态下的图片数组
22         }
23         else if(!workflag)                      //停止换帧工作
24         {
25              result=bitmap[0];                     //统一设置状态
26         }
```

```
27          return result;                                  //返回结果
28    }
```

- 第 1～16 行通过传入方向角以及怪物的种类得到当前怪物精灵行走方向下的一组图片。
- 第 17～28 行为得到某一行走方向上的某一状态下的怪物精灵图片。

（5）上面实现了怪物精灵的一些开发代码，若希望怪物在攻击时能够显示当前的血量，下面就详细讲解和怪物精灵同步的血量的解决问题。这里通过得到当前怪物的总血量和当前血量得到当前血量在图片上的位置，然后选择性地画出当前血量。实现代码如下。

```
1     public void drawSelf(Canvas canvas,Paint paint) {          //绘制当前血量和怪物精灵
2           //省略部分代码，读者可参见随书光盘
3           if(bloodsum>0){
4                 canvas.drawBitmap(gameView.shengming01,x1,y1,paint);    //绘制怪物精灵
5                 canvas.save();                                          //保存
6                 canvas.clipPath(makePathDashTarget(bloodsumNO,bloodsum));
                                                            //绘制四边形得到路径
7                 canvas.drawBitmap(gameView.shengming, x1,y1, null);     //绘制当前血量
8                 canvas.restore();                         //恢复
9           }}
10    private  Path makePathDashTarget(float sumblood,float currentblood) {
                                                            //得到当前血量的四边形范围
11          Path p = new Path();
12          float x1=ballx-19+(SINGLE_RODER-GUAI_BLOOD_WEIGHT)/2;    //顺时针起点
13          float y1=bally-19-10-GUAI_BLOOD_HEIGHT-5;
14          float x2=ballx-19+(SINGLE_RODER-GUAI_BLOOD_WEIGHT)/2+    //第二个点
15                        currentblood/sumblood*GUAI_BLOOD_WEIGHT;
16          float y2=bally-19-10-GUAI_BLOOD_HEIGHT-5;
17          float x3=ballx-19+(SINGLE_RODER-GUAI_BLOOD_WEIGHT)/2+    //第三个点
18                        currentblood/sumblood*GUAI_BLOOD_WEIGHT;
19          float y3=bally-19-10-GUAI_BLOOD_HEIGHT-5+GUAI_BLOOD_HEIGHT;
20          float x4=ballx-19+(SINGLE_RODER-GUAI_BLOOD_WEIGHT)/2;
21          float y4=bally-19-10-GUAI_BLOOD_HEIGHT-5+GUAI_BLOOD_HEIGHT;
22          p.moveTo(x1, y1);                                       //起点
23          p.lineTo(x2, y2);                                       //第二个点
24          p.lineTo(x3, y3);                                       //第三个点
25          p.lineTo(x4, y4);                                       //第四个点
26          p.lineTo(x1, y1);
27          return p;
28    }
```

- 第 4 行为绘制怪物精灵图片。
- 第 5～8 行为通过怪物精灵当前的血量和总血量切割声明图片纹理，并且绘制该纹理。
- 第 10～28 行为得到当前怪物精灵血量纹理矩形的路径。

15.6.2　箭塔 SingleJianta 类

上述完成了游戏中的怪物精灵的开发，精灵可以行走、选择方向以及随着方向变化自动转身，下面介绍本游戏的另外一个主角—箭塔，该游戏分为两种类型的箭塔，长相不同，攻击范围有所区别，自然要建造不同箭塔的金钱数也不同。下面用代码介绍箭塔类的开发。代码如下所示。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 SingleJianta.java。

```
1     package com.bn.tag;
2     import android.graphics.Bitmap;
3     //此处省略了部分类的引入代码，读者可以自行查阅随书光盘的源代码
```

```
4    public class SingleJianta {
5         GameView gameView;                                  //GameView 的引用
6         Bitmap bitmap;                                      //用于绘制的图片
7         int clo;                                            //列数
8         int row;                                            //行数
9         int state;                                          //箭塔的设计半径，半径不同，箭塔不同
10        public SingleJianta(GameView gameView,Bitmap bitmap,int x,int y,int state){

11             this.gameView=gameView;                        //GameView
12             this.bitmap=bitmap;                            //当前箭塔的图片
13             this.clo=x;                                    //箭塔要放置的地图列数
14             this.row=y;                                    //箭塔要放置的地图行数
15             this.state=state;                              //箭塔的射击半径
16        }
17        public void drawSelf(Canvas canvas,Paint paint){
18             float x1=SINGLE_RODER*clo;                     //绘制箭塔图片的起点 x 坐标
19             float y1=SINGLE_RODER*row-JIAN_TA_HEIGHT+SINGLE_RODER; //绘制起点的 y 坐标
20             canvas.drawBitmap(bitmap, x1,y1,paint);        //绘制箭塔图标
21   }}
```

- 第 5～9 行为一些常用变量的声明。
- 第 10～16 行箭塔类构造器的声明，箭塔图标种类、箭塔位置和箭塔半径定位一个唯一的箭塔。
- 第 17～21 行为通过计算出绘制图片的起点 x、y 坐标绘制当前箭塔。

15.6.3 箭 Shell 类

上面实现了游戏中最重要的两个主要角色精灵怪物和箭塔类的开发设计，下面要讲解的是连接两个角色的桥梁——箭 Shell 类，玩家通过摆放箭塔发射箭射杀精灵怪物，下面就将分布介绍这部分内容。

（1）首先介绍箭 Shell 类的整体框架代码，具体代码实现如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 Shell.java。

```
1    package com.bn.tag;
2    import java.util.List;
3    //此处省略部分要引入的包的名称，读者可参见随书光盘代码
4    public class Shell {
5         GameView gameView;                                  //GameView 引用
6         private Bitmap bitmap;                              //位图
7         float shellx;                                       //箭行走的 x 坐标
8         float shelly;                                       //y 坐标
9         SingleJianta jianta;                                //发射该箭的箭塔
10        static List<Shell> shl=new Vector<Shell>();         //箭临时列表
11        static List<Target> tas=new Vector<Target>();       //精灵临时列表
12        Target tartee;                                      //当前箭要射击的精灵对象
13        double direction;                                   //方向
14        public Shell(GameView gameView,Bitmap bitmap,       //构造器
15        float shellx,float shelly,Target tartee,SingleJianta jianta){
16             this.gameView=gameView;                        //GameView
17             this.bitmap=bitmap;                            //位图
18             this.shellx=shellx;                            //箭的坐标
19             this.shelly=shelly;
20             this.tartee=tartee;                            //绑定怪物精灵
21             this.jianta=jianta;                            //绑定箭塔
22        }
```

```
23        public void drawSelf(Canvas canvas,Paint paint){     //绘制
24             go();
25             float dnX=shellx;                                //当前箭的 x 坐标
26             float dnY=shelly;                                //y 坐标
27             canvas.drawBitmap(bitmap, dnX, dnY,paint);       //绘制当前箭
28        }
29        public void go(){                                     //箭行走的方法
30             //此处省略箭行走的方法体，后面将对其进行详细介绍
31        }
32        public float getlength(float x1,float y1,float x2,float y2){
                                                          //求解两点之间的距离
33             float result=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2);//数学方法求解两点之间的距离
34             return result;                             //返回结果集
35        }
36      public double calDirection1(float x1,float y1,float x2,float y2){
37             //此处省略了箭追踪怪物精灵过程中的方向问题，后面将详细介绍相关内容
38      }
39             //此处省略了箭和怪物精灵的碰撞判定方法实现，后面将详细讲解
40   }
```

- 第 5～13 行为一些变量的声明。
- 第 14～22 行为类 Shell 的构造器。
- 第 29～40 行省略了箭行走、方向设定以及和怪物精灵碰撞判定的方法。

（2）上面实现了 Shell 类的整体框架，使读者对箭类的开发有一个简单的了解，下面介绍箭的行走过程，具体实现方法 go()的代码如下。

```
1    public void go(){
2         direction=calDirection1(shellx,shelly,tartee.ballx,tartee.bally);//方向
3         float llx=(float) (SPEED*Math.sin(direction*Math.PI/180)+shellx);//x 坐标
4         float lly=(float) (SPEED*Math.cos(direction*Math.PI/180)+shelly);//y 坐标
5         if(tas.contains(tartee)){                   //如果当前箭绑定的怪物已亡
6              shl.add(this);                         //该箭进入删除列表
7         }
8         if(IsTwoRectCross                           //一个矩形的 4 个顶点之一是否在另一个矩形内
9         (
10             llx,lly,JIAN_TOU_WEIGHT,JIAN_TOU_HEIGHT,   //左上点 x,y 坐标，长、宽
11             tartee.ballx-SINGLE_PIC/2+20,tartee.bally-SINGLE_PIC/2,SINGLE_PIC-15,
               SINGLE_PIC-15
12        )){                                          //箭击中怪物精灵的方法
13             tartee.bloodsum-=1;                      //怪物精灵血量自减 1
14             if(tartee.bloodsum==0){                  //怪物精灵当前血量为 0
15                  if(!tas.contains(tartee)){
16                       gameView.doller+=5;            //击杀一个怪物精灵获赏金 5
17                       gameView.shaNUM+=1;            //杀敌数加 1
18                       gameView.shuijingMiddleNum+=1; //水晶物品中间计算数加 1
19                       tas.add(tartee);               //绑定怪物进入删除列表
20                  }
21             }
22             shl.add(this);                           //当前箭进入删除列表
23             if(gameView.activity.isSoundOn()){       //如果系统开启音效
24                  gameView.playSound(2, 0);           //音效响起
25             }
26        }
27        else if(getlength(llx,lly,jianta.clo*SINGLE_RODER,        //箭超出箭塔射程
28                  jianta.row*SINGLE_RODER)>R_LENGTH*R_LENGTH){
29             shl.add(this);                                        //箭进入删除列表
30        }
31        else{
32             shellx=llx;                                           //x 坐标
```

```
33                shelly=lly;                                        //y坐标
34    }}
```

- 第 5～7 行为判断该箭绑定的怪物是否已经阵亡，如果怪物精灵已经不出现了，箭进入删除列表。
- 第 8～26 行为判断箭是否射击到怪物精灵身上，如果射击到怪物身上，进行相应的金钱、血量、音效处理。
- 第 27～30 行为当箭超出箭塔射击范围时，箭进入删除列表，在游戏界面中消失。
- 第 31～34 行为箭没有任何碰撞，可以沿该方向继续向前。

（3）接下来要实现的是箭追击怪物精灵过程中方向的方法，本游戏中对箭塔射出的箭的行走方向时刻是朝向一开始绑定的怪物精灵对象，开发的方法 calDirection1（）代码如下。

```
1    public double calDirection1(float x1,float y1,float x2,float y2){
2         double direction = 0;                              //声明方向的结果变量
3         float dx=x1-x2;                                    //箭和怪物精灵之间的坐标差值
4         float dy=y1-y2;
5         if(dx!=0||dy!=0){
6              if(dx>0&&dy>0){                               //相对于屏幕，箭在怪物的右下方
7                   direction=180+Math.toDegrees(Math.atan(dx/dy));
8         }
9              else if(dx<0&&dy>0){                          //相对于屏幕，箭在怪物的左下方
10                  direction=180-Math.toDegrees(Math.atan(-dx/dy));
11             }
12             else if(dx<0&&dy<0){                          //箭在怪物的左上方
13                  direction=Math.toDegrees(Math.atan(dx/dy));
14             }
15             else if(dx>0&&dy<0){                          //箭在怪物的右上方
16                  direction=360-Math.toDegrees(Math.atan(dx/-dy));
17             }
18             else if(dx==0){
19                  if(dy>0){                                //箭在怪物的正下方
20                       direction=180;
21                  }
22                  else{                                    //箭在怪物的正上方
23                       direction=0;
24                  }
25             }
26             else if(dy==0){
27                  if(dx>0){
28                       direction=270;                      //相对于屏幕，箭在怪物的右边
29                  }
30                  else{
31                       direction=90;                       //箭在怪物的左边
32        }}}
33        return direction;
34   }
```

说明　通过这个方法解决了箭塔射箭过程中，箭的射击方向问题，箭相对于怪物精灵行走的方向是时刻在变的，通过时刻计算箭和绑定怪物精灵之间的坐标差值得出箭的射击方向。

（4）上面代码实现了箭的射击以及方向确定等过程，当然，游戏最终要的效果是箭能射击怪物精灵，怪物精灵血量为 0 从而消失，下面就详细讲解箭与怪物精灵的碰撞过程，具体代码如下。

```
1    public static boolean IsTwoRectCross(        //一个矩形的 4 个顶点之一是否在另一个矩形内
2         float xLeftTop1,float yLeftTop1,float length1,float width1,
                                                   //左上点 x,y 坐标，长、宽
3         float xLeftTop2,float yLeftTop2,float length2,float width2){
4         if(
5              isPointInRect(xLeftTop1,yLeftTop1,xLeftTop2,yLeftTop2,
6                                  length2,width2)||                    //左上顶点
7              isPointInRect(xLeftTop1+length1,yLeftTop1,xLeftTop2,yLeftTop2,
8                                  length2,width2)||                    //右上顶点
9              isPointInRect(xLeftTop1,yLeftTop1+width1,xLeftTop2,
10                                 yLeftTop2,length2,width2)||          //左下顶点
11             isPointInRect(xLeftTop1+length1,yLeftTop1+width1,xLeftTop2,
12                                 yLeftTop2,length2,width2)||          //右下顶点
13             isPointInRect(xLeftTop2,yLeftTop2,xLeftTop1,yLeftTop1,
14                                 length1,width1)||                    //左上顶点
15             isPointInRect(xLeftTop2+length2,yLeftTop2,xLeftTop1,
16                                 yLeftTop1,length1,width1)||          //右上顶点
17             isPointInRect(xLeftTop2,yLeftTop2+width2,xLeftTop1,yLeftTop1,
18                                 length1,width1)||                    //左下顶点
19             isPointInRect(xLeftTop2+length2,yLeftTop2+width2,xLeftTop1,
20                                 yLeftTop1,length1,width1)            //右下顶点
21        ){
22             return true;
23        }
24        return false;
25   }
26   public static boolean isPointInRect(                   //一个点是否在矩形内（包括边界）
27        float pointx,float pointy,
28        float xLeftTop,float yLeftTop,float length,float width){
29        if(
30             pointx>=xLeftTop&&pointx<=xLeftTop+length&& //判断测点是否在矩形内
31             pointy>=yLeftTop&&pointy<=yLeftTop+width
32        ){
33             return true;                                  //如果在，返回真
34        }
35        return false;                                      //返回假
36   }
```

- 第 1～25 行为判断箭塔射出的箭是否射中绑定的怪物精灵，通过判断计算怪物精灵的 4 个点中的任一位置坐标是否在箭的图片坐标范围内，同时也判断箭图标的 4 个点中的任一位置坐标是否在怪物精灵的图片的坐标范围内，如果满足任一条件，则说明箭射中怪物精灵。
- 第 26～36 行为判断一个点是否在一个矩形范围内的方法，为上一种方法的调用方法。

15.6.4　精灵怪物的出击现场——TargetNumThread

上面实现了精灵怪物类 Shell 的开发，通过新建 Shell 类可以得到不同种类、不同血量的怪物精灵，本游戏作为一款箭塔防守游戏，一定要有源源不断的怪物精灵侵袭，直到守城失败，游戏结

束，本小节将详细介绍怪物精灵的产生以及数量的控制，具体代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 TargetNumThread.java。

```
1    package com.bn.tag;
2    //此处省略部分引入包类，读者可参见随书光盘代码
3    public class TargetNumThread extends Thread {
4         GameView gameView;                                  //GameView 的引用
5         private int sleepSpan=10000;                        //睡眠数
6         private boolean flag=true;                          //工作线程标志位
7         private boolean whileflag=true;                     //线程开启标志位
8         private int distancespan=1000;                      //怪物间出现的时间间隔
9         private int blood_A=2;                              //不同怪物种类最低级的血量
10        private int blood_B=3;
11        private int blood_C=4;
12        private int NUMBER=20;                              //每 12 波改变一次怪物精灵的血量
13        double direction=0.0*Math.PI;                       //怪物精灵初始的行走方向
14        int tempk=1;                                        //中间变量
15        TargetNumThread(GameView gameView){
16             this.gameView=gameView;
17        }
18        @Override
19        public void run(){                                  //重写 run()方法
20             while(whileflag){                              //开启线程标志位
21                  try{
22                       Thread.sleep(distancespan);          //睡眠指定毫秒数
23                  }
24                  catch(Exception e){
25                       e.printStackTrace();                 //打印堆栈信息
26                  }
27                  if(flag){                                  //产生怪物精灵的工作标志位
28                       if(tempk<=NUMBER){                    //没有达到一波的数量上限
29                            tempk=tempk+1;
30                            if(tempk==NUMBER){               //数量达到一波上限
31                                 setblood();                 //调用怪物血量增加函数
32                            }
33                       }
34                       else {
35                            tempk=1;
36                       }
37                       if(gameView.alTarget1.size()==0){ //游戏界面中的怪物精灵都被击杀
38                            for(int i=0;i<tempk;i++){
39                                 if(flag){                   //工作标志位
40                                      double m = Math.random();
                                                //根据不同随机数产生不同怪物精灵
41                                      if(m<0.5){
42                                           gameView.alTarget1.add(new Target(game-
                                             View,
43                                           gameView.target1Bitmap,20,180,blood_A,
                                             blood_A,
44                                           0,direction,0)); //创建目标对象，加入列表
45                                      }
46                                      else if(m<0.8&&m>=0.5){
47                                           gameView.alTarget1.add(new Target(game-
                                             View,
48                                           gameView.target1Bitmap,20,180,blood_B,
                                             blood_B,
49                                           1,direction,0)); //创建目标对象，加入列表
50                                      }
51                                      else if(m>=0.8){
```

```
52                                          gameView.alTarget1.add(new Target(game-
                                            View,
53                                          gameView.target1Bitmap,20,180,blood_C,
                                            blood_C,
54                                          2,direction,0)); //创建目标对象，加入列表
55                                      }
56                                  }
57                                  else if(!flag){   //停止工作线程时，产生数量线程暂时停止
58                                      i--;          //返回到暂停前状态
59                                  }
60                                  try{
61                                  Thread.sleep(distancespan);
                                                //睡眠线程，为精灵间的出生间隔指定毫秒数
62                              }
63                              catch(Exception e){
64                                  e.printStackTrace();        //打印堆栈信息
65                          }}}}
66                      try{
67                      Thread.sleep(sleepSpan);                //睡眠线程，指定毫秒数
68                  }
69                  catch(Exception e){
70                      e.printStackTrace();                    //打印堆栈信息
71          }}}
72          public void setFlag(boolean flag) {                 //设置精灵产生工作标志位
73               this.flag = flag;
74          }
75          public void setwhileflag(boolean whileflag){        //设置线程开启标志位
76               this.whileflag=whileflag;
77          }
78          void setblood(){                                    //怪物精灵血量增加函数
79               blood_A=blood_A+2;
80               blood_B=blood_B+2;
81               blood_C=blood_C+2;
82      }}
```

- 第 4～14 行为一些变量的声明。
- 第 15～17 行为 TargetNumThread 类的构造器。
- 第 19～71 行为通过随机数产生不同种类的、不同血量的怪物精灵，这些精灵之间的出生间隔为 distancespan，当游戏中途暂停时，产生怪物精灵的工作进程关闭。结束暂停后，回到暂停前的状态。
- 第 72～77 行设置线程标志位和工作进程标志位。
- 第 78～82 行为设置每一定数量波的怪物侵袭之后血量增加。

15.6.5　箭塔控制发射线程——ShellNumThread

上面解决了游戏中所涉及的怪物精灵的开发问题，接下来要讲解的是箭塔发射箭的开发过程，游戏过程中可能有相当数量的怪物精灵经过一个箭塔，这时摆在箭塔面前的有一个问题，该射击哪一个怪物精灵呢？本游戏的解决方案是：在箭塔射程范围内总是射击走在最前面的怪物精灵，具体开发实现代码如下所示。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 ShellNum-Thread.java。

```
1    package com.bn.tag;
```

```
2    import static com.bn.tag.Constant.*;
3    //此处省略部分引入包类，读者可参见随书光盘代码
4    public class ShellNumThread extends Thread {
5         GameView gameView;                                  //GameView 引用
6         private boolean flag=true;                          //工作标志位
7         private boolean whileflag=true;                     //线程开启标志位
8         ShellNumThread(GameView gameView){                  //构造器
9              this.gameView=gameView;
10        }
11        @Override
12        public void run(){
13             while(whileflag){                              //线程开启标志位
14                  if(flag){
15                       List<Target> targetmid=new Vector<Target>(gameView.alTarget1);
                                                             //中间列表
16                       List<SingleJianta> jiantalista= //中间箭塔列表
17                                 new Vector<SingleJianta>(gameView.
                                   jiantaList);
18                       for(int i=0;i<jiantalista.size();i++){   //遍历箭塔列表
19                            List<Target> tartar=new Vector<Target>();
20                            float y1=jiantalista.get(i).row*SINGLE_RODER+SINGLE_
                              RODER/2;                                //底部中心位置
21                            float x1=jiantalista.get(i).clo*SINGLE_RODER+SINGLE_
                              RODER/2;
22                            float  y2=jiantalista.get(i).row*SINGLE_RODER-JIAN_TA_
                              HEIGHT
23                                      +SINGLE_RODER;
24                            for(int j=0;j<targetmid.size();j++){//遍历怪物精灵列表
25                                 if((y1-targetmid.get(j).bally)*(y1-targetmid.
                                   get(j).bally)
26                                 +(x1-targetmid.get(j).ballx)*(x1-targetmid.get(j).
                                   ballx)
27                                 <=jiantalista.get(i).state*jiantalista.get(i).
                                   state){                                //箭塔攻击范围内
28                                      tartar.add(targetmid.get(j)); //添加进临时列表
29                                 }
30                                 else{
31                                      continue;
32                                 }
33                            }
34                            if(tartar.size()!=0){                  //列表中存在怪物精灵
35                                 Target targett=getMaxNumtarget(tartar);
                                                                   //找到最前面的怪物精灵
36                                 gameView.shellsjian.add(new Shell(gameView,
                                   gameView.tubiao1,
37                                                        x1,y2,targett,
                                                          jiantalista.get(i)));
38                            }                            //新建箭
39                            tartar.clear();              //清空临时列表
40                  }}
41                  try{
42                  Thread.sleep(2000);                    //睡眠指定毫秒数
43             }
44             catch(Exception e){
45                  e.printStackTrace();                   //打印堆栈信息
```

```
46          }}}
47          public void setFlag(boolean flag) {                    //设置工作标志位
48               this.flag = flag;
49          }
50          public void setwhileflag(boolean whileflag){           //设置线程开启标志位
51               this.whileflag=whileflag;
52          }
53          public Target getMaxNumtarget(List<Target> target){ //在箭塔攻击范围内得到
                                                                //最前面的怪物精灵
54               Target result = target.get(0);              //结果变量
55               for(int i=1;i<target.size();i++){           //遍历箭塔范围内的怪物精灵列表
56                    if(result.num<target.get(i).num){
57                         result=target.get(i);             //找到最前面的怪物精灵
58               }}
59               return result;                              //返回结果
60     }}
```

- 第 4～10 行为类 ShellNumThread 的构造器。
- 第 18～33 行遍历游戏中的箭塔列表和游戏中的怪物精灵列表找到当前箭塔射程范围内行走的所有怪物精灵。
- 第 34～38 行从当前箭塔射程范围内的怪物列表中找到走在最前面的怪物精灵，新建箭，绑定当前怪物精灵，进行瞄准射击。
- 第 47～52 行为设置工作标志位和线程开启标志位。
- 第 53～60 行为在箭塔攻击范围内得到最前面的怪物精灵的函数方法。

15.6.6　游戏地图矩阵模拟化

前面已经完成了游戏中主要角色怪物精灵、箭塔以及箭的设计开发，本游戏对游戏界面的布局采取数学上的矩阵知识，将游戏界面类比成矩阵界面，根据矩阵中设置不同数字绘制不同的游戏场景，下面介绍游戏中的矩阵界面，实现代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 SingleGoJianta.java。

```
public static final int[][] MAP={              //游戏地图类比矩阵，手机屏幕为 480*800 横屏
     {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3},      //3 代表高的树木
     {3,2,2,2,0,0,0,4,4,4,0,0,0,0,0,2,3,3,3,3},
     {3,4,4,4,0,0,1,1,1,1,1,0,0,0,0,2,3,3,3,3},
     {3,2,0,0,0,4,1,0,0,0,1,0,0,0,0,2,4,4,4,3},      //4 代表花草
     {1,1,1,1,0,0,1,0,0,0,1,0,3,0,0,0,0,0,0,0},
     {2,2,0,1,0,2,1,0,0,0,1,0,3,0,1,1,1,1,1,0},
     {2,0,0,1,0,2,1,0,3,3,1,0,2,2,1,0,0,0,0,3},      //1 代表公路，怪物精灵行走在上面
     {2,0,0,1,0,2,1,0,3,3,1,0,2,2,1,0,0,0,0,3},
     {2,0,0,1,0,0,1,0,2,3,1,0,0,0,1,0,0,0,0,3},      //2 代表木桩
     {2,0,0,1,1,1,1,0,2,0,1,1,1,1,1,0,2,2,3,3},
     {3,3,0,0,0,0,0,0,2,0,0,0,0,0,0,2,2,3,3,3},
     {3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3}
};
```

说明　通过将游戏地图数学矩阵模拟化，能很好地布局游戏中的场景，只需设定矩阵中的元素值，即可完成游戏中场景的改变，本游戏中只用到了一张地图，读者可以在今后的编程当中采用这样的技术，开发加载多幅地图的游戏，做法与本游戏做法一样。

15.6.7 摆放箭塔守护城池

上面几个小节实现了怪物精灵、箭塔以及箭的开发，但是以一种什么样的形式摆放箭塔就成了一个要解决的问题，本游戏采取用手拖动箭塔进行摆放，操作不但简便，玩法酷、炫，符合广大玩家的意愿，下面就分步骤详细讲解这方面的内容。

（1）为了解决问题方便，该游戏又开发了摆放箭塔过程类 SingleGoJianta，实现代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 SingleGoJianta.java。

```
1    package com.bn.tag;
2    import android.graphics.Bitmap;
3    //此处省略部分引入包类，读者可参见随书光盘代码
4    public class SingleGoJianta {
5         GameView gameView;                                        //GameView 引用
6         Bitmap bitmap;                                            //用于绘制的图片
7         int clo;                                                  //箭塔在地图上的行列数
8         int row;
9         public SingleGoJianta(GameView gameView,Bitmap bitmap){  //函数构造器
10             this.gameView=gameView;
11             this.bitmap=bitmap;
12        }
13        public void drawSelf(Canvas canvas,Paint paint){     //绘制
14             float x1=SINGLE_RODER*clo;                      //得到绘制起点坐标
15             float y1=SINGLE_RODER*row-JIAN_TA_HEIGHT+SINGLE_RODER;
16             canvas.drawBitmap(bitmap, x1,y1,paint);         //绘制拖动过程中的箭塔坐标
17        }
18        public void getXY(float x,float y){                  //得到位置坐标
19             clo=(int)(x/SINGLE_RODER);
20             row=(int)(y/SINGLE_RODER);
21   }}
```

- 第 5～8 行为一些变量的声明。
- 第 9～12 行为类 SngleGoJianta 的构造器。
- 第 13～17 行为拖动箭塔过程中，得到箭塔当前坐标，绘制拖动过程中的箭塔。
- 第 18～212 行为动态获得当前箭塔拖动过程中的位置坐标。

（2）触摸选定箭塔，通过触摸手机屏幕上的不同箭塔图标进行箭塔拖动选定，实现代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
1    public boolean onTouchEvent(MotionEvent event) {          //游戏界面中的触摸方法
2         if(event.getAction() == MotionEvent.ACTION_DOWN){    //触摸中的按下操作
3              int x = (int) event.getX();                 //当前的触摸位置的 x 坐标
4              int y = (int) event.getY();                 //y 坐标
5              int temppx=(int) (x/SINGLE_RODER);          //得到当前位置在地图中的行列数
6              int temppy=(int) (y/SINGLE_RODER);
7              //省略其他操作的部分代码，后面将对其详细介绍
8              if(doller>=25&&x>SCREEN_HEIGHT-LEVEL_WEIGHT-100&&x<SCREEN_HEIGHT-
9                   LEVEL_WEIGHT-100+TIBIAO_WEIGHT
10                  &&y>SCREEN_WIDTH-TIBIAO_HEIGHT-10&&y<SCREEN_WIDTH-TIBIAO_HEIGHT-
11                  10+TIBIAO_HEIGHT&&caidanflag==false){//触摸到箭塔图标 1
12                  taflag1=true;                       //设置标志位为 true
13                  sg.getXY(x, y);                     //获得坐标
14                  return true;                        //返回 true
15             }
```

```
16              if(doller>=15&&x>SCREEN_HEIGHT-LEVEL_WEIGHT-10&&x<SCREEN_HEIGHT-
17                   LEVEL_WEIGHT-10+TIBIAO_WEIGHT
18                   &&y>SCREEN_WIDTH-TIBIAO_HEIGHT-10&&y<SCREEN_WIDTH-
19                   TIBIAO_HEIGHT-10+TIBIAO_HEIGHT&&caidanflag==false){//触摸到箭塔图标 2
20                   taflag2=true;                          //设置标志位为 true
21                   sgsg.getXY(x, y);                      //获得坐标
22                   eturn true;                            //返回 true
23              }
24              candoiflag=false;
25         }
26         //此处省略其他操作代码，后面将对其详细介绍
27    }
```

- 第 3～6 行为触摸当前位置得到当前位置坐标，转换成地图矩阵中的行列数。
- 第 8～15 行触摸到箭塔图标 1，可以拖动摆放的前提条件是，玩家金钱数要大于箭塔价钱数。
- 第 16～23 行触摸到箭塔图标 2。

（3）当选定好箭塔时，便可以进行拖动摆放了，同时，为了标记哪些地方可以摆放，哪些地方不可以摆放，系统采用红色的圆圈表示拖动经过的地方可以摆放，没有红色圆圈不可以摆放。该开发过程中用到了类 CanOrIndexPic，读者可自行参见随书光盘代码，这里就不再进行介绍。其他开发代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
1     public boolean onTouchEvent(MotionEvent event) {
2          //省略部分代码，读者可参见随书光盘
3          else if (event.getAction() == MotionEvent.ACTION_MOVE){  //移动事件
4          int x = (int) event.getX();                              //获得当前位置坐标
5          int y = (int) event.getY();
6          if(taflag1==true||taflag2==true){                        //已经触摸到相应箭塔
7               middleballx=x;
8               middlebally=y;
9               sg.getXY(x, y);                                     //得到位置坐标
10              sgsg.getXY(x, y);
11              if(MAP[(int)(y/SINGLE_RODER)][(int)(x/SINGLE_RODER)]==0){//当前可放置箭塔
12                   canLocation.ifdraw=true;                       //当前位置可放置
13                   canLocation.getXY(x, y);                       //获得坐标
14              }
15              else{
16                   canLocation.ifdraw=false;                      //当前位置不可放置
17              }
18              return true;
19         }
20         else {
21              return true;                                        //返回
22         } }
23         return false;
24    }
```

说明　当玩家触摸选定了一个箭塔图标，在拖动过程中系统会根据当前的位置是否有树木、树桩等物品判断显示当前位置是否可放，并设置可放置标志位。

绘制拖动过程中的箭塔以及在可放置位置处显示可放置标志，实现代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
1    if(taflag1||taflag2){                                          //箭塔触摸标志位
2         for(int i=0;i<20;i++){                                    //绘制竖线网格
3              float xStart=SINGLE_RODER*i;                         //起点
4              float yStart=0;
5              float xEnd1=SINGLE_RODER*i;                          //终点
6              float yEnd1=SINGLE_RODER*12;
7              paint.setARGB(255,255,255,255);                      //设置颜色和可见度
8              canvas.drawLine(xStart, yStart, xEnd1, yEnd1, paint);   //绘制竖线
9         }
10        for(int i=0;i<12;i++){                                          //绘制横线
11             float yStart=SINGLE_RODER*i;                            //起点
12             float xStart=0;
13             float yEnd1=SINGLE_RODER*i;                             //终点
14             float xEnd1=SINGLE_RODER*20;
15             paint.setARGB(255,255,255,255);                      //设置颜色和可见度
16             canvas.drawLine(xStart, yStart, xEnd1, yEnd1, paint);   //绘制横线
17        }
18        if(canLocation.ifdraw==true){                            //绘制可放置标志位
19             canLocation.drawSelf(canvas, paint);
20        }
21        if(taflag1){                                             //绘制拖动过程中箭塔 1 图标
22             sg.drawSelf(canvas, paint);
23        }
24        else if(taflag2){                                        //绘制拖动过程箭塔 2 图标
25             sgsg.drawSelf(canvas, paint);
26   }}
```

- 第 2～17 行为在箭塔拖动过程中绘制地图网格，方便玩家在相应位置进行箭塔摆放。
- 第 18～20 行为当前拖动位置如果可放置箭塔，绘制可放置图标的标志位。
- 第 21～26 行根据触摸的箭塔图标绘制拖动过程中相应的箭塔图标。

（4）在决定要摆放的位置摆放箭塔，相应的代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
1    public boolean onTouchEvent(MotionEvent event) {              //摆放箭塔
2         //省略部分代码，读者可参见相应代码
3         else if (event.getAction() == MotionEvent.ACTION_UP) {   //触摸抬起事件
4              int x = (int) event.getX();                          //抬起位置坐标
5              int y = (int) event.getY();
6              int xx=(int) (x/SINGLE_RODER);                       //换算地图行列数
7              int yy=(int)(y/SINGLE_RODER);
8              if(MAP[yy][xx]==0&&taflag1==true||MAP[yy][xx]==0&&taflag2==true){
                                                                    //可放置箭塔标志
9                   if(taflag1==true){
10                       jiantaList.add(new SingleJianta(this,jianta,xx,yy,130));
                                                                    //新建箭塔 1
11                      doller=doller-25;                           //金钱花费
12                      taflag1=false;                              //标志位置 false
13                  }
14                  else if(taflag2==true){                         //箭塔 2
15                      jiantaList.add(new SingleJianta(this,jianta1,xx,yy,100));
                                                                    //新建箭塔 2
16                      doller=doller-15;                           //金钱花费
17                      taflag2=false;                              //标志位置 false
18                  }
19             }
20             taflag1=false;                                       //标志位置 false
21             taflag2=false;
```

```
22                CanOrIndexPic.ifdraw=false;
23                return true;                                    //返回真
24            }
25            return false;
26        }
```

- 第 4～7 行获得当前触摸抬起的位置坐标。
- 第 9～13 行为当前位置放置箭塔 1，同时进行金钱减少，标志位置 false 操作。
- 第 14～18 行为当前位置放置箭塔 2，同时进行金钱减少，标志位置 false 操作。

15.6.8　游戏中的 2.5D 效果

本游戏采用的是 2.5D 技术开发的塔防游戏，运行游戏时给玩家一种立体的感觉，例如，在某些路段会出现箭塔遮挡怪物精灵，在某些路段怪物精灵遮挡箭塔，同时也可能路旁的植物也参与遮挡，该效果的实现是 2D 绘图的绘制顺序决定的，下面就详细讲解其实现的方法，代码如下所示。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
1     for(int i=0;i<12;i++){                                    //遍历游戏矩阵图
2         for(int j=0;j<20;j++){
3             if(MAP[i][j]==2){                                 //绘制树桩
4                 canvas.drawBitmap(smalltree, TREE_WEIGHT*j,TREE_HEIGHT*i, null);
5             }
6             else if(MAP[i][j]==3){                            //绘制高树
7                 canvas.drawBitmap(bigtree, BIG_TREE_WEIGHT*j,GONGLU_HEIGHT*i-
8                                 BIG_TREE_HEIGHT+GONGLU_HEIGHT, null);
9             }
10            else if(MAP[i][j]==4){                            //绘制花草
11                canvas.drawBitmap(flower, FLOWER_WEIGHT*j,FLOWER_HEIGHT*i, null);
12            }
13            synchronized (jiantaList){                        //绘制箭塔
14                for(SingleJianta sj:jiantaList){              //循环箭塔列表
15                    if(sj.clo==j&&sj.row==i){
16                        sj.drawSelf(canvas, paint);           //绘制箭塔
17                    }
18                }
19            }
20            synchronized (alTarget1){                         //绘制怪物精灵
21                for(Target tar:alTarget1){                    //循环怪物精灵列表
22                    if((int)(tar.ballx/SINGLE_RODER)==j&&(int)(tar.bally/
23                        SINGLE_RODER)==i){
24                        tar.drawSelf(canvas, paint);          //绘制怪物精灵
25    }}}}}
```

说明　本游戏采用的 2.5D 技术开发游戏，游戏界面采用数学矩阵模拟化形式抽象化，使开发工作难度大大降低，要实现游戏的 2.5D 效果，只是在刷新绘图的时候根据矩阵知识自左向右、自上而下的绘制顺序绘制当前位置的游戏场景，这样便形成了游戏的立体效果。

15.6.9　弹指间怪物灰飞烟灭——游戏水晶

前面已经完成了本塔防游戏中的大部分开发工作，为了增加游戏的可玩性，本游戏增加了游戏水晶模块，通过击杀一定数量的怪物精灵，系统会奖励玩家一个游戏水晶，玩家触摸屏幕上的水晶

模块便可轻松消灭游戏界面中的所有怪物，当然水晶技能不能连续使用，存在技能CD，效果与平时玩的英雄技能CD效果一致，下面就分步骤详细介绍这部分内容。

（1）绘制游戏水晶同时根据是否用过游戏水晶绘制其技能恢复过程，实现代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
1    canvas.drawBitmap(shuijing, SHUIJIAN_STARTX,SHUIJIAN_STARTY, null);//绘制水晶图标
2    shuijingNUM=shuijingMiddleNum/25;                                  //得到水晶数量
3    String shuijingStr=shuijingNUM+"";                                 //数量转化为字符型
4    int lengthshui=shuijingStr.length();                               //字符串长度
5    for(int i=0;i<lengthshui;i++){                                     //遍历字符串
6         int temp=shuijingStr.charAt(i)-'0';                           //得到数字字符
7         canvas.drawBitmap(iscore[temp], SHUIJIAN_STARTX+(4+i)*scoreWidth,15, null);
                                                                        //绘制数字
8    }
9    if(shuijing_Flag){                                                 //水晶技能恢复
10        canvas.save();
11        canvas.clipPath(makePathDash(shuijing_change_x,shuijing_change_y));
                                                                        //得到恢复纹理
12        canvas.drawBitmap(shuijing_cover, SHUIJIAN_STARTX,SHUIJIAN_STARTY, null);
                                                                        //绘制
13        canvas.restore();                                             //取消设定
14   }
15   if(shuijing_D_Z_Flag){                                             //绘制水晶技能释放效果
16        paint.setAlpha(alpha);                                        //设置可见度
17        canvas.drawBitmap(shuijingZCZ, 0,0, paint);                   //绘制效果图片
18        paint.setAlpha(255);                                          //恢复
19   }
```

- 第 1～8 行绘制游戏水晶的图标，根据当前游戏的杀怪物数量得出获得的水晶数，同时绘制水晶数量。
- 第 9～14 行判断水晶技能是否处于恢复之中，如果是，绘制恢复过程中的图片纹理。
- 第 15～19 行绘制游戏水晶技能释放效果。

（2）触摸游戏水晶释放技能，消灭界面中的怪物精灵，模块代码位于 GameView 类的方法 onTouchEvent()中，具体实现代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
1    if(shuijingNUM>0&&x>SHUIJIAN_STARTX&&x<SHUIJIAN_STARTX+SHUIJING_WEIGHT
2         &&y>SHUIJIAN_STARTY&&y<SHUIJIAN_STARTY+SHUIJIAN_HEIGHT){//界面中的水晶位置
3         alTarget1.clear();                                          //清空怪物精灵列表
4         shuijingMiddleNum-=25;                                      //水晶数量减 1
5         timeTh.setFlag(true);                                       //设置线程标志位
6         this.shuijing_Flag=true;                                    //设置绘制标志位
7         this.shuijing_D_Z_Flag=true;                                //设置绘制效果标志位
8         shuijingTh.setFlag(true);                                   //技能渐变线程开启
9    }
```

说明　当触摸界面中的游戏水晶时，界面会出现水晶技能效果，游戏中的所有怪物精灵都被击杀，同时水晶技能不能连续使用，此时开启水晶技能恢复工作线程。

（3）游戏水晶技能效果的开发，本游戏采取的做法是，当释放水晶技能时，通过线程不断改变技能图片的可见度，具体开发代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 ShuijingThread.java。

```
1    package com.bn.tag;                                          //声明包语句
2    public class ShuijingThread extends Thread{
3         GameView gv;                                            //GameView 对象的引用
4         boolean flag;                                           //线程是否执行标志位
5         boolean whileflag=true;                                 //游戏是否进行标志位
6         public ShuijingThread(GameView gv){                     //构造器
7              this.gv = gv;
8              flag = false;
9         }
10        public void run(){                                      //继承 Thread 类重写的 run 方法
11             while(whileflag){                                  //线程开启标志位
12                  if(flag){
13                       gv.alpha=gv.alpha-10;                    //可见度渐变
14                       if(gv.alpha<0){                          //减为 0 时
15                            gv.alpha=200;                       //恢复可见度
16                            gv.shuijing_D_Z_Flag=false;//技能绘制标志位置假
17                            this.setFlag(false);                //设置本线程工作标志位为 false
18                       }
19                  }
20                  try{
21                       Thread.sleep(50);                        //线程空转时间
22                  }
23                  catch(Exception e){
24                       e.printStackTrace();                     //打印异常信息
25        }}}
26        public void setFlag(boolean flag) {                     //设置工作线程标志位
27             this.flag = flag;
28        }
29        public void setwhileflag(boolean whileflag){            //设置线程开启标志位
30             this.whileflag=whileflag;
31   }}
```

- 第 3～5 行一些变量的声明。
- 第 6～9 行为类 ShuijingThread 的构造器。
- 第 12～19 行动态改变图片的可见度，当 Alpha 减为零时关闭工作进程。
- 第 26～31 行为线程开启标志位设置以及工作线程标志位设置函数。

（4）游戏水晶恢复过程的开发，这里主要实现纹理切割技术以及方向的开发，实现代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
private  Path makePathDash(float x6,float y6) {          //获得技能恢复情况下多边形路径
     Path p = new Path();                                 //新建路径
     float x1=SHUIJIAN_STARTX+SHUIJING_WEIGHT/2;          //顺时针得到多边形的顶点坐标
     float y1=SHUIJIAN_STARTY+SHUIJIAN_HEIGHT/2;
     float x2=SHUIJIAN_STARTX+SHUIJING_WEIGHT;
     float y2=SHUIJIAN_STARTY;
     float x3=SHUIJIAN_STARTX+SHUIJING_WEIGHT;
     float y3=SHUIJIAN_STARTY+SHUIJIAN_HEIGHT;
     float x4=SHUIJIAN_STARTX;
     float y4=SHUIJIAN_STARTY+SHUIJIAN_HEIGHT;
     float x5=SHUIJIAN_STARTX;
     float y5=SHUIJIAN_STARTY;
     if(y6==SHUIJIAN_STARTY){                             //六边形路径
```

```
14                p.moveTo(x1, y1);
15                p.lineTo(x6, y6);
16                p.lineTo(x2, y2);
17                p.lineTo(x3, y3);
18                p.lineTo(x4, y4);
19                p.lineTo(x5, y5);
20                p.lineTo(x1, y1);
21                return p;
22            }
23            else if(x6==SHUIJIAN_STARTX+SHUIJING_WEIGHT){   //五边形路径
24                p.moveTo(x1, y1);
25                p.lineTo(x6, y6);
26                p.lineTo(x3, y3);
27                p.lineTo(x4, y4);
28                p.lineTo(x5, y5);
29                p.lineTo(x1, y1);
30                return p;
31            }
32            else if(y6==SHUIJIAN_STARTY+SHUIJIAN_HEIGHT){   //四边形路径
33                p.moveTo(x1, y1);
34                p.lineTo(x6, y6);
35                p.lineTo(x4, y4);
36                p.lineTo(x5, y5);
37                p.lineTo(x1, y1);
38                return p;
39            }
40            else if(x6==SHUIJIAN_STARTX){                   //三角形路径
41            p.moveTo(x1, y1);
42                p.lineTo(x6, y6);
43            p.lineTo(x5, y5);
44                p.lineTo(x1, y1);
45                return p;
46            }
47            return null;
48        }
49        public void changeShuijingXY(int angle){            //动态得到路径中的点的坐标
50            if(angle>=0&&angle<90){                         //六边形情况下
51                    shuijing_change_x=(float) (SHUIJIAN_STARTX+SHUIJING_WEIGHT/2
52                            -SHUIJIAN_HEIGHT/2*Math.tan((45-angle)*Math.PI/180));
53                    shuijing_change_y=SHUIJIAN_STARTY;
54            }
55            else if(angle>=90&&angle<180){                      //五边形情况下
56                shuijing_change_x=SHUIJIAN_STARTX+SHUIJING_WEIGHT;
57                shuijing_change_y=(float) (SHUIJIAN_STARTY+SHUIJIAN_HEIGHT/2-
58                            SHUIJING_WEIGHT/2*Math.tan((135-angle)*Math.PI/180));
59            }
60            else if(angle>=180&&angle<270){                     //四边形情况下
61                shuijing_change_x=(float) (SHUIJIAN_STARTX+SHUIJING_WEIGHT/2+
62                            SHUIJIAN_HEIGHT/2*Math.tan((225-angle)*Math.PI/180));
63                shuijing_change_y=SHUIJIAN_STARTY+SHUIJIAN_HEIGHT;
```

```
64          }
65          else if(angle>=270&&angle<=360){                    //三角形情况下
66              shuijing_change_x=SHUIJIAN_STARTX;
67              shuijing_change_y=(float) (SHUIJIAN_STARTY+SHUIJIAN_HEIGHT/2+
68                          SHUIJING_WEIGHT/2*Math.tan((315-angle)*Math.PI/180));
69    }}
```

- 第 1～48 行得到水晶技能恢复不同状态下的多边形路径，用于后面的图片纹理切割。
- 第 49～69 行根据游戏水晶恢复的状态，即顺时针转动的角度得到多边形路径中变化的点的坐标。

（5）动态绘制游戏的恢复过程，实现代码如下。

代码位置：见随书光盘中源代码/第 15 章/TafangGameActivity/src/com/bn/tag 目录下的 GameView.java。

```
1    if(shuijing_Flag){                          //绘制水晶技能恢复过程的标志位
2        canvas.save();
3        canvas.clipPath(makePathDash(shuijing_change_x,shuijing_change_y));
                                                 //得到多边形路径下图片纹理
4        canvas.drawBitmap(shuijing_cover, SHUIJIAN_STARTX,SHUIJIAN_STARTY, null);
                                                 //绘制
5        canvas.restore();                       //恢复设置
6    }
```

说明　本模块要解决的问题是，有选择性的绘制加载图片的不同区域，解决方案是，设置并获得该区域下的多边形路径 Path，然后直接绘制即可完成要求，绘制完成之后，要恢复画布设置。

15.7 游戏的优化和改进

到此为止，塔防游戏——精灵塔防已经基本开发完成，也实现了最初设计的功能，但是，通过开发后的试玩、测试发现，游戏中仍然存在一些需要优化和改进的地方，下面列举需要改进的地方。

- 增设适当不同地图

本游戏开发之初只设想了单地图，但和多关卡地图游戏相比功能是完全一致的，唯一不同的是，游戏的可玩性降低了，读者可在本游戏基础上适当添加关卡地图。

- 适当增加不同种类的箭塔，增设箭塔升级模块

本游戏中只设计了两种箭塔，并且都不能升级，这样的情况下可以适当增加不同种类的箭塔，同时增加同种类箭塔的不同级别，这样就可以增加游戏的可玩性。

第 16 章　策略游戏——回到战国

策略游戏允许玩家自由控制、管理和使用游戏中的人物、资源，并通过较为自由的手段对抗敌人以达到游戏所要求的目标，取得游戏的胜利。“回到战国”便是一个典型的策略游戏，其取材于战国时期诸侯纷争年代，游戏中玩家通过掷骰子控制主人公的行走，并通过合理地管理自己的城池与将领，合理地分配粮草与兵力来达到统一中原的目标。

本章通过介绍策略类游戏——“回到战国”在 Android 平台下的设计与实现，使读者了解该类游戏的开发过程，同时也使读者掌握开发大型 RPG 游戏常使用的设计模式与实现思路。

16.1 游戏的背景及功能概述

本节将对该游戏的背景以及所要实现的基本功能进行简单介绍，使读者在正式进入开发之前对该游戏有一定的感性认识。

16.1.1　背景概述

早期的策略游戏玩法比较单一，游戏结果一般是统一国家或开拓殖民地，后来逐步发展成游戏方法比较固定的模拟类游戏，模拟类游戏通过模拟现实生活的世界或过去的世界，在游戏中充分利用自己的智慧来建立城池、招募将领，并通过努力管理城池和将领以达到游戏所设计的目标。

在“回到战国”游戏中，玩家可以合理地管理兵力、粮草等资源，并通过攻占敌方城池、开疆拓土来完成最终的统一中原大业。

16.1.2　功能简介

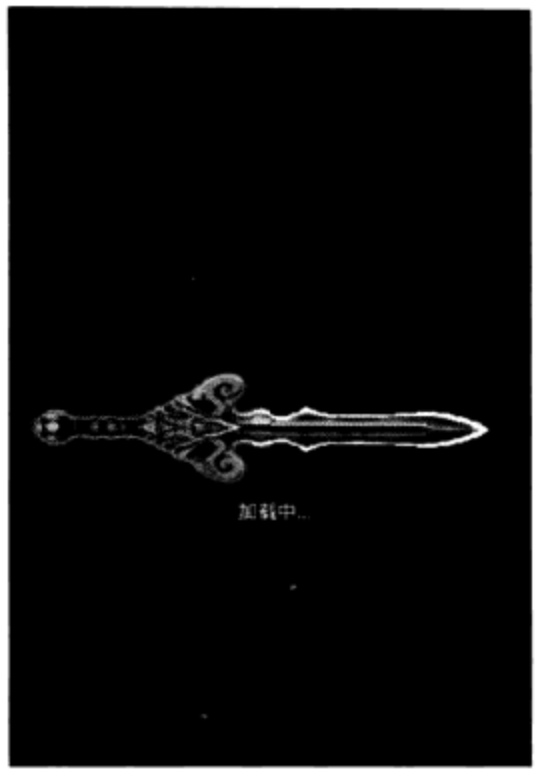
本节将对该游戏基本功能以及程序的运行效果进行简单介绍，操作步骤如下。

（1）运行该游戏，首先出现的是简单的声音询问界面，询问玩家是否需要开启游戏音效。

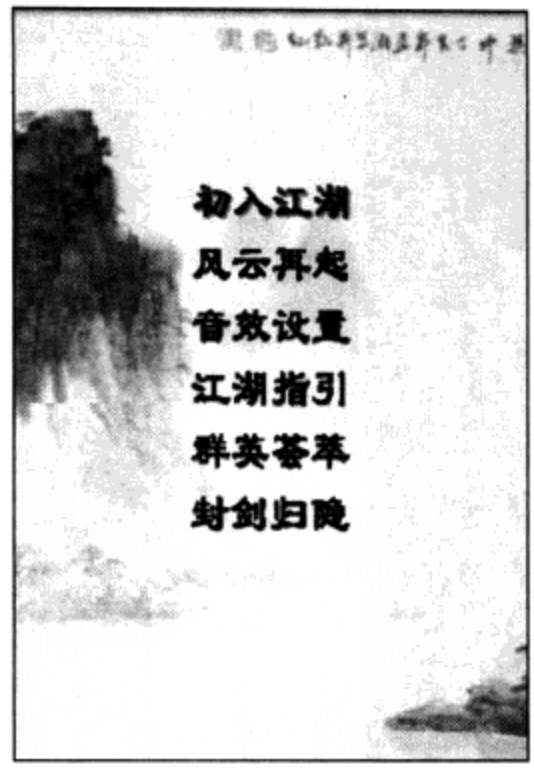
（2）之后将进入如图 16-1 所示加载界面，此时后台正在加载菜单界面的资源。

（3）加载完毕后将进入主菜单界面，并根据设置播放背景音乐，滚动背景图片，如图 16-2 所示。

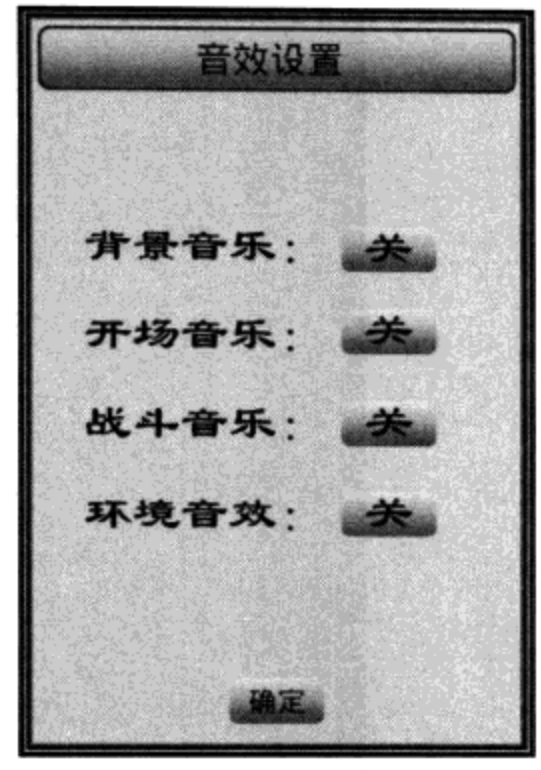
（4）玩家可以通过“音效设置”菜单进入音效设置界面，在音效设置界面中，可以对背景音乐、开场音乐、战斗音乐以及环境音效进行设置，如图 16-3 所示。



▲图 16-1　加载进度界面



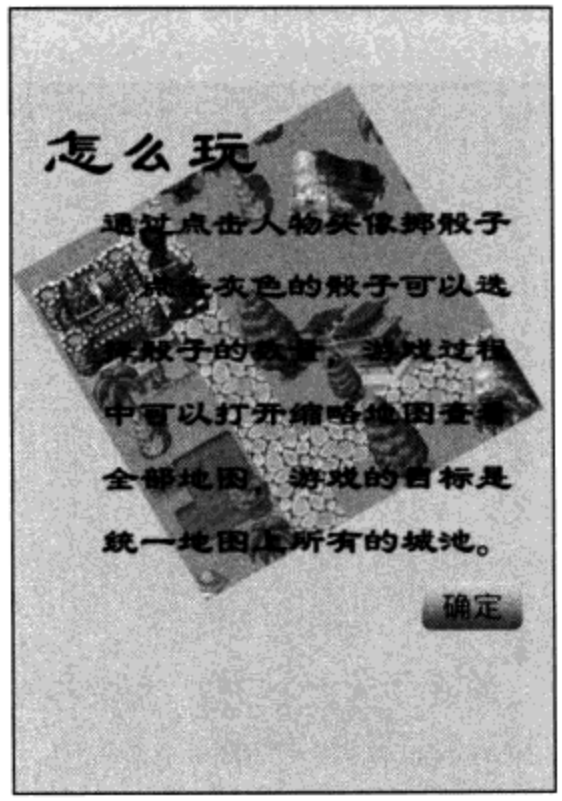
▲图 16-2　菜单界面



▲图 16-3　音效设置界面

（5）如果在菜单界面，玩家单击“江湖指引”菜单将进入帮助界面，如图 16-4 所示。

（6）如果在菜单界面单击“群英荟萃”菜单便会进入关于界面，显示游戏的相关信息，其效果如图 16-5 所示。



▲图 16-4　帮助界面

▲图 16-5　关于界面

（7）如果之前保存过游戏，则可通过“风云再起”菜单直接进入游戏界面，反之可通过“初入江湖”菜单开始新的游戏，首先进入竹简滚屏界面介绍背景之后才会进入游戏界面，如图 16-6 和图 16-7 所示。

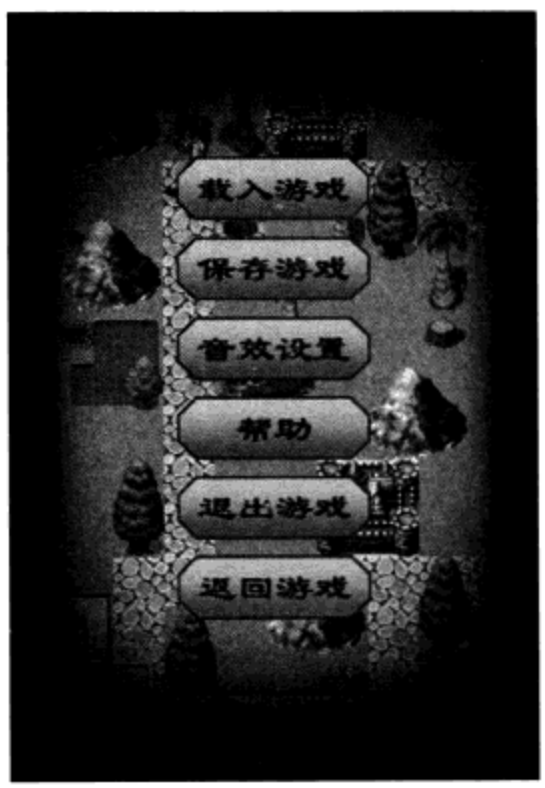
▲图 16-6　背景介绍界面

▲图 16-7　主游戏界面

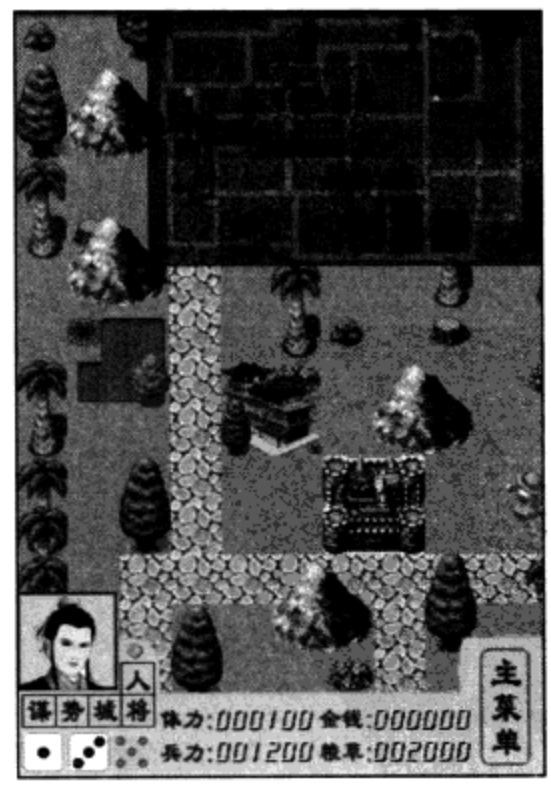
（8）在游戏界面中，可单击“主菜单”按钮进入主菜单，在主菜单界面可载入之前的存档，可以保存当前的存档，还可以查看帮助或退出游戏，效果如图 16-8 所示。

（9）游戏的玩法是通过单击人物头像掷骰子，然后根据所掷的点数来移动人物。

（10）单击人物头像右侧的小按钮便可以打开缩略地图，如图 16-9 所示。



▲图 16-8　游戏主菜单

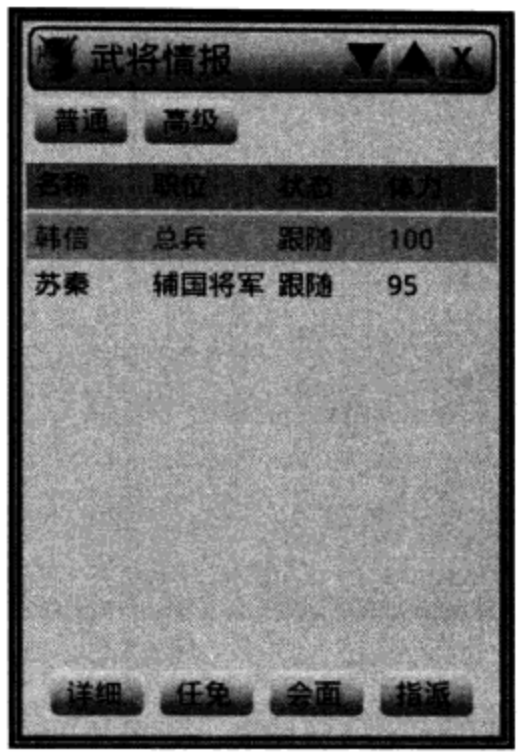


▲图 16-9　缩略地图

（11）单击“人”按钮可查看自身属性、技能等信息，如图 16-10 所示，可通过右上角的翻页按钮来控制页面的显示。

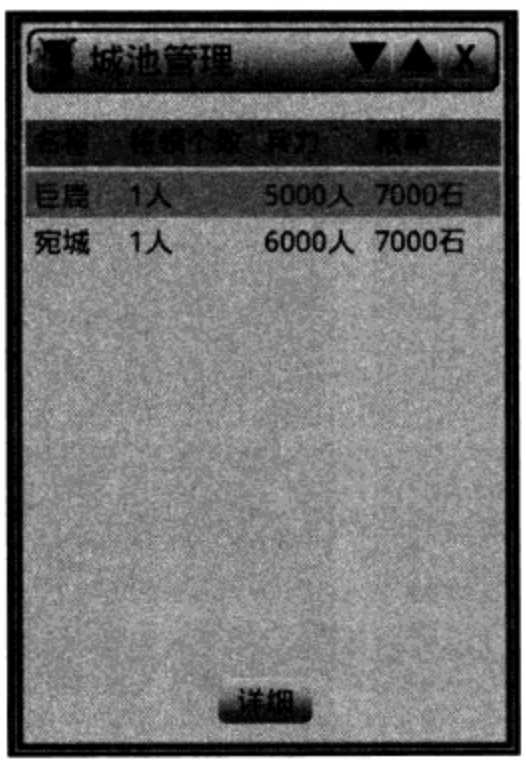
（12）单击“将”按钮可查看并管理玩家所拥有的所有将领，如图 16-11 所示，在界面还可以对将领执行查看详细信息、任免官职、会面、指派等操作。

▲图 16-10　人物属性界面

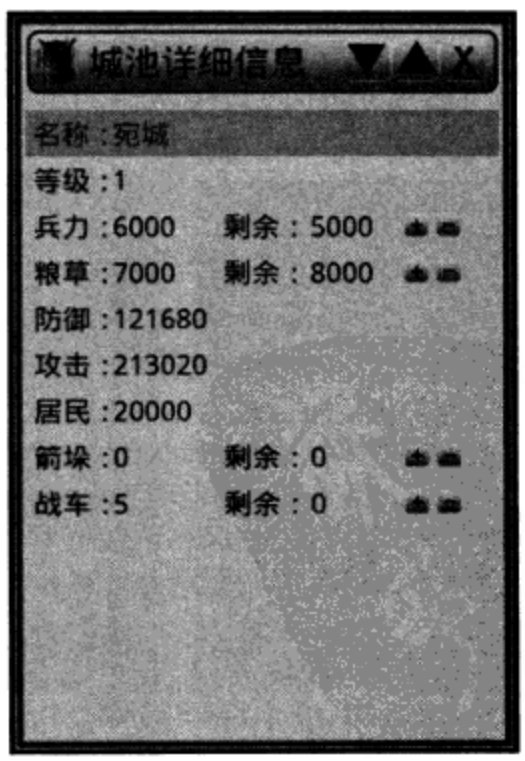


▲图 16-11　武将情报界面

（13）单击“城”按钮可查看英雄拥有的所有城池，如图 16-12 所示，单击“详细”按钮便可对选中的城池进行管理，例如，分配粮草，调拨兵力、箭垛、战车等，如图 16-13 所示。



▲图 16-12　城池查看界面

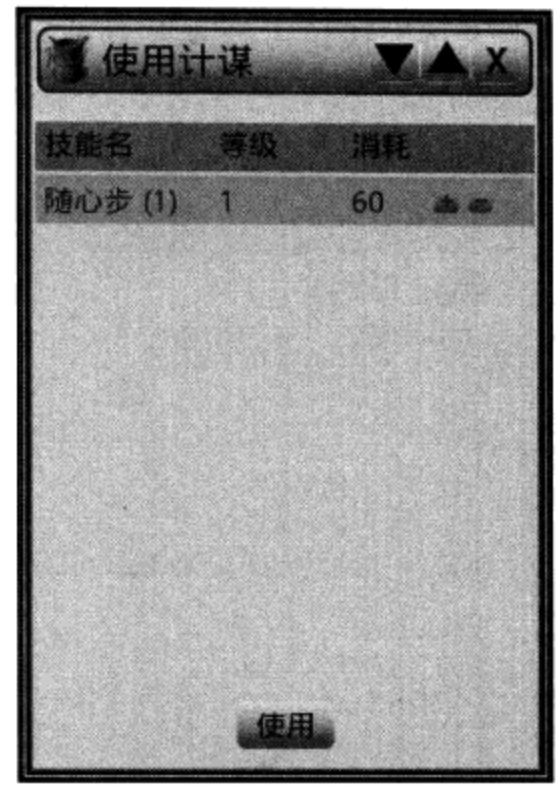


▲图 16-13　城池管理界面

（14）通过“势”按钮可以查看当前天下所有城池的基本信息，如图 16-14 所示。

（15）在“谋”界面中可以使用当前玩家所能使用的技能，效果如图 16-15 所示，选中需要使用的技能后，单击“使用”按钮即可。

▲图 16-14　天下局势界面

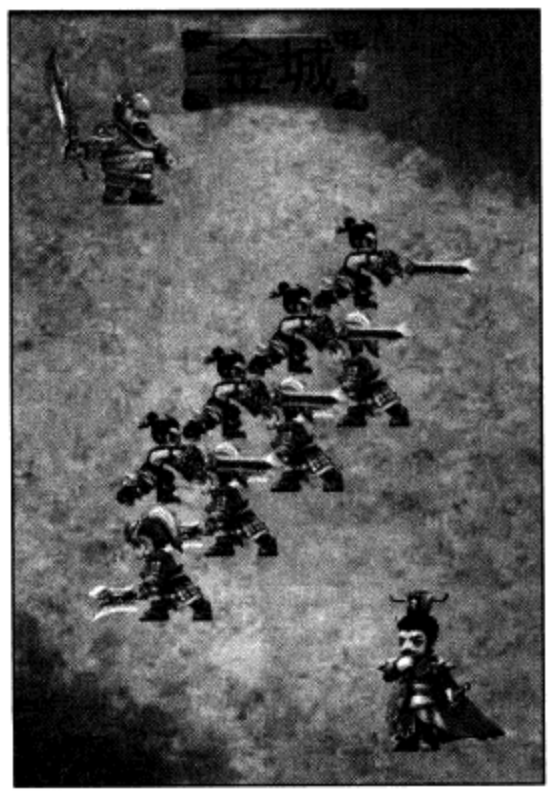


▲图 16-15　使用计谋界面

（16）当遇到敌方城池时，可选择缴税或者攻城，如果选择攻城，便会出现攻城动画显示战斗过程，战斗结束后显示战斗结果，如图 16-16 所示。

（17）在游戏过程中遇到麦田或稻田可以得到粮食，遇到村庄可以招兵买马，遇到矿山、渔场、森林可以得到金钱，遇到学堂可以为武将传授知识，遇到驿站可以恢复体力，遇到鲁班可以学习建造箭垛与战场，遇到马场可以训练武将马术。

（18）当遇到敌方城池却交不起过路费时，游戏失败，需重新开始游戏，效果如图 16-17 所示。当统一天下后，游戏胜利，效果如图 16-18 所示。



▲图 16-16　战斗画面

▲图 16-17　游戏失败

▲图 16-18　游戏胜利

16.2 游戏的策划及准备工作

了解了“回到战国”游戏的背景和功能之后，下面来进行游戏的策划和准备工作，确定游戏的开发方案。策划是游戏开发周期中必不可少的环节，这些策划包括故事情节、人物形象等内容的设计以及图片声音等资源的准备。

16.2.1 游戏的策划

游戏的策划是指对游戏中主要功能的实现方案进行确定的过程，大型游戏需要缜密的策划才可以开发，例如，呈现技术、目标平台等内容。

- 游戏情节

本游戏的故事背景定在战国乱世，游戏中的人名、城池名、对话等都将根据故事背景来确定。玩家的目标是控制英雄在群雄并起中通过不断扩张自己的势力最终统一天下。

- 地图设计器

本游戏的地图界面采用图元技术，由于本游戏中的地图元素不仅仅只有通过与否那么简单，因此开发该游戏时必须使用地图设计器，否则，在设计地图及地图元素时将很难进行。地图设计器可以使用第三方产品，也可以自己开发。

- 呈现技术

本游戏采用的游戏视角为正 90°2.5D 俯视视角。同时，由于地图的尺寸超过了手机屏幕的尺寸，还需要在游戏中实现滚屏功能。

- 目标平台

本游戏的目标平台为 Android 2.1。

- 操作方式

游戏的操作方式为触控操作，在游戏中单击英雄头像掷骰子，英雄会根据骰子点数移动相应的步数。游戏菜单及各种控制面板的弹出也是通过单击屏幕上的按钮来实现的。

16.2.2 Android 平台下游戏的准备工作

本小节将进行游戏开发之前的准备工作，主要是搜集和制作游戏中与代码无关的图片和声音资源，本游戏中用到的图片资源如表 16-1 所示。

表 16-1　　游戏中用到的图片资源及说明

图 片 名	大小（KB）	像素（w×h）	用　途	图 片 名	大小（KB）	像素（w×h）	用　途
about.jpg	9.41	320×480	关于图片	battle_enemy_general.png	13.6	320×100	敌方将领图片
battle_field.jpg	5.48	320×480	战场图片	dialog_back.png	3.28	320×120	对话框背景
maitian.png	13.2	62×62	麦田图片	digit_0～digit_9.png	2.54	10×13	数字图片
board.png	11.2	115×53	显示城池	enemy_1～enemy_4.png	6.63	155×80	敌方士兵图片

续表

图 片 名	大小（KB）	像素（w×h）	用　　途	图 片 名	大小（KB）	像素（w×h）	用　　途
caodi.png	1.83	31×31	草地图片	game_menu_options.png	8.47	120×240	菜单按钮图片
chengshi.png	4.57	62×62	城池图片	head_dynamic.png	3.20	60×60	英雄走路头像
close.png	1.40	60×29	关闭按钮	head_static.png	2.84	60×60	英雄静止头像
cunzhuang.png	5.67	62×62	村庄图片	hua1～hua2.png	4.7	31×31	花图片
daotian.png	10.2	62×62	稻田图片	hero_soldier_1～4.png	7.67	128×80	己方士兵图片
dark.png	33.0	320×480	遮盖图片	battle_hero.png	15.3	328×100	我方将领图片
dashboard.png	4.22	320×116	仪表盘	loading1～loading2.png	10.9	280×73	加载图片
dice.png	2.81	150×25	骰子图片	zhaoxianguan.png	5.57	62×31	招贤馆图片
gonglu.png	1.83	31×31	道路图片	machang.png	2.67	62×31	马场图片
help.png	185	313×312	帮助图片	huangcheng.png	8.3	62×62	皇城图片
hero.png	53.4	62×496	英雄图片	menu_options.png	14.3	600×50	菜单按钮文字
jing.png	2.5	31×31	井图片	menu_title.png	0.23	300×30	面板标题背景
kuangshan.png	9.47	62×62	矿山图片	menu_back.jpg	47.6	905×480	菜单界面背景
logo.png	2.45	30×30	面板图标	buttons.png	6.13	240×30	各种按钮图片
mozi.png	4.05	31×62	墨子图片	muzhuang.png	2.21	31×31	木桩图片
open.png	1.25	60×29	打开图片	panel_back.png	2.27	320×480	面板背景
scroll.jpg	13.9	280×360	卷轴图片	select_back.png	0.2	300×30	选中高亮图片
senlin.png	4.6	31×62	森林图片	sound_back.png	5.59	170×250	声音界面背景
sunzi.png	3.95	31×62	孙子图片	tiejiangpu.png	8.76	62×62	铁匠铺图片
wuguan.png	9.69	62×62	武馆图片	xuetang.png	5.59	62×31	学堂图片
yizhan.png	7.64	62×62	驿站图片	yuchang.png	8.91	62×62	渔场图片
luban.png	7.2	62×62	鲁班图片				

表 16-1 中的图片文件存放在项目文件夹 res/drawable-mdpi 目录下。下面来准备游戏中用到的声音资源，如表 16-2 所示。

表 16-2　　游戏中用到的声音资源及说明

声音文件	大小（KB）	格式	用　　途	声音文件	大小（KB）	格式	用　　途
startsound	26.8	mid	菜单界面背景音乐	backsound	9.12	mid	游戏进行时背景音乐
battle	23.6	ogg	战斗时播放	dingdong	10.6	ogg	遇到可遇物时播放

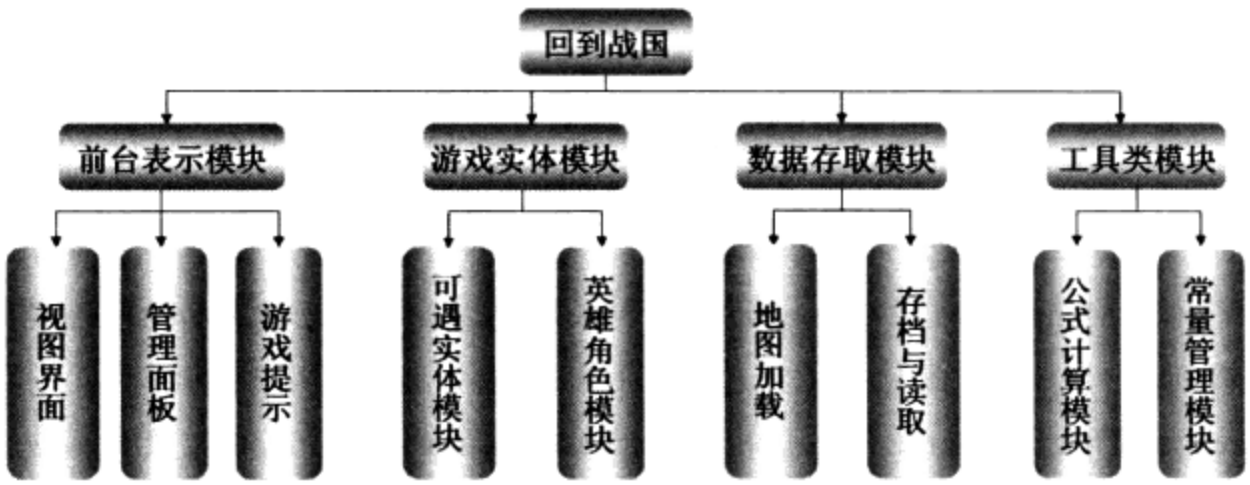
在项目文件夹的 res 目录下新建一个名为“raw”的目录，将表 16-2 所列的声音文件放置到新建的 raw 目录中。

16.3 游戏的架构

本小节将对游戏的架构进行简单的介绍，因为本游戏涉及的类较多，且这些类有些具有相同的特征或实现相同的功能，而有些则通过协作服务于同一个模块。设计好游戏的架构可以使开发的过程更加规范，少走弯路。

16.3.1　游戏的模块架构

在本节中将会对游戏功能模块的结构做一个简单的介绍，本游戏中各模块的结构如图 16-19 所示。



▲图 16-19　游戏的总架构

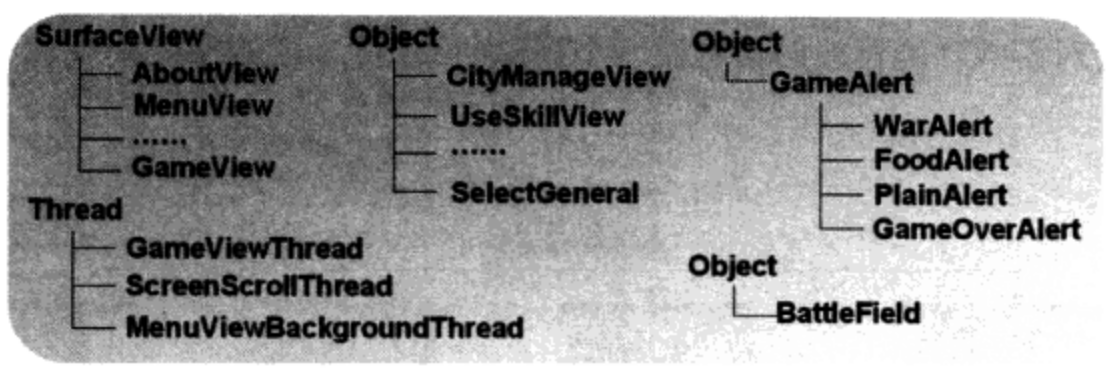
- 前台表示模块主要用于游戏画面的渲染，其中包括视图界面、管理面板和游戏提示 3 个模块。视图界面主要为游戏中出现的，诸如加载界面、菜单界面等视图；管理面板包括将领管理、城池管理等界面，主要是向玩家提供查看和修改游戏数据的面板；游戏提示负责在游戏进行中向玩家显示粮草危机、游戏失败等信息。
- 游戏实体模块主要用于后台游戏逻辑。包括英雄角色模块和可遇实体模块。英雄角色模块主要负责控制英雄的移动以及检测是否与地图中可遇实体发生碰撞，英雄角色模块还包括英雄技能模块，技能模块包括普通技能（如伐木、打渔等）和计谋（如随心步、回头是岸等）；可遇实体模块是地图上那些可以被英雄遇到并激发一系列动作的实体，如遇到稻田可以收获、遇到城池可以攻打等。
- 数据存取模块包括地图加载模块和游戏的存档与读取模块。地图加载主要是将从地图设计器获得的地图信息文件加载到游戏中并生成地图矩阵的过程；游戏的存档与读取模块用于将游戏的状态保存到文件并在需要的时候读取到游戏中。
- 工具类模块将自身的静态成员或方法提供给游戏中的其他类使用。本游戏中的工具类包括对常量进行统一管理的 ConstantUtil 类和用于存储游戏公式的 GameFormula 类，这里的游戏公式主要包括战斗的输赢、英雄体力增加等计算。

16.3.2 游戏各个类的简要介绍

前面对游戏的总体架构进行了介绍，在了解了游戏中的功能模块之后，本小节将对实现这些功能模块的具体类进行简单的说明。

1. 前台表示模块的类结构

前台表示模块的类结构如图 16-20 所示。



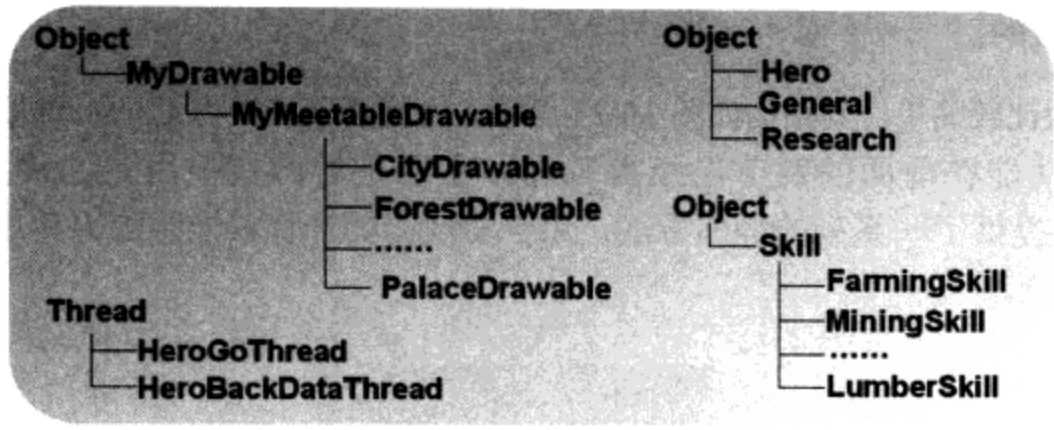
▲图 16-20 前台表示模块的类结构

- AboutView、MenuView 和 GameView 等一系列继承自 SurfaceView 的视图是组成游戏视图界面模块的主要部分，这些视图分别用来显示不同的内容，如菜单、帮助、滚动卷轴等。继承自 Thread 的 GameViewThread、ScreenScroll 和 MenuViewBackgroundThread 类负责修改上述视图的后台数据。BattleField 类的功能是在 GameView 界面绘制战斗动画。
- 游戏的管理面板模块中包括的类有 CityManageView、UseSkillView 和 SelectGeneral 等继承自 Object 的自定义类，这些类功能是在 GameView 中绘制不同的管理面板，如个人属性面板、将领面板、城池面板等。
- 游戏提示模块是由 GameAlert 的子类组成的，GameAlert 是继承自 Object 的自定义类，其子类的主要功能是用 GameView 显示游戏的提示信息，如有敌人偷袭城池、粮草发生危机或游戏结束时分别可以通过 WarAlert、FoodAlert 和 GameOverAlert 提示玩家进行相应操作。当只需要向玩家提示信息而不需要进行其他后续操作时，可以通过 PlainAlert 来实现。

2. 游戏实体模块的类结构

通过 16.3.1 节可以了解到，游戏的实体模块部分包括地图的可遇实体和英雄角色模块，这两个子模块的类结构如图 16-21 所示。

- 本游戏地图中要绘制的图元用 MyDrawable 对象表示，MyDrawable 对象分两种。第一种很简单，功能仅仅是进行图元绘制。第二种不但具有绘制图元的功能，还可以与英雄相遇，在相遇之后可以触发特定的操作。这种可遇的实体对象均继承自 MyMeetableDrawable 类，该类为继承自 MyDrawable 类的抽象类。游戏将会为地图中所有的可遇实体创建一个相应的 MyMeetableDrawable 对象，例如，英雄在地图上遇到森林时，ForestDrawable 对象会调用相关的方法来与玩家进行交互。

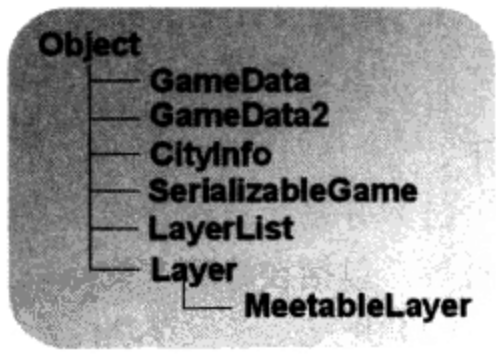


▲图 16-21　游戏实体模块的类结构

- 游戏中另外一个重要的实体对象就是 Hero 对象，Hero 对象中封装了英雄的属性，如所有城池、手下将领、金钱、粮草等。General 和 Research 类均为 Hero 类的辅助类，General 类为将领类，其中封装了防御力、武力和忠诚度等信息。Research 为科研类，当英雄建造箭垛或战车时，就会创建一个 Research 类来负责在后台进行建造。
- 继承自 Thread 类的 HeroGoThread 和 HeroBackDataThread 为 Hero 的附属线程，HeroGoThread 负责根据掷出的骰子点数将英雄移动指定的距离，同时进行可遇检查。HeroBackDataThread 负责修改英雄的个人数据，如定时消耗粮食、增加科研项目的进度以及随机产生攻打英雄城池事件，该线程的休眠时间很长，因为这些工作并不需要频繁地执行。
- 在游戏中，英雄具有一定的技能，这些技能均为继承自抽象类 Skill 的子类对象，Skill 类为继承自 Object 的自定义类，其中包含一些需要子类实现的抽象方法。在 Hero 类中会维护一个 Skill 对象列表来存储英雄学会的技能。

3. 数据存取模块的类结构

数据存取模块涉及的类及其结构如图 16-22 所示。

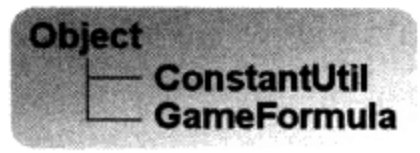


▲图 16-22　游戏存取模块的类结构

- GameData 和 GameData2 类实现的功能类似，都是加载存放在 Assert 中的地图信息文件，根据其内容创建游戏地图的 MyDrawable 二维数组。二者所不同的是，GameData 负责加载并生成下层平铺层的地图，而 GameData2 类负责加载并生成上层平铺层的地图，地图中的可遇实体对象位于上层平铺层。
- Layer 类为继承自 Object 的自定义类，该类用于维护一个图层的绘制矩阵（MyDrawable 矩阵）和不可通过矩阵。MeetableLayer 类继承自 Layer 类，其除了进行和 Layer 相同的工作外，还负责维护该图层的可遇矩阵，英雄移动到某个位置后通过该可遇矩阵来判断是否与地图中的可遇实体发生相遇。
- SerializableGame 类的主要功能是存储游戏状态和加载已存储的游戏，该类提供了用于实现这些功能的静态方法以及检测游戏是否存过档的方法。
- CityInfo 用于存储城池信息，其中封装了诸如城池名称、人口、所属国、守城将领、兵力等信息，游戏中创建 CityDrawable 对象时会从 CityInfo 中读取信息。

4. 游戏工具类的

游戏工具类模块比较简单，其结构如图 16-23 所示，主要涉及的类为 ConstantUtil 和 GameFormula。ConstantUtil 类的功能在前面的章节已经提及过，主要是对游戏中用到的常量进行统一管理。GameFormula 类中提供了一系列的静态方法，用于实现诸如计算城池防御力和攻击力、按一定算法增加英雄体力等功能。



▲图 16-23　游戏工具模块类结构

16.4 地图设计器的开发

在该类游戏的开发中，地图设计器的使用是必不可少的，又因为考虑到本游戏的特殊性，第三方地图设计器并不能满足需求，所以，需要自行开发特定的地图设计器，本节将对回到战国游戏的地图设计器进行介绍，该地图设计器会为后面的游戏开发提供地图信息文件。

16.4.1　底层地图设计器的开发

本游戏中的地图是两层的，底层全部是不可遇的实体，如道路、木桩以及花草等。而上层主要是可遇的建筑，如城池、村庄、驿站等。接下来先对底层地图设计器进行开发。

1. 地图设计器的开发设计思路

首先对该地图设计器的设计思路进行简单介绍，使读者增加对该地图设计器的理解，其步骤如下。

（1）开发元素设计模块，该模块负责设计该层地图中所用到的所有元素，例如，道路、草地、木桩等，包括设计其占位点以及不可通过点。

（2）开发元素保存功能，能够将设计好的元素列表保存到指定文件中，下次重新启动设计器时可以直接加载保存过的元素列表，无需从头重新设计。

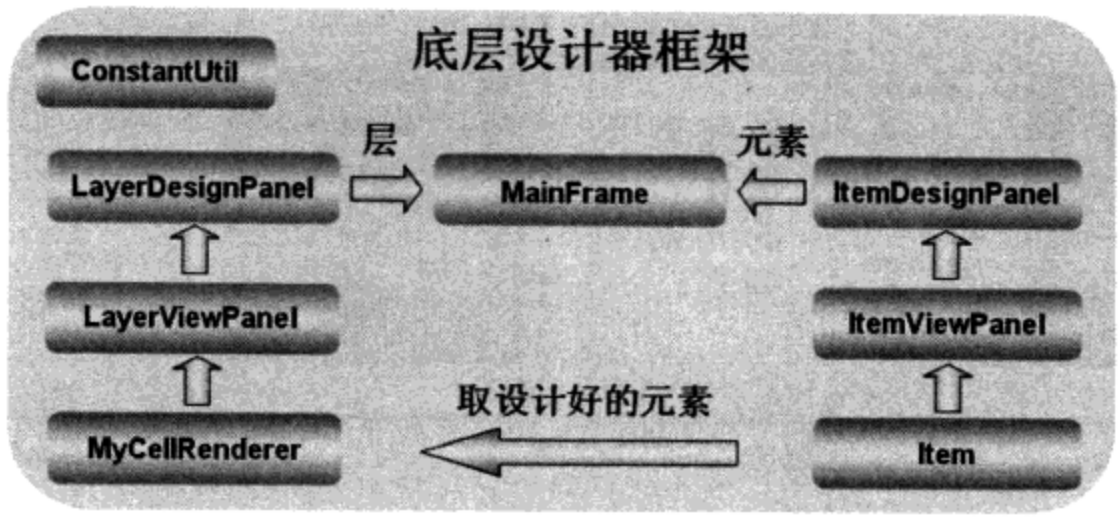
（3）开发层设计模块，该模块负责使用之前设计好的元素来设计地图的某一层，进入该模块前应该已经设计好所有元素并加载到元素列表中。

（4）开发层的保存功能，使用同样的技术将设计好的开发层信息保存到文件中，下次加载回来后可继续进行设计。

（5）地图信息文件的生成，产生包含该层所有信息的文件，该文件即为地图文件，将该文件存放到 Android 项目中的 assert 文件夹下，游戏便可以读取其信息。

2. 底层地图设计器的框架介绍

地图设计器的框架如图 16-24 所示。



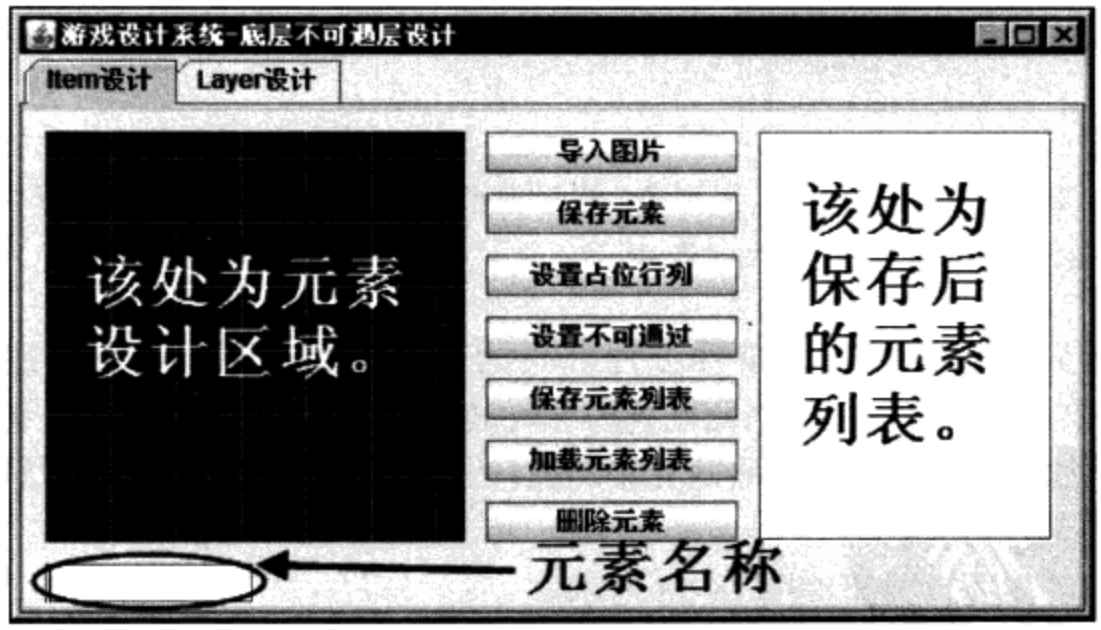
▲图 16-24　底层设计器框架图

该地图设计器分为两部分，一部分是底层地图中元素的设计，包括设计占位点以及不可通过点。而另一部分是层的设计，使用之前设计好的地图元素来设计底层地图。

3. 底层地图设计器的开发步骤

底层地图设计器的开发步骤如下。

（1）框架的搭建，首先创建一个普通的 Java 项目，然后搭建界面，效果如图 16-25 所示。



▲图 16-25　元素设计界面

（2）开发“导入图片”按钮的事件处理，当用户单击导入图片按钮，应该弹出文件选择窗口，在文件选择窗口中选择一张图片，并将该图片显示到元素设计区域的左下角。

（3）开发“设置占位行列”以及“设置不可通过”按钮的监听事件，此时导入一张图片，便可通过按钮设置图片的占位点以及不可通过点，效果如图 16-26 所示。

（4）“保存元素”按钮事件的开发。根据用户设置的占位点以及不可通过图片的信息创建一个 Item 对象，并将该对象添加到元素列表中，如图 16-27 所示。

▲图 16-26　村桩元素的设计

▲图 16-27　保存元素

（5）之后开发保存元素列表的代码。当用户单击“保存元素列表”按钮时将元素列表序列化即可，前提是 Item 类实现了 Externalizable 接口并实现了接口中的方法，保存的代码如下。

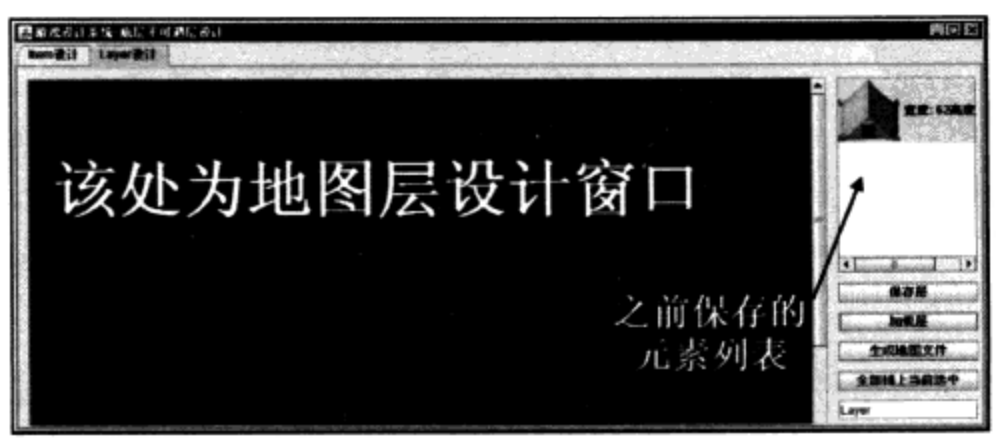
```
1    ArrayList<Item> alItem=new ArrayList<Item>();                    //成员变量，用于存放 Item
2    if(e.getSource()==jbSaveList){                                   //保存元素列表
3         status=5;                                                   //将界面的状态值置成 5
4         try{
5              FileOutputStream fout=new FileOutputStream("ItemList.wyf");//创建文件流
6              ObjectOutputStream oout=new ObjectOutputStream(fout);//序列化流
7              oout.writeObject(alItem);                              //将元素列表 alItem 序列化
8              oout.close();                                          //关闭相关流
9              fout.close();
10        }
11        catch(Exception ea){                                        //捕获异常
12             ea.printStackTrace();                                  //打印异常信息
13        }}
```

说明　该段代码为“保存元素列表”按钮监听方法中的处理代码，只需将元素列表 alItem 保存到 ItemList.wyf 文件中即可。

（6）“加载元素列表”按钮的事件处理代码的开发。当用户单击该按钮时，应该从文件 ItemList.wyf 中读取数据，直接恢复到元素列表 alItem，并将其显示到元素列表窗口中。

（7）元素模块的最后一步是开发删除元素功能，先得到元素列表中用户所选中的元素，然后将其从列表 alItem 中删除即可。

（8）搭建层设计界面，效果如图 16-28 所示。



▲图 16-28　地图层设计界面

（9）此时可以编写元素列表的事件监听方法。选中某个元素后，再单击右侧的设计窗口，便将选中的元素添加到后台的 Item 数组中，并在右侧的窗口中显示。

提示　地图的一层实际上是一个 Item 的二维数组，地图的设计实际上是对 Item 二维数组的填充。保存与加载地图也只是对该二维数组进行操作。

（10）之后开发保存与加载的处理代码。保存时将 Item 的二维数组 itemArray 序列化到指定文件中，而加载时从保存的文件中读取 itemArray，并恢复界面的显示。

（11）“生成地图文件”按钮的处理，实现代码如下，将下列代码插入到按钮监听方法中。

```
1    if(e.getSource()==jbCreate){                                  //单击生成地图文件按钮
2        try{                                                      //捕捉异常
3            FileOutputStream fout = null;                         //声明文件流
4            DataOutputStream dout = null;                         //声明数据流
5            fout = new FileOutputStream("maps.so");               //初始化文件流
6            dout = new DataOutputStream(fout);                    //初始化数据流
7            int totalBlocks=0;                                    //计数器
8            for(int i=0; i<40; i++){
9                for(int j=0; j<60; j++){                          //循环
10                   Item item = itemArray[i][j];
11                   if(item != null){
12                       totalBlocks++;                            //计算有多少个 Item 对象
13                   }
14               }
15           }
16           dout.writeInt(totalBlocks);                           //写入不空块的数量
17           for(int i=0; i<40; i++){
18               for(int j=0; j<60; j++){                          //对地图数组循环
19                   Item item = itemArray[i][j];                  //得到地图中该位置的元素
20                   if(item != null){
21                       int w = item.w;                           //元素的图片宽度
22                       int h = item.h;                           //元素的图片高度
23                       int col = item.col;                       //元素的地图列
24                       int row = item.row;                       //元素的地图行
25                       int pCol = item.pCol;                     //元素的占位列
26                       int pRow = item.pRow;                     //元素的占位行
27                       String leiMing = item.leiMing;            //类名
28                       int [][] notIn = item.notIn;              //不可通过
29                       int [][] keYu = item.keYu;                //可遇矩阵
30                       //计算图片下标
31                       int outBitmapInxex=0;
32                       if(leiMing.equals("Grass")){              //是草地时
33                           outBitmapInxex=0;
34                       }else if(leiMing.equals("XiaoHua1")){     //是小花 1 时
35                           outBitmapInxex=1;
36                       }else if(leiMing.equals("MuZhuang")){     //是木桩时
37                           outBitmapInxex=2;
38                       }else if(leiMing.equals("XiaoHua2")){     //小花 2 时
39                           outBitmapInxex=3;
40                       }else if(leiMing.equals("Road")){         //是道路时
41                           outBitmapInxex=4;
42                       }else if(leiMing.equals("Jing")){         //是井时
43                           outBitmapInxex=5;
44                       }
45                       dout.writeByte(outBitmapInxex);           //记录图片下标
46                       dout.writeByte(0);     //记录可遇标志，0-不可遇 底层都不可遇
47                       dout.writeByte(w);                        //图片宽度
```

```
48                                  dout.writeByte(h);                      //图片高度
49                                  dout.writeByte(col);                    //总列数
50                                  dout.writeByte(row);                    //总行数
51                                  dout.writeByte(pCol);                   //占位列
52                                  dout.writeByte(pRow);                   //占位行
53                                  int bktgCount=notIn.length;             //不可通过点的数量
54                                  dout.writeByte(bktgCount);              //写入不可通过点的数量
55                                  for(int k=0; k<bktgCount; k++){         //写入不可通过矩阵
56                                       dout.writeByte(notIn[k][0]);
57                                       dout.writeByte(notIn[k][1]);
58                                  }
59                             }
60                        }
61                   }
62                   dout.close();                                      //关闭数据流
63                   fout.close();                                      //关闭文件流
64              }catch(Exception ea){                                   //捕获异常
65                   ea.printStackTrace();                              //打印异常信息
66              }}
```

- 第 3～15 行初始化输出流并计算地图数组中元素的总个数。
- 第 17～61 行对地图数组进行循环，将数组中每个 item 元素的信息写入到输出流中，即保存到文件中，然后在第 62～63 行将相关流关闭。

提示 此处写入的顺序需要与之后在 Android 游戏中读取的顺序完全相同。需要将生成的 maps.so 文件复制到需要的项目中的 assets 文件夹下才可使用。

16.4.2 上层地图设计器的开发

上层地图设计器与底层基本相同，只有可遇点的设置不同，上层设计器的元素设计界面完成后效果如图 16-29 所示。

▲图 16-29 上层地图元素设计界面

开发步骤如下。

（1）基于下层的元素设计界面开发上层，只需向底层元素界面中增添一个“设置可遇行列”按钮，而当单击该按钮时，同样只需将可遇点存入数组。

（2）层设计界面比底层少一个“全部铺上当前选中”按钮，因为上层不会用同一元素填满整个

地图，下层一般需要先将整个地图铺满草地再进行设计。

（3）最后只需将保存的代码稍微修改一下，将生成的地图信息文件改为 mapsu.so，将计算图片下标的代码改为根据上层元素计算的下标，即将上一节代码中第 32～44 行用于判断 equals 的元素改成上层地图所拥有的元素名称。

到此，本游戏的地图设计器便开发完成，由于篇幅有限，所以，只对地图设计器的开发思路进行了介绍，并没有给出详细的代码，读者可以自行查看随书光盘中地图设计器的源代码。

16.5 Activity 和游戏工具类的开发

前面已经介绍了游戏的整体框架，接下来将正式进入游戏的开发，首先介绍的是主控制器 Activity 以及游戏工具相关类。

16.5.1　主控制类——HDZGActivity 的介绍

HDZGActivity 的工作主要是进行界面的切换与控制，根据收到 Handler 消息的不同做出不同的操作，其开发步骤如下。

（1）搭建 HDZGActivity 的框架，代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 HDZGActivity.java。

```
1    package wyf.ytl;
2    import android.app.Activity;                          //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘中的源代码
4    import android.view.WindowManager;                    //引入相关类
5    public class HDZGActivity extends Activity {          //继承自 Activity
6         LoadingView loadingView;                         //进度条界面的引用
7         MenuView menuView;                               //开始游戏的主菜单界面
8         GameView gameView;                               //游戏主界面
9         HelpView helpView;                               //帮助界面
10        SoundView soundView;                             //开始时选择是否开启声音的界面
11        SoundManageView soundManageView;                 //声音设置界面
12        ScreenRollView screenRoll;                       //竹简滚屏界面
13        AboutView aboutView;                             //关于界面
14        boolean isBackSound = true;                      //是否有背景声音
15        boolean isStartSound = true;                     //开场声音
16        boolean isBattleSound = true;                    //战斗声音
17        boolean isEnvironmentSound = true;               //环境音
18        View currentView = null;                         //当前的 View
19        Handler myHandler = new Handler(){               //用来更新 UI 线程中的控件
20             public void handleMessage(Message msg) {    //接收 Handler 消息
21                  switch(msg.what){                      //分支判断
22                  ……//该处省略了根据接收到消息的不同做不同操作的代码，后面将给出
23                  }
24             }
25        };
26         public void onCreate(Bundle savedInstanceState) {   //重写 onCreate 方法
27             super.onCreate(savedInstanceState);
28             requestWindowFeature(Window.FEATURE_NO_TITLE); //全屏
29             getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN ,
30                        WindowManager.LayoutParams.FLAG_FULLSCREEN);
31             soundView = new SoundView(this);             //初始化询问声音开启的界面
32             this.setContentView(soundView);              //先切换到简单的声音开关界面
```

```
33              GameData.resources=this.getResources();    //为 GameData 的静态变量赋值
34              GameData.initMapImage();                   //初始化地图图片
35              GameData.initMapData();                    //初始化地图数据
36              GameData2.resources=this.getResources();   //为 GameData2 静态变量赋值
37              GameData2.initBitmap();                    //初始化地图图片
38              GameData2.initMapData();                   //初始化地图数据
39              GameView.initBitmap(this.getResources());  //初始化 GameView 的图片
40              BattleField.initBitmap(this.getResources());//初始化 BattleField 的图片
41              ……//该处省略了其他界面的图片初始化方法，读者可自行查看随书光盘的源代码
42              SelectGeneral.initBitmap(this.getResources());//初始化SelectGeneral 的图片
43              TianXiaView.initBitmap(this.getResources());//初始化 TianXiaView 界面的图片
44              }
45              public void toLoadingView(int toViewID){            //切换到加载界面
46                   loadingView = new LoadingView(this,toViewID);  //初始化进度条
47                   this.currentView = loadingView;
48                   this.setContentView(loadingView);              //切换到进度条 View
49              }
50              public void toMenuView(){                  //切换到开始游戏的主菜单
51              if(this.menuView != null){                 //当 menuView 不为空时切换屏幕
52                   this.currentView = menuView;
53                   this.setContentView(menuView);
54              }else{                                     //当为空时打印并退出
55                   System.exit(0);                       //退出
56              }
57         }
58         public void toGameView(){                       //切换到游戏主界面
59              if(this.gameView != null){                 //当 gameView 不为空时
60                   this.currentView = gameView;
61                   this.setContentView(gameView);
62              }else{                                     //当为空时打印并退出
63                   System.exit(0);
64              }
65         }
66    public boolean onKeyDown(int keyCode, KeyEvent event) { //重写的按键监听方法
67         if(currentView != null){
68              currentView.onKeyDown(keyCode, event);     //调用各个 View 相应的方法即可
69         }
70         return super.onKeyDown(keyCode, event);
71    }}
```

- 第 6～13 行对游戏中的所有界面进行声明。
- 第 14～17 行为 4 种声音是否播放的标志位，包括背景音、开场音、战斗音以及环境音，每次需要播放声音时只需判断相应的标志位即可。
- 第 19～25 行为 Handler 消息接收对象，在 handleMessage 方法中，根据得到消息类型的不同执行不同的操作或调用不同的方法。
- 因为所有的图片只需加载一次，再次使用时无需再次加载，所以，各个界面对象中的图片资源与 resources 全部为静态的，在初始化 Activity 时便可对各个界面的图片资源进行初始化，如代码的第 33～43 行。
- 第 45～65 行的各个方法的作用是将当前的用户界面切换到相应界面。
- 第 66～71 行为重写的按键监听方法，当有按键被按下时，只需调用当前显示界面的 onKeyDown 方法，无需做出其他操作。

（2）对上面代码中第 22 行所在的方法 handleMessage 进行完善，其代码如下。

```
1    switch(msg.what){                                  //分支判断
```

```
2          case 1:                                  //收到 LoadingView 的消息,切换到 MenuView
3               if(loadingView != null){
4                    loadingView = null;             //释放 LoadingView
5               }
6               toMenuView();                        //切换到开始游戏时的主菜单
7               break;
8               ……//该处省略了收到其他界面消息后切换到相应界面的处理代码，读者可自行查看随书光盘的
                源代码
9          case 99:                                 //收到风云再起的事件响应
10              if(loadingView != null){
11                   loadingView = null;             //释放 LoadingView
12              }
13              currentView = null;                  //将表示当前用户界面置空
14              if(SerializableGame.check(HDZGActivity.this)){    //检测之前是否有过存档
15                   toLoadingView(101);                          //切换到加载界面
16                   new Thread(){                                //创建线程
17                        public void run(){
18                             try{
19                                  Thread.sleep(2000);           //睡眠 2 秒钟
20                             }catch(Exception e){              //捕获异常
21                                  e.printStackTrace();          //打印异常信息
22                             }
23                             Looper.prepare();
24                             if(gameView==null){               //当 gameView 为空时
25                                  gameView = new GameView(HDZGActivity.this);
                                                                  //初始化 GameView
26                             }
27                             SerializableGame.loadingGameStatus(gameView);
                                                                  //加载存档的信息
28                        }
29                   }.start();                                   //启动线程
30              }
31              break;
32         case 100:
33              if(gameView != null){                            //当 gameView 为空时
34                   HDZGActivity.this.setContentView(gameView);
                                                                  //初始化游戏界面 gameView
35              }
36              break;
37     }
```

- 第 2～7 行收到加载界面发来的类型为 1 的消息，说明菜单界面加载完毕，需要将当前用户界面切换到菜单界面。
- 第 8 行省略了其他类型消息的处理代码，其处理方式基本相同，都是先判断消息的类型，然后将用户界面切换到相应的用户界面。
- 第 9～30 行为收到风云再起事件的响应代码，先判断存档文件是否存在，因为只有存在时才可以加载存档，然后初始化游戏界面 GameView，并加载存档信息到 gameView 中。将耗时的初始化工作放到线程中是为了保证前台用户界面的流畅。

16.5.2 公式封装类——GameFormula 的介绍

GameFormula 类提供了一系列的静态方法，主要是游戏进行过程中用到的所有计算公式，例如，攻城输赢的判断、伐木所得金额的多少、体力增加的多少等，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 GameFormula.java。

```
1    package wyf.ytl;
2    public class GameFormula{
3         public static final int BASIC_STRENGTH_INCREMENT = 18;  //英雄的体力基础增加值
4         public static final int EXTRA_STRENGTH_INCREMENT = 2;//英雄的体力额外增加值，和等级有关
5         public static final int FOOD_DECREASE_EACH = 8; //每个人消耗的粮食
6         //计算并返回英雄应该增加的体力
7         public static int getHeroStrengthIncrement(Hero hero){//根据英雄的现状计算体力该增加多少
8              int level = hero.level;
9              int increment = BASIC_STRENGTH_INCREMENT+level*EXTRA_STRENGTH_INCREMENT;
10             if(hero.strength+increment >hero.maxStrength){
11                  increment = hero.maxStrength-hero.strength;
12             }
13             return increment;                          //返回计算结果
14        }
15        //计算并返回将领应该增加的体力，基本算法就是体力越低，恢复越快，等级越高，恢复越慢，但暂时
          //先不这样
16        public static int getGeneralStrengthIncrement(General general){
17             int strength = general.getStrength();      //获得将军体力
18             int increment = BASIC_STRENGTH_INCREMENT;  //计算出增量
19             if(strength + increment > general.maxStrength){     //如果不用加那么多就
                                                                   //已经完全恢复
20                  increment = general.maxStrength - strength;    //恢复满
21             }
22             return increment;                                   //返回结果
23        }
24        //计算并返回粮草的衰减数
25        public static int getFoodDecreaseEach(Hero hero){
26             int level = hero.level;                             //英雄的等级
27             return FOOD_DECREASE_EACH+level;                    //计算粮草的衰减数
28        }
29        //计算并返回技能的收益值，如该获得多少麦子等
30        public static int getSkillEearning(int proficiency,int basicEarning){
31             return basicEarning*(proficiency);                  //返回两个数的乘机
32        }
33        //计算并返回一座城池的防御力
34        public static int getCityDefence(CityDrawable city){
35             int defend =0;                                 //定义临时变量
36             General g = city.guardGeneral.get(0);          //得到城池中的主将
37             float gDefend = (g.defend*0.8f+g.intelligence*0.2f)/100.0f;
                                                              //计算
38             defend = (int)(city.baseDefend*city.army*(1+gDefend)) + city.warTower
               *100;
39             return defend;                                 //返回计算结果
40        }
41        //计算并返回一座城池的攻击力
42        public static int getCityAttack(CityDrawable city){
43             int attack = 0;                                //攻击力
44             General g = city.getGuardGeneral().get(0);     //获得守将对象
45             float gAttack = (g.power*0.7f+g.agility*0.2f+g.intelligence*0.1f)/100.
               0f;
46             attack = (int)(city.baseAttack*city.getArmy()*(1+gAttack)) + city.warTank
               *100;
```

```
47              return (int)attack;                                  //返回攻击力
48          }
49          //计算并返回英雄的攻击力
50          public static int getHeroAttack(Hero hero,General general){
51              int attack = 0;                                      //攻击力
52              float gAttack=(general.power*0.7f+general.agility*0.2f+general.intelligence
                *0.1f)/100f;
                                                                     //攻击加成
53              attack = (int)((hero.basicAttack*hero.armyWithMe)*(1+gAttack));
54              return attack;                                       //返回英雄的攻击力
55          }
56          //计算并返回英雄的防御力
57          public static int getHeroDefence(Hero hero,General general){
58              int defence = 0;                                     //防御力
59              float gDefend = (general.defend*0.8f+general.intelligence*0.2f)/100.0f;
                                                                     //英雄的防御加成
60              defence = (int)((hero.basicDefend*hero.armyWithMe)*(1+gDefend));
61              return defence;                                      //返回防御力
62          }
63      }
```

- 第 7～14 行为计算英雄应该增加体力多少的方法，增加的体力为英雄基础增加值加上英雄等级与额外增加值的乘积。
- 第 15～23 行计算将领的体力增加值，该算法比较简单，只将将领的基础增加值返回。
- 第 25～27 行的方法计算粮食的衰减数，为了演示的简单，只是返回每个消耗的粮食与英雄等级的和，即英雄等级越高消耗的粮食就会越多。
- 第 30～32 行计算技能的收益值，例如，收麦子、收稻田、捕鱼等，此处将这些算法简化并封装到同一方法中。在真实的游戏开发中，应该将其分为多个方法且应用更复杂、更合理的算法。
- 第 33～40 行的方法得到指定城池的总防御力，该防御力是根据镇守武将的信息以及城池中箭垛等防御设施的多少计算。第 41～47 行方法计算的是指定城池的总攻击力。
- 第 49～55 行为英雄攻击力的计算方法，是根据其武力、智力等信息综合计算的，第 57～62 行则根据英雄的信息计算基础防御力。

16.5.3　常量工具类 ConstantUtil 的介绍

前面已经对公式计算类进行了介绍，接下来将介绍另一个工具类——常量类，该类中封装了游戏中所用到的所有常量，将常量封装到一个常量类中的好处是便于管理和维护，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 ConstantUtil.java。

```
1   package wyf.ytl;
2   ……//该处省略了相关类的引入代码。读者可自行查看随书光盘中的源代码
3   public class ConstantUtil {                                  //常量类
4       //MenuView 界面中用到的常量
5       public static final int MENU_VIEW_SLEEP_SPAN = 200;
                                                                 //MenuView 界面刷帧线程睡眠时间
6       ……//该处省略了 MenuView 界面中部分常量的声明，读者可自行查看随书光盘的源代码
7       public static final int PICTUREHEIGHT = 480;
8       //LoadingView 界面中用到的常量
9       public static final int LOADING_VIEW_WORD_SIZE = 12;
```

```
                                                        //进度条界面中"加载中"3 个字的大小
10      ……//该处省略了 LoadingView 界面中部分常量的声明，读者可自行查看随书光盘的源代码
11      public static final int LOADING_VIEW_SLEEP_SPAN = 400; //进度条界面刷帧线程睡眠时间
12      //GameView 界面中用到的常量
13      public static final int GAME_VIEW_SLEEP_SPAN = 100;
                                                        //GameView 界面刷帧线程睡眠时间
14      ……//该处省略了 GameView 界面中部分常量的声明，读者可自行查看随书光盘的源代码
15      public static final int STRENGTH_COST_DECREMENT = 2;
                                                        //每次技能升级减小的体力消耗值
16      //英雄类中用到的常量
17      public static final int HERO_ANIMATION_SEGMENTS = 8;//英雄总共的动画段个数
18      ……//该处省略了英雄类中部分常量的声明，读者可自行查看随书光盘的源代码
19      public static final int RIGHT = 6;              //英雄的移动方向向右，也代表动画段
20      //技能类中用到的常量
21      public static final int LUMBER = 0;             //代表伐木技能，作为 HashMap 的键
22      ……//该处省略了技能类中部分常量的声明，读者可自行查看随书光盘的源代码
23      public static final int WU_ZHONG_SHENG_YOU = 6; //代表无中生有
24      public static final String[] HERO_TITLE = new String[]{ //英雄的官衔
25            "公爵","男爵"
26            ……//该处省略了部分官衔，读者可自行查看随书光盘的源代码
27      };
28      public static final String[] GENERAL_TITLE = new String[]{ //将领的职位
29            "总兵","护国将军"
30            ……//该处省略了部分职位，读者可自行查看随书光盘的源代码
31      };
32      public static final String[] RESEARCH_PROJECT = new String[]{//科研的项目
33            "战车","箭垛"
34      };
35      //敌方城池中的将领
36      public static final ArrayList<General> ENEMY_GENERAL = new ArrayList<General>();
37      static{
38            ENEMY_GENERAL.add(new General("赵奢", 5, 94, 80, 69, 69, 78, 80, 8));
39            ……//该处省略了部分将军的初始化代码，读者可自行查看随书光盘的源代码
40            ENEMY_GENERAL.add(new General("李斯", 1, 56, 52, 62, 59, 68, 68, 2));
41      }
42      //所有的城市信息
43      public static final ArrayList<CityInfo> CITY_INFO = new ArrayList<CityInfo>();
44      static{
45            Collections.shuffle(ENEMY_GENERAL);             //打乱顺序
46            CITY_INFO.add(new CityInfo("洛阳", 0, 5000, 5000, 2, 22, 15,
47                  80000, 5, 8, ENEMY_GENERAL.get(0),1));
48            ……//该处省略了部分城池的初始化代码，读者可自行查看随书光盘的源代码
49            CITY_INFO.add(new CityInfo("宁波", 7, 6000, 6000, 5, 22, 15,
50                  50000, 7, 12, ENEMY_GENERAL.get(21),22));
51      }
52      public static final String[] COUNTRY_NAME = new String[]{//所有的国家
53            "周国","秦国","自己"
54            ……//该处省略了部分国家的名称，读者可自行查看随书光盘的源代码
```

```
55          };
56          public static final int SCREEN_HEIGHT = 480;          //屏幕宽度
57          ……//该处省略了一部分其他常量的声明，读者可自行查看随书光盘的源代码
58          public static final int DIGIT_SPAN = 10;              //两个数码管数字之间的距离
59      }
```

- 第 4～23 行对各个类中所用到的常量进行定义。
- 第 24～28 行初始化英雄的所有官衔，在游戏过程中需要提升官衔时直接从该数组中获取即可。第 29～33 行存储的是将领的所有职位，当任免将领时，同样从本数组中获取数据。第 34～37 行定义了可科研的项目。
- 第 38～44 行初始化敌方城池中用到的所有将领，当初始化敌方城池时，会从中抽取相应的将领到敌方城池中。
- 第 45～54 行初始化地图中所有的城池信息，将其存储到 ArrayList 容器中，在初始化时使用了工具类中的 shuffle 方法，将所有将领打乱顺序，使得每次加载游戏时敌方城池有一定的随机性，提高游戏的可玩性。
- 第 55～60 行定义游戏中的所有国家，城池所属的国家都是该数组中的一种。在代码的第 62～64 行定义了一些共用的常量，例如，屏幕的宽度、高度等。

16.6 数据存取模块的开发

接下来对游戏的数据存取模块进行介绍，该模块主要负责对游戏数据的存储与读取，主要包括地图文件的加载以及游戏存档的恢复。

16.6.1　城池信息以及地图层信息的封装类

本节主要对城池信息的封装类 CityInfo、地图层信息的封装类 Layer 以及层的其他相关类进行介绍，使读者了解本游戏的数据管理方式。

1. 城池信息类 CityInfo

CityInfo 类主要封装了城池的静态信息，包括城池名称、所属国家、兵力、粮草、武将列表等，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 CityInfo.java。

```
1     package wyf.ytl;
2     import java.util.ArrayList;                                  //引入相关类
3     import static wyf.ytl.ConstantUtil.*;                        //引入ConstantUtil 中的静态成员
4     public class CityInfo {
5          String cityName;                                        //城池名称
6          ……//该处省略了部分成员变量，读者可自行查看随书光盘中的源代码
7          int warTower = 0;                                       //箭垛的个数用于增加防御力
8          ArrayList<General> guardGeneral = new ArrayList<General>();//武将列表
9          public CityInfo(String cityName, int country, int army, int food,
10                    int level, int baseAttack, int baseDefend, int citizen,
11                    int warTank, int warTower, General guardGeneral,int info) {//构造器
12               this.cityName = cityName;                          //得到城池名
```

```
13              ……//该处省略了部分成员变量的赋值，读者可自行查看随书光盘中的源代码
14              this.warTank = warTank;                          //得到战车个数
15              this.guardGeneral.add(guardGeneral);
16          }
17          public void setBackToInit(){                         //恢复到默认
18              for(int i=0; i<CITY_INFO.size(); i++){           //循环ConstantUtil中CITY_INFO
19                  CityInfo ci = CITY_INFO.get(i);              //得到每个CityInfo
20                  if(ci.info == this.info){                    //判断是否为当前这个
21                      this.cityName = ci.cityName;             //恢复城池名
22                      ……//该处省略了部分城池信息的恢复代码,读者可自行查看随书光盘中的源代码
23                      this.guardGeneral = ci.guardGeneral;  //恢复武将
24                      break;                                   //中断循环
25                  }}}}
```

- 第 5～8 行该类的成员变量，分别存储着城池的相关信息，例如，城池名、武将列表、箭垛个数等。
- 第 9～16 行为 CityInfo 的构造器，在构造器中将城池的信息进行初始化。
- 第 17～25 行的方法为恢复城池信息到默认值的方法，通过对常量类中的 CITY_INFO 进行循环，得到当前城池的信息，然后进行恢复。

2. 地图底层 Layer 的介绍

地图层 Layer 为底层地图封装类，除了存储表示地图信息的 MyDrawable 数组外，还提供了可遇矩阵的计算方法，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 Layer.java。

```
1   package wyf.ytl;
2   import java.io.Serializable;                                //引入相关类
3   import android.content.res.Resources;                       //引入相关类
4   public class Layer implements Serializable{                 //实现了 Serializable 接口
5       private static final long serialVersionUID = -9052906322948980410L;
                                                                //持久化版本序列号
6       private MyDrawable[][] mapMatrix;                       //表示地图的二维数组
7       public Layer(){}                                        //空构造器
8       public Layer(Resources resources){                      //构造器
9           this.mapMatrix = GameData.mapData;
10      }
11      public int[][] getNotIn(){                              //得到可通过矩阵
12          int[][] result = new int[40][60];                   //创建一个 int 行二维数组
13          for(int i=0; i<mapMatrix.length; i++){
14              for(int j=0; j<mapMatrix[i].length; j++){
15                  int x = mapMatrix[i][j].col - mapMatrix[i][j].refCol;
16                  int y = mapMatrix[i][j].row + mapMatrix[i][j].refRow;
                                                        //通过占位点及在大地图中的点计算
17                  int[][] notIn = mapMatrix[i][j].noThrough;
18                  for(int k=0; k<notIn.length; k++){   //对不可通过数组循环
19                      result[y-notIn[k][1]][x+notIn[k][0]] = 1; //不可通过点置 1
20                  }
21              }
22          }
23          return result;
24      }}
```

- 第 6 行为表示地图的二维数组，该二维数组中存放的是 MyDrawable 对象。
- 第 7～10 行为该类的两个构造器，因为需要将该类的对象进行序列化，所以，必须有第 7

行的空构造器，第 9 行的有参构造器中对该层地图信息进行初始化。

- 第 11～23 行方法的作用是计算该层地图的可通过矩阵，通过对地图数组进行循环，得到每个 MyDrawable 对象，然后得到 MyDrawable 所有的不可通过点，并记录到 int 型数组中。

3. 地图上层 MeetableLayer 的介绍

该类继承自 Layer，为地图上层的封装类，除了包含下层同样的信息外，还包含了地图的可遇矩阵，其代码如下。

代码位置：见随书光盘中源代码/第十六章/ HDZG/src/wyf/ytl 目录下的 MeetableLayer.java。

```
1    package wyf.ytl;
2    import java.io.Serializable;                          //引入相关类
3    import android.content.res.Resources;                 //引入相关类
4    public class MeetableLayer extends Layer implements Serializable{//实现 Serializable 接口
5         private static final long serialVersionUID = 1265133035026861860L;
                                                           //持久化版本序列号
6         private MyMeetableDrawable[][] mapMatrixMeetable;        //实际地图
7         private MyMeetableDrawable[][] mapMatrixForMeetable;     //可遇矩阵
8         public MeetableLayer(){}                                 //空构造器
9         public MeetableLayer(Resources resources) {              //构造器
10             super(resources);
11             this.mapMatrixMeetable = GameData2.mapData;         //得到地图信息
12             initMapMatrixForMeetable();                         //计算可遇矩阵
13        }
14        public void initMapMatrixForMeetable(){                  //计算可遇矩阵
15             mapMatrixForMeetable = new MyMeetableDrawable[40][60];
16             for(int i=0; i<mapMatrixMeetable.length; i++){
17                  for(int j=0; j<mapMatrixMeetable[i].length; j++){
18                       if(mapMatrixMeetable[i][j] != null){
19                            int x = mapMatrixMeetable[i][j].col - mapMatrixMeetable
                              [i][j]. refCol;
20                            int y = mapMatrixMeetable[i][j].row + mapMatrixMeetable
                              [i][j]. refRow;
21                            int[][] meetableMatrix = mapMatrixMeetable[i][j].
                              meetableMatrix;
22                            for(int k=0; k<meetableMatrix.length; k++){
23                                 mapMatrixForMeetable[y-meetableMatrix[k][1]]
                                   [x+meetableMatrix [k][0]]
24                                                = mapMatrixMeetable[i][j];
25                            }}}}}
26        public MyMeetableDrawable check(Hero hero){              //检测是否遇上
27             int col = hero.col;                                 //获取英雄的列数
28             int row = hero.row;                                 //获取英雄的行数
29             switch(hero.direction%4){                           //还是先按方向查看
30             case 0:                                             //向下
31                  if(mapMatrixForMeetable[row][col-1] != null){ //左边检测到了可遇物
32                       return mapMatrixForMeetable[row][col-1];
33                  }else if(mapMatrixForMeetable[row][col+1] != null){ //右边检测到了可遇物
```

```
34                          return mapMatrixForMeetable[row][col+1];
35                      }
36                      break;                                          //中断
37                  ……//该处省略了其他 3 个方向的计算，读者可自行查看随书光盘中的源代码
38                  }
39                  return null;                                    //如果没有检测到则返回 null 值
40          }}
```

- 第 7 行为上层地图的可遇矩阵，该二维数组中存储的并不是 0 和 1，而是可遇的实体，这样的好处是当检测可遇成功后，直接将遇到的实体返回即可。
- 第 9～13 行为该类的构造方法，在构造方法中得到上层地图信息并计算其可遇矩阵。
- 第 14～25 行为计算可遇矩阵的方法，同样是对地图数组进行循环，得到每个可遇的实体 MyMeetableDrawable，然后根据其占位点以及在地图中的位置计算可遇矩阵。
- 第 26～39 行检测是否遇上，判断英雄的上下左右 4 个方向是否有可遇的实体，如果有，则将该实体返回。

4. 地图层管理类 LayerList 的介绍

为了管理方便，本游戏开发了 LayerList 类来管理各个地图层，LayerList 包含了各个层的引用以及计算总不可通过矩阵的方法，实现代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 LayerList.java。

```
1    package wyf.ytl;
2    import java.io.Serializable;                                   //引入相关类
3    import java.util.ArrayList;                                    //引入相关类
4    import android.content.res.Resources;                          //引入相关类
5    public class LayerList implements Serializable{                //实现 Resources 接口
6         private static final long serialVersionUID = 3249868473800473202L;
                                                                    //持久化版本序列号
7         ArrayList<Layer> layers = new ArrayList<Layer>();         //存储 Layer 的容器
8         public LayerList(){}                                      //空构造器
9         public LayerList(Resources resources){                    //构造器
10             this.init(resources);
11        }
12        public void init(Resources resources){                    //初始化资源
13             Layer l = new Layer(resources);
14             layers.add(l);                                       //添加底层
15             MeetableLayer ml = new MeetableLayer(resources);
16             layers.add(ml);                                      //添加上层
17        }
18        public int[][] getTotalNotIn(){                           //得到总不可通过矩阵
19             int[][] result = new int[40][60];
20             for(Layer layer : layers){                           //对所有层进行循环
21                  int[][] tempNotIn = layer.getNotIn();
22                  for(int i=0; i<tempNotIn.length; i++){          //然后对不可通过矩阵循环
23                       for(int j=0; j<tempNotIn[i].length; j++){
24                            result[i][j] = result[i][j] | tempNotIn[i][j];
                                                                    //或运算，得到总不可通过点
25                       }}}
26             return result;
27        }}
```

- 第 7 行的容器存储的是各个层的引用。

- 第 9～11 行为该类的构造方法，在方法中调用初始化方法初始化相关资源，第 12～17 行的初始化方法中，初始化底层以及上层地图，并将其引用存储到 layers 容器中。
- 第 18～27 行为计算总不可通过矩阵的方法，需要对每层地图进行循环，得到每层地图的不可通过矩阵，然后对各个层的不可通过矩阵进行或运算，得到总不可通过矩阵。

16.6.2　数据存取相关类的介绍

本节将对与数据存取有关的 3 个类进行介绍，其中 GameData 类负责读取底层地图的数据，GameData2 类负责读取上层地图的数据，SerializableGame 类负责游戏存档的存储与读取。

1. GameData 类的介绍

GameData 类主要负责读取之前底层地图设计器所生成的地图文件中的地图信息，并将其分析成游戏中可用的信息，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 GameData.java。

```
1    package wyf.ytl;
2    import java.io.DataInputStream;                          //引入相关类
3    ……//该处省略了部分类的引入代码，读者可自行查看随书光盘中的源代码
4    import android.graphics.BitmapFactory;                   //引入相关类
5    public class GameData{                                   //各种MyDrawable 对象初始化
6         static Resources resources;                         //resources 的引用
7         static Bitmap  grassBitmap;                         //草地图片
8         ……//该处省略了部分图片引用的声明，读者可自行查看随书光盘中的源代码
9         static Bitmap jingBitmap;                           //井的图片
10        static Bitmap[] bitmaps;                            //图片数组
11        public static MyDrawable [][] mapData;              //地图矩阵
12        public static void initMapImage(){                  //初始化图片资源的方法
13             grassBitmap = BitmapFactory.decodeResource(resources, R.drawable.caodi);
14             ……//该处省略了部分图片的初始化代码，读者可自行查看随书光盘中的源代码
15             jingBitmap = BitmapFactory.decodeResource(resources, R.drawable.jing);
16             bitmaps=new Bitmap[]{                           //图片数组
17                  grassBitmap,                               //草地图片的引用
18                  xiaoHua1Bitmap,                            //小花 1 图片的引用
19                  muZhuangBitmap,                            //木桩图片的引用
20                  xiaoHua2Bitmap,                            //小花 2 图片的引用
21                  roadBitmap,                                //道路图片的引用
22                  jingBitmap                                 //井的图片引用
23             };
24        }
25        public static void initMapData(){                   //初始化地图数组
26             mapData = new MyDrawable [40][60];
27             try {
28                  InputStream in = resources.getAssets().open("maps.so");    //得到输入流
29                  DataInputStream din = new DataInputStream(in);//得到数据流
30                  int totalBlocks = din.readInt();                //读取实体总共的个数
31                  for(int i=0; i<totalBlocks; i++){
32                       int outBitmapInxex = din.readByte();
33                       int kyf=din.readByte();                    //可遇否，0 表示不可遇
34                       int w = din.readByte();                    //图元的宽度
35                       int h = din.readByte();                    //图元的高度
36                       int col = din.readByte();                  //总列数
37                       int row = din.readByte();                  //总行数
38                       int pCol = din.readByte();                 //占位列
39                       int pRow = din.readByte();                 //占位行
```

```
40                      int bktgCount=din.readByte();              //不可通过点的数量
41                      int[][] notIn=new int[bktgCount][2];
42                      for(int j=0; j<bktgCount; j++){            //读入不可通过点
43                           notIn[j][0] = din.readByte();
44                           notIn[j][1] = din.readByte();
45                      }
46                      mapData[row][col]=new MyDrawable(
47                                bitmaps[outBitmapInxex],         //图片
48                                ((kyf==0)?false:true),           //可遇否标志位
49                                w, h,col,row,pCol,pRow, notIn    //其他信息
50                      );
51                 }
52                 din.close();                                    //关闭输入流
53                 in.close();
54            } catch (IOException e) {                            //捕获异常
55                 e.printStackTrace();                            //打印异常信息
56            }}}
```

- 第 6～10 行为图片资源的引用，因为图片只需加载一次，所以，将其声明成静态成员，第 11 行为底层地图的二维矩阵，其他类需要绘制或使用底层地图时，必须从此处获得。
- 第 12～24 行为初始化图片资源的方法，在方法中先将所有图片进行初始化，然后再将图片的引用存放到数组中便于管理。
- 第 25～56 行为读取地图信息的方法，在方法中先创建 MyDrawable 的二维数组用来表示底层地图，然后通过 getAssets 方法打开输入流，并从文件中读取地图信息。此处读取的顺序必须与地图设计器设计地图时的保存顺序完全相同，存储的顺序见第 16.4.1 节中的代码。读取完信息后，根据这些信息创建 MyDrawable 对象并存储到地图数组中。

提示 需要提前将地图设计器生成的底层地图文件 maps.so 存放到项目目录中的 assets 文件夹下，如果 assets 文件夹不存在可手动创建一个。

GameData2 类的实现与 GameData 基本相同，只是其读取的是 mapsu.so 文件中的信息，创建的是 MyMeetableDrawable 类的子类对象，因本书篇幅有限，在此就不再赘述，读者可自行查阅随书光盘中本章案例的源代码。

2. SerializableGame 类的介绍

SerializableGame 类是用于保存与读取游戏存档的类，该类中包含存档、读取以及检测存档文件是否存在 3 个方法，其开发步骤如下。

（1）开发其代码框架如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 SerializableGame.java。

```
package wyf.ytl;
import java.io.InputStream;                                    //引入相关类
……//该处省略了部分类的引入代码，读者可自行查看随书光盘中的源代码
import android.media.MediaPlayer;                              //引入相关类
public class SerializableGame {
     public SerializableGame(){}                               //空构造器
     public static void saveGameStatus(GameView gameView){     //保存游戏的方法
          ……//该处省略了游戏保存的代码，将在后面给出
     }
```

```
10        public static void loadingGameStatus(GameView gameView){ //加载游戏的方法
11             ……//该处省略了游戏加载的代码，将在后面给出
12        }
13        public static boolean check(HDZGActivity h){          //检查文件是否存在
14             try{
15                  h.openFileInput("game.ytl");                //打开文件流
16             }catch(Exception e){                             //当捕获到异常时
17                  return false;                               //返回 false
18             }
19             return true;                                     //能正常打开时返回 true
20        }}
```

说明　其中 saveGameStatus 用于将当前游戏状态存档，而 loadingGameStatus 方法则是恢复游戏状态到存盘点。

（2）对 saveGameStatus 方法进行完善，其代码如下。

```
1     public static void saveGameStatus(GameView gameView){//保存游戏的方法
2          OutputStream out = null;                    //输出流
3          ObjectOutputStream  oout = null;            //声明 ObjectOutputStream 的引用
4          try{
5               out = gameView.getContext().openFileOutput("game.ytl", 0); //打开文件流
6               oout = new ObjectOutputStream(out);
7               oout.writeObject(gameView.hero);            //保存英雄对象
8               oout.writeInt(gameView.startRow);           //屏幕在大地图中的行数
9               oout.writeInt(gameView.startCol);           //屏幕在大地图中的列数
10              oout.writeObject(gameView.skillToLearn);    //将要学习的技能
11              oout.writeObject(gameView.allCityDrawable);     //存放所有敌方的城池
12              oout.writeObject(gameView.freeGeneral);     //自由将领,即不属于我方和敌方的
13              oout.close();
14              out.close();                                //关闭相关流
15         }catch(Exception e){                             //捕获异常
16              e.printStackTrace();                        //打印异常信息
17         }}
```

说明　该方法只是通过 ObjectOutputStream 将需要存储的游戏数据写入到文件 game.ytl 中，但前提是被序列化的各个类必须实现 Externalizable 或 Serializable 接口。实现 Externalizable 接口的类还须实现接口中的两个抽象方法来完成数据的存储和读取。

（3）接下来对 loadingGameStatus 方法进行介绍，其代码如下。

```
1     public static void loadingGameStatus(GameView gameView){      //加载游戏的方法
2          InputStream in = null;                          //声明 InputStream 的引用
3          ObjectInputStream oin = null;                   //声明 ObjectInputStream 引用
4          try{
5               in = gameView.getContext().openFileInput("game.ytl");   //得到输入流
6               oin = new ObjectInputStream(in);                        //初始化数据流
7               gameView.hero.ht.flag=false;                            //停止 ht 线程
8               gameView.hero.ht.isGameOn=false;
9               gameView.hero.ht.interrupt();
10              gameView.hero.hbdt.flag=false;                          //停止 hbdt 线程
11              gameView.hero.hbdt.interrupt();
12              if(gameView.activity.loadingView.process<90){
13                   gameView.activity.loadingView.process = 90;        //加载界面的进度条
14              }
15              gameView.hero = (Hero) oin.readObject();                //读取英雄对象
16              ……//该处省略了读取其他信息的代码，读者可自行查看随书光盘中的源代码
```

```
17              MyDrawable[][] mba=gameView.layerList.layers.get(1).getMapMatrix();
18              for(CityDrawable c : gameView.allCityDrawable){     //循环所有的城池
19                   int colT=c.col;
20                   ……//该处省略了恢复城池相关信息的代码，读者可自行查看随书光盘中的源代码
21                   c.bmpDialogButton=((CityDrawable)mba[rowT][colT]).bmpDialogButton;
22                   mba[rowT][colT]=c;
23              }
24              for(CityDrawable c : gameView.hero.cityList){       //循环
25                   int colT=c.col;
26                   ……//该处省略了英雄管理的城池信息的恢复代码，读者可自行查看随书光盘中的源代码
27                   c.bmpDialogButton=((CityDrawable)mba[rowT][colT]).bmpDialogButton;
28                   mba[rowT][colT]=c;
29              }
30              ((MeetableLayer)gameView.layerList.layers.get(1)).initMapMatrixFor-
                Meetable();
31              gameView.hero.father = gameView;                    //恢复英雄相关信息
32              gameView.hero.initAnimationSegment(GameView.heroAnimationSegments);
                                                                    //为英雄动画段列表
33              gameView.hero.startAnimation();                     //启动英雄动画
34              gameView.hero.hgt = new HeroGoThread(gameView,gameView.hero);
35              gameView.drawThread.setIsViewOn(true);              //设置标志位
36              gameView.hero.hbdt=new HeroBackDataThread(gameView.hero);
37              gameView.hero.hbdt.start();                         //启动线程
38              if(!gameView.mMediaPlayer.isPlaying()){             //当音乐没有播放时
39                   gameView.mMediaPlayer = MediaPlayer.create(gameView.activity,
                     R.raw.backsound);
40                   gameView.mMediaPlayer.setLooping(true);        //设置循环播放
41                   if(gameView.activity.isBackSound){
42                   gameView.mMediaPlayer.start();                 //播放声音
43                   }}
44              gameView.activity.loadingView.process = 101;        //进度条
45              gameView.activity.myHandler.sendEmptyMessage(100); //发送 Handler 消息
46              oin.close();
47              in.close();                                         //关闭流
48         }
49         catch(Exception e){                                      //捕获异常
50              e.printStackTrace();                                //打印异常信息
51         }}
```

- 第 5～6 行初始化相关输入流。第 7～11 行停止所有相关线程。
- 第 15～18 行从存档文件中读取之前存储的游戏信息，并将其恢复到游戏中。
- 第 19～26 行恢复所有敌方城池的信息，第 27～34 行则是恢复英雄所统治的城池的信息。
- 第 35～53 行恢复游戏相关的线程，并启动应该启动的相关线程。当所有信息恢复完成后，向 Activity 发送 Handler 消息通知 Activity 切换用户界面。

16.7 英雄角色模块的开发

在本章的 16.3.1 节曾对游戏的实体模块进行了简单的介绍，本小节将对实体模块之一——英雄角色模块的开发进行介绍，该模块涉及的类有 Hero、HeroGoThread、HeroBackDataThread 以及 General 和 Research。

16.7.1 Hero 类的代码框架

Hero 类是英雄角色实体部分的主要类，本小节先来简单地介绍 Hero 类的代码框架，实现步骤

如下。

（1）Hero 类的成员变量的声明，Hero 类中的成员变量大多用于记录游戏的状态，如当前金钱、兵力等，同时也有与绘制英雄相关的成员变量。Hero 类的成员变量部分的代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 Hero.java。

```
1    package wyf.ytl;                                    //声明包语句
2    import static wyf.ytl.ConstantUtil.*;               //引入相关类
3    ……//此处省略部分引入相关类的代码，读者可自行查阅随书光盘
4    import android.graphics.Canvas;                     //引入相关类
5    public class Hero implements Externalizable{
6         public static ArrayList<General> MY_GENERAL = new ArrayList<General>();
                                                         //自由将领列表
7         private static final long serialVersionUID = -5433306195939766058L;
                                                         //持久化版本序列号
8         GameView father;                               //GameView 引用
9         private String name = "段世雄";
10        int strength = 100;                            //英雄体力，
11        int maxStrength = 100;                         //体力上限
12        int level = 1;                                 //英雄等级
13        int direction = -1;                            //英雄的移动方向
14        int currentFrame = 0;                          //当前英雄的动画段的当前动画帧，从零开始
15        int food = 10000;                              //随身携带的粮草，不包括每个城池中的粮食
16        int totalMoney=6000;                           //总金钱
17        int armyWithMe = 6000;                         //跟随自己的军队
18        int totalArmy;                                 //总军队，包括跟随自己的和守城在家的
19        int col;                                       //英雄的定位点在大地图中的列
20        int row;                                       //英雄的定位点在大地图中的行
21        int x;                                         //英雄定位点的 x 坐标
22        int y;                                 //英雄定位点的 y 坐标
23        int width;                             //英雄的宽度，在 initAnimationSegment 方法中初始化
24        int height;                            //英雄的高度，在 initAnimationSegment 方法中初始化
25        int title = 0;                         //英雄官衔
26        int warTank = 0;                       //随身携带的战车
27        int warTower = 0;                      //随身携带的箭垛
28        int strengthGrowSpan = 1;
29        int basicAttack = 18;                  //基础攻击力
30        int basicDefend = 18;                  //基础防御力
31        ArrayList<Bitmap []> animationSegment = new ArrayList<Bitmap []>();
                                                 //存放英雄所有的动画段
32        ArrayList<CityDrawable> cityList = new ArrayList<CityDrawable>();
                                                 //存放英雄拥有的城池
33        ArrayList<General> generalList = new ArrayList<General>();
                                                 //存放跟随英雄的将领
34        ArrayList<Research> researchList = new ArrayList<Research>();
                                                 //存放现有的科研项目
35        HashMap<Integer,Skill> heroSkill = new HashMap<Integer,Skill>();
                                                 //存放英雄的所有技能
36        HeroThread ht;                         //负责英雄动画换帧的线程
37        HeroGoThread hgt;                      //负责英雄走路的线程
38        HeroBackDataThread hbdt;               //负责后台数据更改的线程
39        class HeroThread extends Thread{//内部线程类：负责定时更改英雄的动画帧}
40        ……//此处省略 Hero 类的构造器和成员方法，将在随后的步骤中介绍
41   }
```

- 代码第 6 行声明了存放自由将领的列表，游戏中英雄在招贤馆有可能获得该列表中的将领。
- 代码 13 行声明了记录英雄移动方向的成员变量，取值 4、5、6、7 分别代表下、左、右、上，英雄的方向同时也代表当前英雄的动画段号。

- 代码第 19～22 行分别声明了英雄定位点所在地图中的行和列以及该点在地图中的坐标。本游戏中英雄的像素大小为 31×62，英雄的定位点为靠下 31×31 部分的中心。
- 代码第 25 行声明了代表英雄官衔的成员变量，当英雄遇到皇城时，有可能通过进贡金钱给天子获得官衔。代码第 28 行声明的成员变量用于计算游戏体力增加的点数，计算英雄体力增加的方法是用英雄走过的地图格子数乘以该成员变量的值。
- 代码第 29 和 30 行声明的成员变量将用于计算英雄当前的攻击力和防御力，计算时这两个成员变量将会和当前跟随的兵力做乘积。
- 代码第 32～35 行声明了用于存放英雄拥有城池、将领、科研项目以及技能的集合对象，General 和 Research 类的开发比较简单，只包括一些必要的成员变量和对应的 get 和 set 方法，在此将不再列出其详细代码。Skill 类的开发将在后面的内容中提到。
- 代码第 36 行声明了负责为英雄动画换帧的 HeroThread 对象，代码第 39 行为 HeroThread 类的定义代码。代码第 37 行声明了 HeroGoThread，该对象负责根据英雄掷出的骰子移动英雄。代码第 38 行声明了 HeroBackDataThread 对象，该对象负责后台数据更改的线程，如减少粮草、推进科研进度、随机选择城市被偷袭等。

（2）Hero 类成员方法的框架，在前面的步骤介绍了 Hero 类的成员变量，下面来介绍 Hero 类的构造器和成员方法的代码框架，步骤（1）中代码第 40 行省略的代码如下。

```
1    public Hero(){}//空构造器
2    public Hero(GameView father,int col,int row){//构造器：初始化成员变量}
3    public void writeExternal(ObjectOutput out) throws IOException {//实现 Externalizable
                                                                    //接口的方法}
4    public void readExternal(ObjectInput in) throws IOException,ClassNotFoundException
{//实现接口方法}
5    public void initAnimationSegment(Bitmap [][] segments){   //方法：初始化动画段列表}
6    public void addAnimationSegment(Bitmap [] segment){//方法：向动画段列表中添加动画段}
7    public void setAnimationDirection(int direction){//方法：设置方向，同时也将设置动画段}
8    public void startAnimation(){   //方法：开始换帧动画}
9    public void pauseAnimation(){   //方法：暂停动画}
10   public void destroyAnimation(){//方法：销毁动画及其换帧线程}
11   public void drawSelf(Canvas canvas,int startRow,int startCol,int offsetX,int offsetY){
                                                  //方法：绘制自己}
12   public int [] throwDice(int diceNumber){  //方法：掷骰子}
13   public void nextFrame(){                  //方法：在当前动画段执行换帧操作}
14   public void startToGo(int steps){//方法：激活英雄的走路线程，传入格子数使英雄开始移动}
15   public void growStrength(int steps){      //方法：根据英雄移动的距离增加英雄的体力}
17   public void initHeroSkills(){             //方法：初始化英雄的技能}
18   ……//此处省略 Hero 类成员变量的 set 和 get 方法，读者可自行查阅随书光盘
```

- 代码第 1 行为 Hero 类开发了一个无参数、空实现的构造器，该构造器用于将 Hero 对象进行序列化。代码第 3～4 行是对 Externalizable 接口的方法 writeExternal 和 readExternal 的实现。关于游戏的存档和读取已在 16.6 节进行过介绍，在此将不再赘述。
- 代码第 5～10 行和 13 行为对英雄的动画帧进行的一系列操作方法，包括初始化动画段、向动画段列表中添加动画段、开始动画、暂停动画等。
- 代码第 12 行为 throwDice 方法，该方法接收骰子的个数作为参数，然后为每个骰子随机产生一个点数，最后返回一个记录了每个骰子点数的数组。
- 代码第 14 行为 startToGo 方法，调用该方法将激活 HeroGoThread，然后 HeroGoThread 将

根据传入的地图格子数将英雄移动相应的距离。

- 代码第 18 行省略了 Hero 类一些成员变量的 get 和 set 方法。这些方法的实现方式比较简单，由于篇幅有限在此将不列出这些方法。

16.7.2　英雄运动线程——HeroGoThread 类的开发

HeroGoThread 是 Hero 类的重要辅助线程，其功能主要包括根据骰子点数移动英雄，并对是否需要转弯、是否需要滚屏、是否遇到可遇实体进行检测。HeroGoThread 的 run 方法代码框架如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 HeroGoThread.java。

```
1    public void run(){                                          //线程执行方法
2         while(flag){
3              while(isMoving){
4                   for(int i=0;i<steps;i++){                    //对每一格子进行无级移动
5                        //步骤(1)求出本次移动的目的点
6                        //步骤(2)无级从一个格子走到另一个格子
7                   }
8                   //步骤(3)检查是否可遇
9              }
10             try{Thread.sleep(waitSpan);}                     //线程空转等待
11             catch(Exception e){e.printStackTrace();}         //捕获并打印异常
12        }}
```

在游戏中，通过设置代码第 3 行用到的 seMoving 变量为 true 来激活 HeroGoThread，代码第 4 行使用到的 steps 成员变量代表骰子点数，即要移动的地图格子数。在该线程 run 方法的每次循环中，都会按照上述 3 个步骤来进行，下面将对这 3 个步骤进行简单的介绍。

（1）首先根据英雄的方向确定目标点地图格子的行和列，其代码如下所列。

```
1    int moves = TILE_SIZE/HERO_MOVING_SPAN;                     //求出这个格子需要几个小步来完成
2    int hCol = hero.col;                                        //英雄当前在大地图中的列
3    int hRow = hero.row;                                        //英雄当前在大地图中的行
4    int destCol=hCol;                                           //目标格子列数
5    int destRow=hRow;                                           //目标格子行数
6    switch(hero.direction%4){                                   //先求出目的点的格子行和列
7    case 0:                                                     //向下
8         destRow = hRow+1;
9         hero.setAnimationDirection(DOWN);                      //将英雄的方向和动画段设置为动态向下
10        break;
11   ……//此处省略方向为静(动)态向左和静(动)态向右时的处理代码，读者可自行查阅随书光盘
12   case 3:                                                     //向上
13        destRow = hRow-1;
14        hero.setAnimationDirection(UP);                        //将英雄的方向和动画段设置为动态向上
15        break;
16   }
17   int destX=destCol*TILE_SIZE+TILE_SIZE/2+1;                  //目的点 x 坐标，已经转换成中心的了
18   int destY=destRow*TILE_SIZE+TILE_SIZE/2+1;                  //目的点 y 坐标，已经转换成中心点的了
19   int hx = hero.x;                                            //获得英雄的当前位置
20   int hy = hero.y;
```

- 代码第 1 行首先计算出移动一个格子的距离需要几次移动来完成，由于英雄移动的步进并不能整除格子间距离（即图元大小），所以，在步骤（2）的最后还需要对英雄的位置进行修正。
- 代码第 6～16 行通过对英雄当前方向进行判断，计算出目的点的地图格行列数。游戏中英雄可取的方向有 8 种，静止和动态各有 4 个方向，由于动态方向和静态方向之间正好相差 4，所以

进行模 4 操作来减少判断次数。

- 代码第 17～18 行分别计算获得目的点的 x、y 坐标和英雄的当前位置坐标。

（2）确定了目的点的位置，下面就需要对英雄进行无级移动了，移动的代码如下。

```
1    for(int j=0;j<moves;j++){
2         try{Thread.sleep(HERO_MOVING_SLEEP_SPAN);}        //先休眠一下
3         catch(Exception e){e.printStackTrace();}
4         if(hx<destX){                                     //判断英雄坐标和目的点的大小
5              hero.x = hx+j*HERO_MOVING_SPAN;              //修改英雄的 X 坐标
6              hero.col = hero.x/TILE_SIZE;                 //及时更新英雄的行列值
7              checkIfRollScreen(hero.direction);           //检查是否需要滚屏
8         }else if(hx>destX){                               //判断英雄坐标和目的点的大小
9              hero.x = hx-j*HERO_MOVING_SPAN;              //修改英雄的 X 坐标
10             hero.col = hero.x/TILE_SIZE;                 //及时更新英雄的行列值
11             checkIfRollScreen(hero.direction);           //检查是否需要滚屏
12        }
13        if(hy<destY){                                     //判断英雄坐标和目的点的大小
14             hero.y = hy+j*HERO_MOVING_SPAN;              //计算英雄的 Y 坐标
15             hero.row = hero.y/TILE_SIZE;                 //及时更新英雄的行列值
16             checkIfRollScreen(hero.direction);           //检查是否需要滚屏
17        }else if(hy>destY){                               //判断英雄坐标和目的点的大小
18             hero.y = hy-j*HERO_MOVING_SPAN;              //修改英雄的 Y 坐标
19             hero.row = hero.y/TILE_SIZE;                 //及时更新英雄的行列值
20             checkIfRollScreen(hero.direction);           //检查是否需要滚屏
21        }}
22   hero.x = destX;                                        //修正英雄的 X 坐标
23   hero.y = destY;                                        //修正英雄的 Y 坐标
24   hero.col = destCol;                                    //修改英雄在地图中的行数
25   hero.row = destRow;                                    //修改英雄在地图中的列数
26   if(gv.offsetX<HERO_MOVING_SPAN){                       //如果偏移量小于英雄步进
27        gv.offsetX = 0;                                   //舍去
28   }
29   else if(gv.offsetX>TILE_SIZE - HERO_MOVING_SPAN){//图元大小与 offsetX 的差小于英雄移动步进
30        if(gv.startCol + GAME_VIEW_SCREEN_COLS < MAP_COLS -1){//进位
31             gv.offsetX=0;
32             gv.startCol+=1;
33        }}
34   if(gv.offsetY<HERO_MOVING_SPAN){                       //如果偏移量小于英雄步进
35        gv.offsetY = 0;                                   //舍去
36   }
37   else if(gv.offsetY>TILE_SIZE - HERO_MOVING_SPAN){      //图元大小与 offsetY 的差小于英雄移动步进
38        if(gv.startRow + GAME_VIEW_SCREEN_ROWS < MAP_ROWS -1){//进位
39             gv.startRow+=1;
40             gv.offsetY = 0;
41        }}
42   hero.direction = checkIfTurn();                        //检查是否需要拐弯
```

- 代码第 1～21 行根据步骤（1）中计算出的步数对英雄进行移动。每次移动时先判断当前位置和目的位置的大小关系，以此确定对英雄的当前位置做加法还是减法。
- 代码第 22～25 行为对英雄的 x、y 坐标以及其在地图中的行列值进行修正的代码。这么做的原因是，英雄的移动步径不能整除图元的大小，在执行完代码第 1～21 行的无级移动后，英雄所处的位置可能不在目的格子的中心，因此需要进行修正。
- 代码第 26～41 行为对 GameView 的 offsetX 和 offsetY 进行修正的代码。其原因同样是因为，英雄的移动步径不能整除图元大小。
- 对 offsetX 和 offsetY 进行修正的原则是，如果当前的偏移量小于英雄的移动步径（向左或

向上滚屏后的结果），则舍去该偏移量。如果当前偏移量与地图图元的差值小于英雄移动步进时（向右或向下滚屏的结果），则将该偏移量进位，修改相应的 startRow 或 startCol，并将偏移量置零。

- 代码第 42 行调用了 checkIfTurn 方法对英雄所处的新位置进行转弯检查，检查的依据是，判断相对于英雄当前方向的正前方以及左右两边是否有路可走，如果有路，则从可选路中随机选取一条并返回新道路的方向。

（3）进行到步骤（2）为止，英雄已经按照掷出的骰子点数前进了相应的地图格子数，此时应该停止英雄的移动并检查英雄是否与地图中的可遇实体发生相遇，其代码如下。

```
1    this.setMoving(false);                                  //停止走动
2    hero.setAnimationDirection(hero.direction%4);           //设置动画段为相应的静止态
3    if(!checkIfMeet()){                                     //检查停留点有没有可遇的东西
4         this.gv.setStatus(0);                              //设置 GameView 的状态为待命状态
5         gv.gvt.setChanging(true);                          //继续让骰子变换
6    }
7    hero.growStrength(gv.currentSteps);                     //增加英雄体力
```

- 代码第 1 行通过设置 isMoving 标志位来暂停 HeroGoThread 线程的执行。代码第 2 行将英雄的动画段设置为相应方向上的静止态。
- 代码第 3 行调用 checkIfMeet 方法检查英雄是否遇到了地图中可遇实体，如果遇到可遇实体，该方法将进行相应处理并返回 true，否则将返回 false。代码第 4～5 行为 checkIfMeet 方法返回 false 时执行的操作。

提示　由于篇幅有限，上述步骤中调用的 checkIfRollScreen 和 checkIfMeet 方法的具体代码将不予列出，读者可自行查阅随书光盘。

16.7.3　辅助线程——HeroBackDataThread 类的开发

HeroBackDataThread 是 Hero 类的另一个辅助线程，该线程的休眠时间比较长，其 run 方法中主要进行的工作如下。

- 减少英雄及其已占领城池的粮草，如果某个城池发生粮草危机，则提示玩家进行相应处理。
- 如果英雄目前有正在科研的项目，则推进项目的进度。如果项目已完成，则通过提示告知玩家。
- 随机选择一个城池向英雄占领的城池发动袭击，并提示玩家进行相应处理。
- 判断英雄的金钱、随身士兵和粮草是否超过一定值时，如果超过了一定值，则产生一场沙尘暴对英雄的金钱、兵力和粮草进行一定衰减，并通过提示告知玩家。
- 线程长时间休眠。

上述为 HeroBackDataThread 的主要功能，本书由于篇幅所限，同时该类还涉及后面要介绍的游戏提示模块的内容，故在此将不对其实现代码进行详细介绍，读者可自行查阅随书光盘。

16.8　表示层界面模块的开发

前面已经介绍了英雄角色的相关类，接下来将介绍前台表示层的各个界面，主要包括各个 View 以及一些 View 辅助线程。

16.8.1 滚屏类——ScreenRollView 类的开发

ScreenRollView 类采用竹简滚屏技术，是在游戏菜单界面单击“初入江湖”后出现的游戏背景概述界面，其框架代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 ScreenRollView. java。

```
1    package wyf.ytl;
2    import android.content.res.Resources;                //引入相关类
3    ……//该处省略了部分包的引入代码，读者可自行查阅随书光盘中的源代码
4    public class ScreenRollView extends SurfaceView implements SurfaceHolder.Callback{
5         HDZGActivity activity;
6         Bitmap bmpScroll;                               //竹简图片
7         Bitmap bmpBack;                                 //背景图片
8         DrawThread drawThread;                          //后台的重绘线程
9         ScreenRollThread screenRollThread;              //竹简滚屏的后台线程
10        float textSize = 23f;                           //字体的大小
11        int scrollStartX = 320;                         //竹简开始的 x 坐标
12        int scrollStartY = 60;                          //竹简开始的 y 坐标
13        int characterEachLine = 10;                     //每行显示的汉字个数
14        int characterSpanX = 36;                        //文字 x 方向间距
15        int characterSpanY = 25;                        //文字 y 方向间距
16        int characterNumber = 1;                        //已经显示了的字数
17        int textStartX=237;                             //文字起始的 x 坐标
18        int textStartY=135;                             //文字起始的 y 坐标
19        int startChar=0;                                //从字符串中的哪里开始输出，用于滚屏使用
20        int maxChar = 60;                               //竹简上最大能显示的文字个数
21        int alpha = 255;                                //透明度
22        int status;                                     //状态位，1 表示竹简滑入，2 表示文字显示
23        String msg = "东周末年，周室已经名存实亡，天下只是在名义上归于一家。各个诸侯国之间的战争早"
24             +"已是烽火连天，各种吞并和结盟屡见不鲜。不知你能否在这乱世靠着自己的才干谋"
25             +"略为自己守住一方土地，甚至是和其他诸侯国一样，逐鹿中原，并吞八荒？";
26        public ScreenRollView(HDZGActivity activity) {  //构造器
27             super(activity);
28             this.activity = activity;                   //得到 Activity 的引用
29             getHolder().addCallback(this);
30             drawThread = new DrawThread(getHolder());  //初始化绘制线程
31             screenRollThread = new ScreenRollThread(this); //初始化动画线程
32             initBitmap(getResources());                 //初始化图片资源
33             status = 1;                                 //设置状态到竹简划入
34        }
35        public void initBitmap(Resources r){             //初始化图片
36             bmpScroll = BitmapFactory.decodeResource(r, R.drawable.scroll);
                                                           //竹简图片
37        }
38        public void onDraw(Canvas canvas){               //屏幕的绘制方法
39             ……//该处省略了界面的绘制代码，将在后面给出
40        }
41        public void surfaceChanged(SurfaceHolder holder, int format, int width,int
          height){}
42        public void surfaceCreated(SurfaceHolder holder) {
43             ……//该处省略了绘制线程与动画线程的启动工作，读者可自行查阅随书光盘中的源代码
44        }
45        public void surfaceDestroyed(SurfaceHolder holder) {
46             drawThread.setFlag(false);
47        }
48        class DrawThread extends Thread{                 //后台的重绘线程
49             ……//该处省略了绘制线程中的绘制代码，读者可自行查阅随书光盘中的源代码
50        }}
```

- 第 5～22 行为成员变量的声明，主要是对界面中各个图片的坐标进行声明，这样在生成动画时只需要改变相应图片的坐标值即可。
- 第 23～25 行声明一个描述游戏背景的字符串，当绘制文字时，从该字符串中截取。
- 第 26～33 行为该类的构造器，在构造器中初始化绘制线程以及动画线程，并对需要的图片资源进行初始化。
- 第 38～40 行为重写的 onDraw 绘制方法，该方法负责绘制屏幕中的信息。
- 第 41～50 行 3 个接口中方法的实现以及重绘线程实现与之前章节介绍的完全相同，在此就不再赘述，读者可自行查阅随书光盘中的源代码。

16.8.2　滚屏线程——ScreenRollThread 的开发

前一节已经对竹简滚屏的界面进行了介绍，本节将对其后台线程进行开发，使竹简滚动起来，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 ScreenRollThread.java。

```
1    package wyf.ytl;
2    public class ScreenRollThread extends Thread{                //动画生层线程
3         boolean flag;                          //循环控制变量
4         int sleepSpan = 100;
5         ScreenRollView screenRoll;             //声明 screenRoll 的引用
6         int characterControl;                  //控制文字显示速度变量
7         int charNumber;                        //记录显示文字的个数，包括显示了后边滚屏掩盖的
8         public ScreenRollThread(ScreenRollView screenRoll){          //构造器
9              this.screenRoll = screenRoll;                           //得到 screenRoll 的引用
10             flag = true;                                            //设置循环标志位
11        }
12        public void run(){                                           //线程执行方法
13             while(flag){                                            //循环标志位为真时
14                  switch(screenRoll.status){
15                  case 1:                                            //竹简滑入
16                       screenRoll.scrollStartX -= 5;
17                       if(screenRoll.scrollStartX == 20){
18                            screenRoll.status = 2;                    //进入文字显示阶段，竹简停住
19                       }break;
20                  case 2:                                            //文字显示
21                       characterControl ++;
22                       if(characterControl == 3){
23                            screenRoll.characterNumber ++;           //每次显示的文字都多一个
24                            characterControl =0;
25                            charNumber ++;
26                            if(charNumber == screenRoll.msg.length()){  //显示结束
27                                 screenRoll.status = 3;              //3 是画面渐隐状态
28                            }
29                       }break;
30                  case 3:                                            //渐隐状态
31                       screenRoll.alpha -=5;                         //改变透明度
32                       if(screenRoll.alpha == 0){                    //当透明度到达 0 时
33                            screenRoll.status = 4;                   //切换屏幕
34                            screenRoll.activity.myHandler.sendEmptyMessage(11);
35                            this.flag = false;                       //停止该线程
36                       }break;
37                  }
38                  try{
39                       Thread.sleep(sleepSpan);                     //睡眠指定毫秒数
```

```
40                  }catch(Exception e){                        //捕获异常
41                      e.printStackTrace();                    //打印异常信息
42                  }}}}
```

- 第 15～19 行移动竹简，而当竹简移动到位后，将界面状态设置成 2，进入文字显示阶段。
- 第 20～29 行为显示文字阶段，每次多出现一个文字，而当文字全部显示完时，将状态设成 3。
- 第 30～37 行为状态为 3 时的操作代码，每次循环将透明度降低 5 点，而当透明度降到 0 时，将状态值设成 4，并向 Activity 发送 Handler 消息。

提示　因为篇幅有限，菜单（MenuView）、加载（LoadingView）、帮助（HelpView）、战斗（BattleField）等界面在此就不再介绍，这些界面的实现都非常简单，与之前章节介绍过的界面基本相同，读者可自行查阅随书光盘中的源代码。

16.8.3　游戏界面 GameView 的框架介绍

接下来将介绍该游戏最主要的界面——游戏界面 GameView。该界面绘制了游戏过程中的所有游戏信息，其开发步骤如下。

（1）首先搭建 GameView 的框架，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 GameView.java。

```
1    package wyf.ytl;
2    import static wyf.ytl.ConstantUtil.*;                    //静态引入ConstantUtil 的成员变量
3    ……//该处省略了部分类的引入代码，读者可自行查看随书光盘中的源代码
4    import android.view.View;                                //引入相关类
5    public class GameView extends SurfaceView implements SurfaceHolder.Callback,View.
     OnTouchListener {
6        ……//该处省略了成员变量的声明，将在后面给出
7        public GameView(HDZGActivity activity) {             //构造器
8            ……//该处省略了构造器中对各种资源的初始方法，读者可自行查看随书光盘中的源代码
9        }
10       ……//此处省略了声音的初始化和播放两个方法，读者可自行查看随书光盘中的源代码
11       public void initClass(){                             //初始化所有用到的类
12           manPanelView = new ManPanelView(this);
13           ……//该处省略了部分相关类的初始化工作，读者可自行查看随书光盘中的源代码
14           tianXiaView = new TianXiaView(this);             //初始化天下局势界面
15       }
16       public void initMap(){                               //初始化地图
17           ……//该处省略了地图资源的初始化工作，读者可自行查看随书光盘中的源代码
18       }
19       public static void initBitmap(Resources r){          //初始化图片资源的方法
20           ……//该处省略了图片资源的初始化工作，读者可自行查看随书光盘中的源代码
21       }
22       public void onDraw(Canvas canvas){                   //绘制屏幕
23           ……//该处省略了屏幕的绘制代码，将在后面给出
24       }
25       public void drawMiniMap(Canvas canvas,int col,int row){
26           ……//该处省略了缩略地图的绘制工作，读者可自行查看随书光盘中的源代码
27       }
28       public void drawDice(Canvas canvas){                 //画骰子
29           ……//该处省略了绘制骰子的工作，读者可自行查看随书光盘中的源代码
30       }
31       public void drawHeroData(Canvas canvas){             //主要的游戏数据
```

```
32              ……//该处省略了游戏界面下面各种数据，读者可自行查看随书光盘中的源代码
33          }
34          public int[] getDigitSequence(int data){          //获得一项数据的 6 位数码管显示序列
35              ……//该处省略了游戏界面下面各种数据的数码管的绘制，读者可自行查看随书光盘中的源代码
36          }
37          public void drawDigits(Canvas canvas,int [] sequence,int x,int y){
                                                              //按照数码管数字序列画出 6 个数字
38              for(int i=0;i<6;i++){
39                  canvas.drawBitmap(bmpDigit[sequence[i]], x+i*DIGIT_SPAN, y, null);
40              }
41          }
42          public void setCurrentDrawable(MyMeetableDrawable currentDrawable) {
43              this.currentDrawable = currentDrawable;
44          }
45          public void setStatus(int status)    {                //设置 GameView 的状态
46              this.status = status;
47          }
48          public boolean onTouch(View view, MotionEvent event){   //重写的监听器实现方法
49              ……//该处省略了事件监听方法的实现代码，读者可自行查看随书光盘中的源代码
50          }
51          public void stopGame(){
52              ……//该处省略了游戏界面声音线程的释放工作，读者可自行查看随书光盘中的源代码
53          }
54          public GameAlert getCurrentGameAlert() {              //返回当前的 Alert
55              return currentGameAlert;
56          }
57          public void setCurrentGameAlert(GameAlert currentGameAlert){//设置当前的 Alert
58              this.currentGameAlert = currentGameAlert;
59          }
60          public void surfaceChanged(SurfaceHolder holder,int format,int width,int
            height){}
61          public void surfaceCreated(SurfaceHolder holder){    //当 View 被创建时被调用
62              ……//该处省略了相关线程的启动工作，读者可自行查看随书光盘中的源代码
63          }
64          public void surfaceDestroyed(SurfaceHolder holder){ //当 View 被摧毁时被调用
65              this.drawThread.setIsViewOn(false);
66          }
67          class DrawThread extends Thread{                      //刷帧线程
68              ……//该处省略了线程中绘制代码，读者可自行查看随书光盘中的源代码
69          }}
```

- 第 11～15 行的方法初始化所有用到的类。
- 第 16～21 行初始化地图信息及图片资源，初始化地图资源时，需要从 LayerList 的一个对象中读取地图信息。
- 第 22～33 行为各种信息的绘制方法，除了 onDraw 比较复杂外，其他的方法都只将需要的图片绘制到指定位置即可。onDraw 方法将在后面进行介绍。
- 第 48～50 行为屏幕事件监听方法，根据玩家单击屏幕的事件进行相应的处理。
- 第 51～53 行方法的作用是将游戏中的线程以及声音停止，该方法会在游戏退出时被调用。
- 第 54～59 行为获得和设置当前的 GameAlert 方法，Alert 为游戏过程中弹出的提示。
- 第 60～69 行的实现与前面章节介绍的各个 View 中的完全相同，读者可自行查阅随书光盘中的源代码。

（2）下面给出 GameView 类中成员变量声明的代码。

```
1    private int status = 0;//绘制时的状态，0 正常游戏，1 英雄在走动,2 主菜单出现，3 选择将领界面，
```

```
2                                //4 战斗动画，5 破产结束游戏，6 统一中国
3                                //100-人物属性界面,99-武将情报 98-使用计谋 97-城池管理 96-天下局势
4     HDZGActivity activity;                                          //Activity 的引用
5     DrawThread drawThread;                                          //刷帧的线程
6     Hero hero;                                                      //英雄对象
7     static Bitmap dialogBack;                                       //存放对话框背景
8     static Bitmap dialogButton;                                     //存放对话框按钮
9     static Bitmap [][] heroAnimationSegments;                       //存放英雄所有动画段的图片
10    static Bitmap [] bmpDice;                                       //存放骰子的图片数组
11    static Bitmap dashboardBitmap;                                  //下面用于管理的图片
12    static Bitmap darkBitmap;                                       //菜单界面的背景
13    static Bitmap[] smallGameMenuOptions = new Bitmap[6];           //装主菜单分割以后的图片
14    static Bitmap[] bmpDigit;                                       //存放数码管数字的图片
15    static Bitmap bmpBattleFiled;                                   //战场背景图片
16    static Bitmap[] bmpHero;                                        //存放英雄各个动画帧的大图片
17    int [][] notInMatrix=new int[MAP_ROWS][MAP_COLS];               //存放整个地图的不可通过矩阵
18    int miniMapStartX = MINI_MAP_START_X;//初始情况下缩略地图开始的 X 坐标
19    int miniMapStartY = -160;            //初始情况下缩略地图开始的 Y 坐标
20    boolean showMiniMap;                 //是否显示缩略地图
21    int startRow = 0;                    //屏幕在大地图中的行数
22    int startCol = 0;                    //屏幕在大地图中的列数
23    int offsetX = 0;                     //屏幕定位点在大地图上的 x 方向偏移，用来实现无级滚屏
24    int offsetY = 0;                     //屏幕定位点在大地图上的 y 方向偏移，用来实现无级滚屏
25    LayerList layerList;                           //所有的层
26    MyMeetableDrawable [][] meetableMatrix;        //存放大地图的可遇矩阵
27    MyMeetableDrawable currentDrawable;            //记录当前碰到的可遇 Drawable 对象引用
28    MyMeetableDrawable previousDrawable;           //记录上一个碰到的可遇 Drawable 对象引用
29    MeetableLayer meetableChecker;
30    GameViewThread gvt;                            //修改后台数据的线程
31    BattleField battleField;
32    int suiXinBu=0;                                //记录英雄施放“随心步”时选择的步数
33    General fightingGeneral = null;                //存放当前出战的将军
34    ArrayList<Skill> skillToLearn;                 //将要学习的技能
35    int diceCount=2;                               //掷骰子时起作用的骰子个数,最大到 3
36    int []diceValue = {1,3,5};                     //存放骰子的值,0-5 代表 1-6 的骰子图片
37    int currentSteps;                              //记录本次掷骰子需要走几步
38    ManPanelView manPanelView;                     //人物属性窗口
39    WuJiangView wuJiangView;                       //武将情报窗口
40    UseSkillView useSkillView;                     //使用计谋窗口
41    CityManageView cityManageView;                 //城池管理窗口
42    SelectGeneral selectGeneral;                   //选中出征武将窗口
43    TianXiaView tianXiaView;                       //天下局势窗口
44    GameAlert currentGameAlert;                    //记录当前的消息提示
45    ArrayList<CityDrawable> allCityDrawable = new ArrayList<CityDrawable>();
                                                     //存放所有敌方的城池
46    ArrayList<General> freeGeneral;                //自由的将领,即不属于我方和敌方的
47    Paint paint;                                   //画笔
48    public static Resources resources;             //resources 的引用
49    MediaPlayer mMediaPlayer;                      //声音
50    SoundPool soundPool;                           //声音
51    HashMap<Integer, Integer> soundPoolMap;        //管理声音的 HashMap
```

说明 代码中各个成员变量的含义可见后面的注释，在对图片资源进行声明时，同样需要将其声明成静态变量。

16.8.4 游戏界面绘制方法 onDraw 的介绍

上一节介绍了 GameView 的框架，接下来将对其进行完善，首先是对 onDraw 绘制方法的完善，

该方法主要负责界面的绘制工作，完善步骤如下。

（1）该方法的框架如下，根据游戏状态值的不同绘制不同的界面。

```
1    public void onDraw(Canvas canvas){                          //绘制屏幕
2         if(status == 0 || status == 1 || status == 2){         //游戏主界面时
3              ……//该处省略了游戏主界面的绘制工作，代码将在后面给出
4         }
5         else if(status == 4){                                  //绘制战斗画面
6              battleField.onDraw(canvas);                       //调用战争动画的绘制方法
7         }
8         else if(status == 100){                                //绘制人物属性界面
9              manPanelView.onDraw(canvas);                      //调用人物属性界面的方法
10        }
11        else if(status == 99){                                 //绘制武将情报
12             wuJiangView.onDraw(canvas);                       //调用武将情报的绘制方法
13        }
14        else if(status == 98){                                 //绘制使用计谋
15             useSkillView.onDraw(canvas);                      //调用使用技能界面绘制方法
16        }
17        else if(status == 97){                                 //绘制城池管理
18             cityManageView.onDraw(canvas);                    //调用城池管理的绘制方法
19        }
20        else if(status == 3){                                  //绘制选择出征将领
21             selectGeneral.onDraw(canvas);                     //调用选择将领界面绘制方法
22        }
23        else if(status == 96){                                 //绘制天下局势
24             tianXiaView.onDraw(canvas);                       //调用天下局势的绘制方法
25        }
26        Paint paint = new Paint();                             //初始化画笔
27        paint.setColor(Color.RED);                             //设置颜色
28        paint.setTextSize(38);                                 //设置字体大小
29   }
```

说明　该方法对游戏的状态值进行判断，根据不同的状态绘制不同的信息，当游戏状态值表示不需要绘制游戏主界面时（即不为 0、1、2 时），调用相应封装类的绘制方法即可。

（2）游戏主界面的绘制代码如下，将下列代码插入到第（1）步代码的第 3 行处。

```
1    canvas.drawColor(Color.BLACK);                          //清除屏幕
2    int heroX=hero.x;                                       //英雄中心点的 x 坐标
3    int heroY=hero.y;                                       //英雄中心点的 y 坐标
4    int hCol = heroX/TILE_SIZE;                             //计算英雄中心点位于哪个格子
5    int hRow = heroY/TILE_SIZE;                             //计算英雄中心点位于哪个格子
6    int tempStartRow = this.startRow;                       //记录本次绘制时的定位行
7    int tempStartCol = this.startCol;                       //记录本次绘制时的定位列
8    int tempOffsetX = this.offsetX;                         //记录本次绘制时的 x 偏移
9    int tempOffsetY = this.offsetY;                         //记录本次绘制时的 y 偏移
10   for(int i=-1; i<=GAME_VIEW_SCREEN_ROWS; i++){          //绘制底层
11        if(tempStartRow+i < 0 || tempStartRow+i>MAP_ROWS){//当多画的那一行不存在，就继续
12             continue;
13        }
14        for(int j=-1; j<=GAME_VIEW_SCREEN_COLS; j++){
15             if(tempStartCol+j <0 || tempStartCol+j>MAP_COLS){ //当多画的那一列不存在，就继续
16                  continue;
17             }
18             Layer l = (Layer)layerList.layers.get(0);      //获得底层的图层
```

```
19              MyDrawable[][] mapMatrix=l.getMapMatrix();
20              if(mapMatrix[i+tempStartRow][j+tempStartCol] != null){
21                   mapMatrix[i+tempStartRow][j+tempStartCol].drawSelf(canvas,
22                        i,j,tempOffsetX,tempOffsetY);
23              }}}
24   for(int i=-1; i<=GAME_VIEW_SCREEN_ROWS; i++){          //绘制上层
25        if(tempStartRow+i < 0 || tempStartRow+i>MAP_ROWS){ //当多画的那一行不存在，就继续
26              continue;
27        }
28        for(int j=-1; j<=GAME_VIEW_SCREEN_COLS; j++){
29              if(tempStartCol+j <0 || tempStartCol+j>MAP_COLS){ //当多画的那一列不存在，就继续
30                   continue;
31              }
32              Layer l = (Layer)layerList.layers.get(1);  //获得上层的图层
33              MyDrawable[][] mapMatrix=l.getMapMatrix();
34              if(mapMatrix[i+tempStartRow][j+tempStartCol] != null){
                                                            //判断此处 MyDrawable 是否为空
35                   mapMatrix[i+tempStartRow][j+tempStartCol].drawSelf(canvas,
36                        i,j,tempOffsetX,tempOffsetY);
37              }
38              if(hRow-tempStartRow == i && hCol-tempStartCol == j){   //如果英雄位于此处
39                   hero.drawSelf(canvas,tempStartRow,tempStartCol,tempOffsetX,
                     tempOffsetY);
40              }}}
41   //绘制仪表盘
42   canvas.drawBitmap(dashboardBitmap, 0,
43        ConstantUtil.SCREEN_HEIGHT-dashboardBitmap.getHeight(), paint);//绘制仪表盘
44   drawDice(canvas);                                        //绘制骰子
45   drawHeroData(canvas);                                    //绘制仪表盘上的数据
46   canvas.drawBitmap(bmpHero[status==1?1:0], 0, HERO_FACE_START_Y, null);
                                                             //画英雄的头像
47   if(showMiniMap){                                         //检查是否需要显示小地图
48        drawMiniMap(canvas,hCol,hRow);
49   }
```

- 第 1～10 行清除屏幕并复制与绘制相关的数据。
- 第 10～23 行绘制底层地图，从 layerList 中得到底层地图的引用 1，然后对其表示地图信息的数组 mapMatrix 循环，得到其中的实体元素，再调用实体元素自身的绘制方法绘制图片。
- 第 24～40 行绘制上层地图的信息，同样从 layerList 得到上层的 Layer，然后循环绘制界面，当英雄绘制点时，调用英雄自身的绘制方法绘制英雄。
- 第 41～46 行绘制游戏界面的下方仪表盘，首先绘制背景图片，然后在指定位置绘制骰子、英雄个人数据以及英雄的头像。
- 第 47～49 行先判断是否需要绘制缩略地图，需要时调用 drawMiniMap 方法绘制缩略地图。

16.8.5 游戏界面屏幕监听方法 onTouch 的介绍

上一小节介绍了游戏界面的绘制方法，但是玩家还不能操控游戏，本节将对屏幕事件监听方法进行介绍，使玩家能够与游戏进行交互，其步骤如下。

（1）onTouch 方法框架的搭建，其框架代码如下。

```
1    public boolean onTouch(View view, MotionEvent event) {        //屏幕事件监听
2         if(event.getAction() == MotionEvent.ACTION_DOWN){        //捕捉屏幕被按下的事件
3              int x = (int)event.getX();                          //得到 X 坐标
4              int y = (int)event.getY();                          //得到 Y 坐标
```

```
5                if(this.status == 0){                                    //游戏主界面中
6                     ……//该处省略了游戏主界面显示时的屏幕事件处理代码。将在后面给出
7                }else if(this.status == 2){                              //主菜单出现时
8                     ……//该处省略了主菜单界面出现时的屏幕事件处理代码。读者可以自行查阅随书光盘
9                }else if(status == 100){                                 //人物属性界面出现
10                    manPanelView.onTouchEvent(event);
11               }else if(status == 99){                                  //武将情报界面
12                    wuJiangView.onTouchEvent(event);
13               }else if(status == 98){                                  //使用计谋界面
14                    useSkillView.onTouchEvent(event);
15               }else if(status == 97){                                  //城池管理界面
16                    cityManageView.onTouchEvent(event);
17               }else if(status == 3){                                   //选择武将界面
18                    selectGeneral.onTouchEvent(event);
19               }else if(status == 96){                                  //天下局势
20                    tianXiaView.onTouchEvent(event);
21               }
22          }
23          return true;
24     }
```

说明　在该方法中，根据当前游戏状态的不同，即用户界面显示的不同，做出不同的监听。当用户界面正在显示主菜单时，根据单击的菜单按钮的不同做出不同的操作。当游戏界面正在显示控制面板界面时，直接调用控制面板封装类中的事件处理方法即可，其中控制面板类将在第 16.9 节中进行介绍。

（2）游戏界面正在显示游戏过程时的屏幕监听，即当游戏状态值为 0 时的处理代码如下。

```
1     if(x>282 && x<310  && y>400 && y<470){                          //单击主菜单
2          darkBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.dark);
3          this.status = 2;                                            //改变状态值
4     }else if(x>62 && x<80 && y>408 && y<426){                        //单击了人物属性按钮
5          this.manPanelView.initData();                               //调用初始化数据的方法
6          this.status = 100;
7     }else if(x>62 && x<80 && y>427 && y<445){                        //武将情报
8          this.wuJiangView.initData();                                //调用初始化数据的方法
9          this.status = 99;
10    }else if(x>42 && x<61 && y>427 && y<445){                        //城池管理
11         this.cityManageView.initData();
12         this.status = 97;
13    }else if(x>22 && x<41 && y>427 && y<445){                        //天下局势
14         this.tianXiaView.initData();                                //调用初始化数据的方法
15         this.status = 96;                                           //改变状态值
16    }else if(x>2 && x<21 && y>427 && y<445){                         //使用计谋
17         this.useSkillView.initData();                               //调用初始化数据的方法
18         this.status = 98;                                           //改变状态值
19    }
20    //判断是否单击了人物头像
21    else if(x >0 && x<HERO_FACE_WIDTH && y>HERO_FACE_START_Y
22               && y< HERO_FACE_START_Y+HERO_FACE_HEIGHT){
23         if(this.status == 0){                                       //是否是待命状态
24              this.gvt.setChanging(false);                           //暂停变换骰子的线程
```

```
25              this.diceValue = hero.throwDice(this.diceCount);
26              currentSteps = 0;                          //记录要走几格
27              for(int i=0;i<diceCount;i++){              //骰子的个数
28                   currentSteps = currentSteps+diceValue[i]+1;
29              }
30              hero.startToGo(currentSteps);              //移动英雄
31         }
32    }else if(x > MAP_BUTTON_START_X && x< MAP_BUTTON_START_X+MAP_BUTTON_SIZE
33         && y> MAP_BUTTON_START_Y && y<MAP_BUTTON_START_Y+MAP_BUTTON_SIZE){//地图开关
34         showMiniMap = !showMiniMap;
35         miniMapStartY = -160;                          //复位
36    }else if(x > DICE_START_X+DICE_SPAN && x<DICE_START_X+DICE_SPAN+DICE_SIZE
37              && y>DICE_START_Y && y<DICE_START_Y+DICE_SIZE){//单击了第二颗骰子
38         if(this.diceCount == 1){                    //当前如果只有 1 个骰子活跃，就变成 2 个
39              this.diceCount++;                      //个数加 1
40         }else if(this.diceCount == 2){              //当前如果有 2 个骰子活跃，就变成 1 个
41              this.diceCount --;
42         }
43    }
44    else if(x > DICE_START_X+DICE_SPAN*2 && x<DICE_START_X+DICE_SPAN*2+DICE_SIZE
45              && y>DICE_START_Y && y<DICE_START_Y+DICE_SIZE){//点下了第三颗骰子
46         if(this.diceCount == 2){                    //当前如果只有 2 个骰子活跃，就变成 3 个
47              this.diceCount ++;                     //个数加 1
48         }else if(this.diceCount == 3){              //当前如果有 3 个骰子活跃，就变成 2 个
49              this.diceCount --;
50         }}
```

- 第 1～19 行根据玩家单击的位置判断所单击的是哪个按钮，然后根据单击按钮的不同初始化不同的数据，最后改变状态值。
- 第 21～31 行为单击人物头像时的处理代码。先判断状态值是否为 0，为 0 表示英雄正在待命状态，可以掷骰子走路。然后掷骰子并根据所掷子的点数移动英雄的位置。
- 第 32～35 行是玩家若单击缩略地图开关的响应代码，主要功能为置反是否显示缩略地图的标志位，同时缩略地图的 y 坐标设置到屏幕之外。
- 第 36～50 行当玩家单击第二个骰子或者第三个骰子时，根据当前骰子的个数决定是否显示所单击位置处的骰子。

16.8.6 游戏界面后台线程 GameViewThread 的介绍

GameViewThread 类主要负责定时读取 GameView 的状态，如果为待命则切换骰子的帧使其产生动画，并且当需要显示缩略地图时，实现缩略地图的滑入效果，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 GameViewThread.java。

```
1    package wyf.ytl;                                    //声明包语句
2    public class GameViewThread extends Thread{
3         GameView gv;                                   //游戏视图类的引用
4         int sleepSpan = 300;                           //休眠时间
5         int waitSpan = 1500;                           //空转时的等待时间
6         boolean flag = true;                           //线程是否执行标志位
```

```
7        boolean isChanging;                                    //是否需要变换骰子动画
8        public GameViewThread(GameView gv){                    //构造器
9             this.gv = gv;
10       }
11       public void run(){                                     //线程执行方法
12            while(flag){                                      //线程正在执行
13                 while(isChanging){                           //需要换帧
14                      int diceNumber = gv.diceCount;
15                      for(int i=0;i<diceNumber;i++){
16                           gv.diceValue[i] = (gv.diceValue[i]+1)%6;
17                      }
18                      if(gv.showMiniMap){                     //判断是否显示缩略图
19                           gv.miniMapStartY+=30;
20                           if(gv.miniMapStartY>=0){          //已经移动到位
21                                gv.miniMapStartY = 0;
22                           }
23                      }
24                      try{
25                           Thread.sleep(sleepSpan);          //睡眠指定毫秒数
26                      }catch(Exception e){                    //捕获异常
27                           e.printStackTrace();              //打印异常信息
28                      }
29                 }
30                 try{                                         //线程的空转等待时间
31                      Thread.sleep(waitSpan);                 //睡眠指定毫秒数
32                 }catch(Exception e){                         //捕获异常
33                      e.printStackTrace();                    //打印异常信息
34                 }}}
35       public void setChanging(boolean isChanging){           //设置是否需要换骰子
36            this.isChanging = isChanging;
37       }}
```

- 第 4～5 行声明两个表示睡眠时间的值，一个是正常的休眠时间，另一个是线程空转的睡眠时间，因为有时不需要转动骰子也不需要绘制缩略地图，只需让该线程空转。
- 第 14～17 行需要换骰子时，循环更换骰子的点数。第 18～23 行为显示缩略地图时，移动缩略地图位置的代码。

16.9 管理面板模块的开发

上一节对游戏的主界面以及其相关界面进行了介绍，本节将对游戏过程中的各个管理面板界面进行简单介绍，这些管理面板的作用是查询或管理游戏的相关信息，例如，城池、粮草、将领信息等。

16.9.1　人物属性面板类 ManPanelView 的开发

首先介绍的是人物属性界面，该界面为玩家单击主界面中的“人”按钮弹出的界面，主要的作用是显示英雄的基本属性以及英雄拥有的技能。其运行效果如图 16-30 和图 16-31 所示。

接下来对该界面的开发步骤进行简单介绍，步骤如下。

（1）框架的搭建，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 ManPanelView.java。

```
1    package wyf.ytl;
2    import java.util.HashMap;                                  //引入 HashMap 类
```

```
3     ……//该处省略了部分类的引入代码，读者可以自行查阅随书光盘中的源代码
4     import android.view.MotionEvent;                    //引入 MotionEvent 类
5     public class ManPanelView{
6          GameView gameView;                             //gameView 的引用
7          int yeSpan = 11;                               //每页显示的个数
8          int yeSpan2 = 10;                              //每页显示的个数
9          int currentI = 0;                              //当前绘制的屏幕最上边的元素下标
10         int selectI = 0;                               //当前选中行数
```

▲图 16-30　基本属性

▲图 16-31　技能信息

```
11         int startY = 120;                              //y 坐标
12         int startX = 25;
13         int status = 0;                                //显示的标记位,0-属性列表  1-技能列表
14         String[][] items1;                             //需要列出的属性
15         String[][] items2;                             //需要列出的技能
16         static Bitmap threeBitmap;                     //右上角的 3 个按钮
17         ……//该处省略了部分图片引用的声明。读者可自行查看随书光盘中的源代码
18         static Bitmap logo;
19         Paint paint;                                   //画笔
20         public ManPanelView(GameView gameView) {       //构造器
21              this.gameView = gameView;                 //得到 gameView 的引用
22              paint = new Paint();                      //初始化画笔
23              paint.setARGB(255, 42, 48, 103);          //设置字体颜色
24              paint.setAntiAlias(true);                 //抗锯齿
25              initData();                               //初始化数据
26         }
27         public static void initBitmap(Resources r){//初始化图片资源
28              ……//该处省略了图片资源的初始化工作，读者可自行查阅随书光盘中的源代码
29         }
30         public void initData(){                        //初始化数据
31              ……//该处省略了该界面所用到的数据的初始化工作，将在后面进行介绍
32         }
33         public void onDraw(Canvas canvas){             //绘制的方法
34              ……//该处省略了界面绘制的代码，将在后面进行介绍
35         }
36         public boolean onTouchEvent(MotionEvent event) {   //屏幕监听方法
37              ……//该处省略了该界面的屏幕监听方法，将在后面进行介绍
38         }}
```

- 第 7～8 行的两个变量分别表示基本属性界面以及技能属性界面每页最多显示的数量。

- 第 9～10 行的两个变量则表示，当前屏幕中最上面信息的下标以及当前选中信息的下标。
- 第 20～26 行为该类的构造方法，在方法中对画笔进行初始化并设置画笔颜色等，同时调用数据初始化方法对数据进行初始化。
- 第 27～29 行的方法为图片资源初始化的方法，在该方法中，对该界面所用到的图片进行初始化，第 30～32 行则是对该界面需要的数据进行初始化。
- 第 33～35 行为该界面的绘制方法，在介绍 GameView 的 onDraw 方法时，已经讲到，当 GameView 中状态值表示当前显示的是人物属性界面，直接调用人物属性界面的 onDraw 方法进行绘制。
- 第 36～38 行为屏幕监听方法，同样是在 GameView 的屏幕监听方法中被调用。

（2）初始化数据方法 initData 的实现，其代码如下。

```
1    public void initData(){                                          //初始化数据的方法
2         this.items1 = new String[][]{                               //初始化人物属性
3              {"名称:", gameView.hero.getName()},                    //英雄的名称
4              {"等级:", gameView.hero.getLevel()+" 级"},             //英雄的等级
5              {"官衔:", gameView.hero.getTitle()},                   //英雄的官衔
6              {"将军:", gameView.hero.getGeneralNumber()+" 个"},     //将军的个数
7              {"城池:", gameView.hero.getCityList().size()+" 个"},//城池的个数
8              {"黄金:", gameView.hero.getTotalMoney()+" 金"},        //黄金的数量
9              {"粮草:", gameView.hero.getTotalFood()+" 石"},         //粮草的数量
10             {"兵力:", gameView.hero.getTotalArmy()+""},            //兵力的数量
11             {"总人口:", gameView.hero.getTotalCitizen()+" 人"},    //总人口的数量
12             {"投石车:", gameView.hero.getTotalWarTank()+" 个"},    //投石车的个数
13             {"箭垛:", gameView.hero.getTotalWarTower()+" 个"},     //箭垛的数量
14        };
15        //初始化技能
16        HashMap<Integer,Skill> heroSkill = gameView.hero.getHeroSkill();//得到英雄的所有技能
17        items2 = new String[heroSkill.size()][3];                    //初始化 items2 数组
18        Set<Integer> s = heroSkill.keySet();                        //得到HashMap键的集合
19        int k = 0;
20        for(Integer i : s){                                         //循环
21             Skill skill = heroSkill.get(i);                        //得到技能
22             items2[k][0] = skill.getName();                        //技能名字
23             items2[k][1] = skill.getProficiencyLevel()+"";         //技能等级
24             items2[k][2] = skill.getStrengthCost()+"";             //技能消耗的体力
25             k++;
26        }}
```

- 第 2～14 行根据 gameView 中的 hero 对象初始化人物属性界面需要的相关信息，并将这些信息整理到字符串数组中。
- 第 16～25 行对英雄拥有的技能进行循环，得到每个技能，并将其信息存放到数组中。

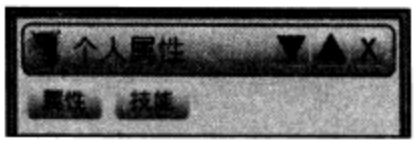
（3）屏幕绘制方法 onDraw 的实现，其代码如下。

```
1    public void onDraw(Canvas canvas){                          //绘制的方法
2         canvas.drawBitmap(panel_back, 0, 0, paint);            //在指定位置绘制图片
3         ……//该处省略了图片以及文字的绘制代码，读者可自行查看随书光盘中的源代码
4         canvas.drawText("技能", 97, 78, paint);                //在指定位置绘制文字
5         if(status == 0){                                       //画属性界面时
6              int tempCurrentI = currentI;
7              if(items1.length < yeSpan){                       //当一个屏幕可以显示图全部时
8                   for(int i=0; i<items1.length; i++){
9                        for(int j=0; j<items1[i].length; j++){
10                            canvas.drawText(items1[i][j], startX+j*115, startY + 30*i,
```

```
                                paint);
11                          }
12                    }
13              }else{                                          //当一个屏幕显示不下全部图时
14                    for(int i=tempCurrentI; i<tempCurrentI+yeSpan; i++){
15                          for(int j=0; j<items1[i].length; j++){
16                                canvas.drawText(items1[i][j], startX+j*115, startY +
                                  30*(i- tempCurrentI), paint);
17                          }}}
18              canvas.drawBitmap(selectBackground, 10, startY + 30*selectI - 22, paint);
                                                                //选中后的效果
19              if(tempCurrentI != 0){                          //绘制小的向上箭头
20                    canvas.drawBitmap(upBitmap, 150, 85, paint);  //绘制向上箭头
21              }
22              if(items1.length>yeSpan && (tempCurrentI+yeSpan) < items1.length){
                                                                //绘制小的向下箭头
23                    canvas.drawBitmap(downBitmap, 150, 455, paint); //绘制向下箭头
24              }
25        }else if(status == 1){                                //绘制技能界面
26              canvas.drawBitmap(menuTitle, 10, 100, paint);   //绘制菜单头
27              canvas.drawText("技能名", 15, 120, paint);      //绘制文字“技能名”
28              canvas.drawText("等级", 125, 120, paint);       //绘制文字“等级”
29              canvas.drawText("消耗", 240, 120, paint);       //绘制文字“消耗”
30              int tempCurrentI = currentI;
31              if(items2.length < yeSpan2){                    //当一个屏幕可以显示全部图时
32                    for(int i=0; i<items2.length; i++){
33                          canvas.drawText(items2[i][0], 15, startY + 35 + 30*i, paint);
                                                                //绘制文字
34                          canvas.drawText(items2[i][1], 125, startY + 35 + 30*i, paint);
                                                                //绘制文字
35                          canvas.drawText(items2[i][2], 240, startY + 35 + 30*i, paint);
                                                                //绘制文字
36                    }
37              }else{                                          //当一个屏幕显示不全部图时
38                    for(int i=tempCurrentI; i<tempCurrentI+yeSpan2; i++){
39                          canvas.drawText(items2[i][0], 15, startY + 35 + 30*(i-tempCurrentI),
                            paint);
40                          canvas.drawText(items2[i][1], 125, startY + 35 + 30*(i-tempCurrentI),
                            paint);
41                          canvas.drawText(items2[i][2], 240, startY + 35 + 30*(i-tempCurrentI),
                            paint);
42                    }
43              }
44              canvas.drawBitmap(selectBackground, 10, startY + 35 + 30*selectI - 22,
                paint);
45              if(tempCurrentI != 0){                          //绘制小的向上箭头
46                    canvas.drawBitmap(upBitmap, 150, 85, paint);    //绘制图片
47              }
48              if(items2.length>yeSpan2 && (tempCurrentI+yeSpan2) < items2.length){
                                                                //绘制小的向下箭头
49                    canvas.drawBitmap(downBitmap, 150, 455, paint); //绘制图片
50              }}
```

- 第 2～4 行绘制人物属性界面公共的部分，如图 16-32 所示。
- 第 5～24 行为绘制人物属性界面的代码，其中第 7～12 行为一个屏幕可以显示下所有信息的情况，即不需要分页显示，第 14～17 行为需要分页的情况。第 19～24 行根据一页是否显示完全来绘制上下小箭头，来表示上面或者下面是否还有信息。
- 第 25～50 行为绘制技能信息的代码，第 26～29 行绘制界面头，如图 16-33 所示，之后的

绘制与人物基本属性的界面基本相同，根据是否分页采用不同的绘制方法。



▲图 16-32　人物属性界面公共部分

▲图 16-33　技能界面的头信息

（4）屏幕监听方法 onTouchEvent 的实现，其代码如下。

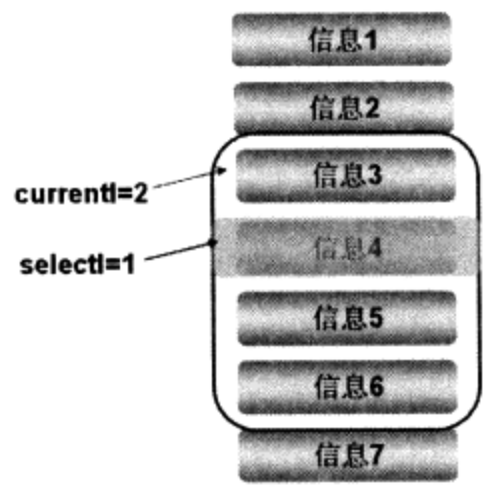
```
1    public boolean onTouchEvent(MotionEvent event) {
2         if(event.getAction() == MotionEvent.ACTION_DOWN){   //屏幕被按下
3              int x = (int) event.getX();                    //得到 x 坐标
4              int y = (int) event.getY();                    //得到 y 坐标
5              if(y>15 && y<45){
6                   if(x>212 && x<242){                       //单击了向下翻页按钮
7                        if(status == 0){                     //人物基本属性时
8                             selectI++;                      //选中下一条信息
9                             if(items1.length < yeSpan){     //当一个屏幕可以全部显示时，即不需
                                                              //要滚屏
10                                 if(selectI > items1.length-1){
11                                      selectI = items1.length-1;//当到达最后时
12                                 }
13                            }else {                   //当一屏显示不了全部，需要滚屏时
14                                 if(selectI > yeSpan-1){     //当到页面最下面时
15                                      selectI = yeSpan-1;    //选中下一个
16                                      currentI++;            //滚动屏幕
17                                      if((currentI+yeSpan) > items1.length){
18                                           currentI--;       //恢复
19                                      }}}
20                        }else if(status == 1){              //技能显示时
21                             selectI++;                      //选中下一条
22                             if(items2.length < yeSpan2){    //当一个屏幕可以全部显示时，
                                                              //即不需要滚屏
23                                 if(selectI > items2.length-1){//到达最后一条信息时
24                                      selectI = items2.length-1;
25                                 }
26                            }else {                         //当一屏显示不了全部，需要滚屏时
27                                 if(selectI > yeSpan2-1){    //到达页面最下面时
28                                      selectI = yeSpan2-1;   //选中下一个
29                                      currentI++;            //滚动屏幕
30                                      if((currentI+yeSpan2) > items2.length){
31                                           currentI--;       //恢复
32                                      }}}}
33                   }else if(x>243 && x<273){                //单击了向上翻页按钮
34                        if(--selectI < 0){                  //当到达最前面时
35                             selectI = 0;                   //选中第一条
36                             if(--currentI < 0){
37                                  currentI = 0;             //设置为第一条
38                             }}
39                   }else if(x>274 && x<304){                //单击了关闭按钮
40                        this.status = 0;                        //恢复状态值到 0
41                        this.selectI = 0;                       //选中第一个
42                        this.currentI = 0;                      //滚屏到第一条信息
43                        gameView.setStatus(0);                  //设置 gameView 的状态值
44                   }
45              }else if(y>57 && y<57+buttonBackGround.getHeight() &&
46                        x>15 && x<15 + buttonBackGround.getWidth()){ //单击了属性按钮
47                   this.status = 0;                                  //恢复状态值到 0
48                   this.selectI = 0;                                 //选中第一个
```

```
49                  this.currentI = 0;                                  //滚屏到第一条信息
50              }else if(y>57 && y<57+buttonBackGround.getHeight() &&
51                      x>85 && x<85 + buttonBackGround.getWidth()){ //单击了技能按钮
52                  this.status = 1;                                    //设置状态值到 1
53                  this.selectI = 0;                                   //选中第一个
54                  this.currentI = 0;                                  //滚屏到第一条信息
55              }}
56          return true;
57      }
```

- 第 6～33 行为单击向下翻页按钮的处理代码，其中第 7～19 行表示当前显示的是人物基本属性界面，其中根据一屏是否能够显示全部进行分类处理。
- 第 33～38 行单击的是向上翻页按钮，选中上一条，当到达第一条时不移动。
- 第 39～44 行表示玩家单击的是关闭按钮，将所有变量恢复到 0 状态，然后设置 gameView 的状态值到游戏状态。
- 第 45～49 行为“属性”按钮的事件监听，将状态置成 0 并选中第一条信息，同时将屏幕滚动到最上面。
- 第 50～55 行为“技能”按钮的事件监听，将状态置成 1 并选中第一条信息，同时也将屏幕滚动到最上面。

提示

其中 selectl 与 currentl 分别表示当前屏幕中所选中的信息编号与当前屏幕最上方的信息在所有信息中的编号，如图 16-34 所示。



▲图 16-34　多条信息分页显示

16.9.2　城池管理面板类 CityManageView 的开发

上一节已经介绍了人物属性界面的开发，本节将介绍另一个控制面板——城池管理界面的开发，该界面主要是对英雄所拥有的城池进行管理，例如，粮草的分配、兵力的划拨等，其运行效果如图 16-35 和图 16-36 所示。

接下来对该界面的开发步骤进行简单介绍，步骤如下。

（1）框架的搭建，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 CityManageView.java。

```
1    package wyf.ytl;
```

```
2     import java.util.ArrayList;                                    //引入相关类
3     ……//该处省略了部分类的引入代码，读者可自行查阅随书光盘中的源代码
4     import android.view.MotionEvent;                               //引入相关类
5     public class CityManageView {
6          GameView gameView;
7          int yeSpan1 = 9;                                          //主页显示城池的个数
8          int yeSpan2 = 9;                                          //管理页显示城池的个数
9          int currentI = 0;                                         //当前绘制屏幕最上边的元素下标
10         int selectI = 0;
```

▲图 16-35　城池列表

▲图 16-36　城池管理界面

```
11         int startY = 120;
12         int startX = 17;
13         int status = 0;                                           //0-主界面，1-详细信息
14         String[][] items1;                                        //城池列表
15         String[][] items2;                                        //城池管理界面列表
16         ArrayList<CityDrawable> cityList;                         //存放英雄拥有的城池
17         CityDrawable selectCityDrawable;                          //当前选中的城市
18         static Bitmap menuTitle;                                  //表格的标题背景
19         ……//该处省略了部分图片的声明，读者可自行查看随书光盘的源代码
20         static Bitmap logo;
21         Paint paint;                                              //画笔
22         public CityManageView(GameView gameView) {                //构造器
23              ……//该处省略了构造方法中的相关代码，读者可自行查看随书光盘中的源代码
24         }
25         public static void initBitmap(Resources r){               //初始化图片资源
26              ……//该处省略了图片资源的初始化代码，读者可自行查阅随书光盘中的源代码
27         }
28         public void initData(){                                   //初始化数据
29              ……//该处省略了数据的初始化工作，读者可自行查阅随书光盘中的源代码
30         }
31         public void onDraw(Canvas canvas){                        //屏幕绘制的方法
32              ……//该处省略了屏幕的绘制工作，读者可自行查阅随书光盘中的源代码
33         }
34         public boolean onTouchEvent(MotionEvent event){           //屏幕监听方法
35              ……//该处省略了屏幕事件的处理代码，代码将在后面给出
36         }
37         public void initItems2(){
38              ……//该处省略了城池详细信息的初始化工作，代码将在后面给出
39         }}
```

- 第 16 行的 ArrayList 存储的是英雄所拥有的所有城池。
- 第 22～24 行的构造方法中先对画笔进行初始化并做相应设置，然后调用初始化数据方法 initData 初始化相关数据。
- 第 25～27 行初始化所有图片资源。第 28～29 行则是该界面中数据的初始化方法。
- 第 31～33 行为界面的绘制方法，在方法中会根据该类中当前的状态值绘制不同的界面。
- 第 34～36 行为界面的屏幕监听方法，根据当前的状态值以及玩家单击的位置做出相应的处理。
- 第 37～39 行的代码根据玩家选择的城池，初始化城池的详细信息。

（2）接下来介绍城池详细信息的初始化方法 initItems2，其代码如下。

```
1    public void initItems2(){                                          //初始化选中城池的信息
2         selectCityDrawable = cityList.get((selectI+currentI));        //得到选中的城池
3         items2 = new String[][]{
4              {"名称：", selectCityDrawable.getCityName(),"","",},  //城池的名称
5              {"等级：", selectCityDrawable.getLevel()+"","","",},  //城池的等级
6              {"兵力：", selectCityDrawable.getArmy()+"","剩余：",gameView.hero.getArmyWith-
               Me()+""},
7              {"粮草:",selectCityDrawable.getFood()+"","剩余:",gameView.hero.getFood()+""},
8              {"防御：", GameFormula.getCityDefence(selectCityDrawable)+"","","",},
9              {"攻击：", GameFormula.getCityAttack(selectCityDrawable)+"","","",},
10             {"居民：", selectCityDrawable.getCitizen()+"","","","","",}, //城池的居民人口
11             {"箭垛：", selectCityDrawable.getWarTower()+"","剩余：",gameView.hero.
               warTower+""},
12             {"战车：", selectCityDrawable.getWarTank()+"","剩余：",gameView.hero.
               warTank+""}
13        };}
```

说明 代码中的第 2 行得到在城池列表中选中的城池，然后第 3～12 行得到该城池的信息，并组织到字符串数组 items2 中以供界面的显示。

（3）屏幕监听方法 onTouchEvent 的实现，在该方法中，根据状态值的不同分为两部分，第一部分为城池列表界面的处理代码，另一部分为某个城池管理界面的处理代码。城池列表界面的处理方式与人物属性界面的基本相同，在此不再赘述，读者可自行查阅随书光盘中的源代码，下面给出的是城池管理界面的处理代码。

```
1    else if(status == 1){                                          //当是城池管理界面时
2         if(x>250 && x<265 && y>130 && y<145){                     //加兵力
3              if(gameView.hero.getArmyWithMe()>=500){              //兵大于 500 时
4                   gameView.hero.setArmyWithMe(gameView.hero.getArmyWithMe()-500);
5                   this.selectCityDrawable.setArmy(this.selectCityDrawable.getArmy()
                    +500);
6              }else {
7                   this.selectCityDrawable.setArmy(this.selectCityDrawable.getArmy()
8                                                  +gameView.hero.getArmyWithMe());
9                   gameView.hero.setArmyWithMe(0);                  //置空
10             }
11             initItems2();                                        //初始化城池的详细信息
12        }else if(x>270 && x<285 && y>130 && y<145){               //减兵力
13             if(this.selectCityDrawable.getArmy()>=500){          //当大于 500 时
```

```
14                  gameView.hero.setArmyWithMe(gameView.hero.getArmyWithMe()+500);
15                  this.selectCityDrawable.setArmy(this.selectCityDrawable.getArmy()
                    -500);
16              }else {
17                  gameView.hero.setArmyWithMe(gameView.hero.getArmyWithMe()
18                                  +this.selectCityDrawable.getArmy()); //当不足 500 时
19                  this.selectCityDrawable.setArmy(0);         //设置为 0
20              }
21              initItems2();                                   //初始化城池的详细信息
22          ……//该处省略了部分按钮的处理代码，读者可自行查阅随书光盘中的源代码
23          }else if(x>250 && x<265 && y>310 && y<325){         //加战车
24              if(gameView.hero.getWarTank()>0){               //当只有英雄身上有战车时
25                  this.selectCityDrawable.setWarTank(this.selectCityDrawable.
                    getWarTank()+1);
26                  gameView.hero.setWarTank(gameView.hero.getWarTank()-1);
27              }
28              initItems2();                                   //初始化城池的详细信息
29          }else if(x>270 && x<285 && y>310 && y<325){         //减战车
30              if(this.selectCityDrawable.getWarTank()>0){     //当城池中有战车时
31                  gameView.hero.setWarTank(gameView.hero.getWarTank()+1);
32                  this.selectCityDrawable.setWarTank(this.selectCityDrawable.
                    getWarTank()-1);
33              }
34              initItems2();                                   //初始化城池的详细信息
35          }}
```

- 第 2～11 行为加兵力按钮的事件处理代码，先判断英雄身上的兵力是否大于 500，当大于 500 时划拨 500 兵力到选中的城池，而不足 500 时将全部兵力划拨到选中的城池中。
- 第 12～21 行是减兵力按钮的事件处理代码，此时需要先判断选择的城池中的兵力是否大于 500，因为需要从城池中划拨 500 兵力跟随英雄，而不足 500 时，需要将全部兵力划拨到英雄身边。
- 第 22 行省略了处理箭垛、粮草划拨的代码，读者可自行查阅随书光盘中的源代码。
- 第 23～34 行为处理战车调拨的代码，其原理与划拨兵力的相同，在此就不再赘述，读者可自行开发或查阅随书光盘中的源代码。

提示

其他管理面板的实现技术与前面介绍的两个面板类基本相同，因本书篇幅有限，在此就不再介绍，读者可自行查阅随书光盘中的源代码。

16.10 地图中可遇实体模块的开发

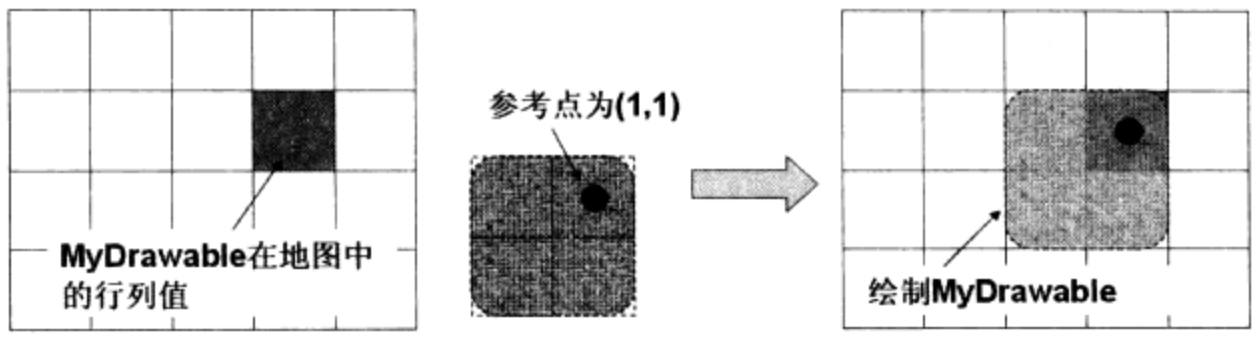
游戏中英雄每走完指定骰子数的地图格子，都将检测当前位置是否与地图的可遇实体发生相遇，本节就来简单介绍可遇实体对象的开发，其中涉及的类有 MyDrawable、MyMeetableDrawable 以及继承自 MyMeetableDrawable 的各个子类。

16.10.1　绘制类——MyDrawable 的开发

MyDrawable 是 MyMeetableDrawable 的父类，游戏中不可遇的地图元素如道路、花等均是

MyDrawable 类的对象。在本游戏中，MyDrawable 的宽度和高度可以是地图图元大小（31 像素）的任意倍数，如代表稻田的大小为 62×62，占地图中 4 个格子。

当 MyDrawable 的大小超过 31×31 时，就需要为其指定参考点，在绘制 MyDrawable 时将根据其参考点和在地图中所占的行和列进行绘制。在本游戏中，参考点用相对于 MyDrawable 左下角的行和列来指定，图 16-37 说明了如何基于参考点来绘制 MyDrawable。



▲图 16-37　在地图中绘制 MyDrawable 示意图

在图 16-37 中，要绘制的 MyDrawable 在地图中占 2 行 2 列，其中定位点位于右上角的那个格子，相对左下角的坐标为(1,1)。在绘制 MyDrawable 时，将参考点所在的格子对到 MyDrawable 在地图中所占的行和列即可。

同时，由于 MyDrawable 的大小可能超过一个地图格子的尺寸，在表示其可通过情况时，不可以用 0 或 1 简单概括。因为有些 MyDrawable 可能部分位置可通过，其他位置不可通过，这时需要将 MyDrawable 内部所有不可通过的点记录下来。如图 16-37 的 MyDrawable，如果其上边两个图元不可通过，下面两个可通过，则其不可通过点为(1,0)和(1,1)，同样是相对于 MyDrawable 左下角的坐标。

下面来介绍 MyDrawable 类的开发，其代码框架如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 HeroGoThread.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    import static wyf.ytl.ConstantUtil.*;                               //引入相关类
3    ……//此处省略部分引入相关类的代码，读者可自行查阅随书光盘
4    import android.graphics.Canvas;                                     //引入相关类
5    public class MyDrawable implements Externalizable{
6         private static final long serialVersionUID = 9191440096790116 82L;//持久化版本序列号
7         Bitmap bmpSelf;                                                //图片的引用
8         int width;                                                     //图元的宽度
9         int height;                                                    //图元的高度
10        int col;                                                       //在地图中所在的列
11        int row;                                                       //在地图中所在的行
12        int refCol;                    //定位点在本 MyDrawable 中所占的列，以左下角为原点
13        int refRow;                    //定位点在本 MyDrawable 中所占的行，以左下角为原点
14        int [][] noThrough;                                            //不可通过矩阵
15        boolean meetable;                                              //是否可以遇到
16        public void writeExternal(ObjectOutput out) throws IOException {}
17        public void readExternal(ObjectInput in) throws IOException,ClassNotFound-
          Exception {}
18        public MyDrawable(){//无参数构造器}
19        public MyDrawable(Bitmap bmpSelf,boolean meetable,int width,int height,int
          col,int row,
20                                 int refCol,int refRow,int [][] noThrough){
                                                        //构造器}
```

```
21          public void drawSelf(Canvas canvas,int screenRow, int screenCol,int offsetX,int
            offsetY){                                          //绘制自己
22                int x = (screenCol-refCol)*TILE_SIZE; //求出MyDrwable左上角的x坐标
23                int y = screenRow*TILE_SIZE+(refRow+1)*TILE_SIZE-height;
                                                               //求出MyDrwable左上角的y坐标
24                canvas.drawBitmap(bmpSelf, x-offsetX, y-offsetY, null);
                                                               //根据自己的左上角的xy坐标画出自己
25          }}
```

- 代码第 10～11 行声明了用于记录 MyDrawable 对象在地图中位置的成员变量，代码第 12～13 行声明了用于记录参考点在 MyDrawable 内部位置的成员变量。
- 代码第 14 行为记录 MyDrawable 自身通过情况的二维数组，由于不可通过位置可能不止一个，所以，将所有不可通过位置存放到一个二维数组中。
- 代码第 15 行声明的 meetable 为该 MyDrawable 可遇与否的标志位，true 表示可遇，false 表示不可遇。
- 代码第 16～17 行为对 Externalizable 接口中 writeExternal 和 readExternal 方法的实现，代码第 18 行为 MyDrawable 的无参数构造器，这些都是服务于游戏存档和读取模块的，在此将不详述。
- 代码第 19 行为 MyDrawable 的构造器，在该构造器中将 MyDrawable 的主要成员变量初始化。
- 代码第 21～25 行为 drawSelf 方法，该方法首先通过将要绘制到屏幕上的行和列以及参考点的位置计算出 MyDrawable 左上角的 x 和 y 坐标，然后减去各自的偏移量进行绘制。

16.10.2　抽象类——MyMeetableDrawable 的开发

MyMeetableDrawable 为抽象类，其子类对象代表地图中的可遇实体，要想让 MyMeetableDrawable 具有可遇的特性，除了要继承 MyDrawable 外，还需要实现 View.OnTouchListener 接口。游戏中当英雄与可遇实体发生相遇时，可遇实体对象将会用自己替换掉 GameView 的 View.OnTouchListener 监听器以执行特定的业务逻辑。MyMeetableDrawable 类的代码框架如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 MyMeetableDrawable. java。

```
1     package wyf.ytl;                                    //声明包语句
2     import static wyf.ytl.ConstantUtil.*;               //引入相关类
3     ……//在此省略部分引入相关类的代码，读者可自行查阅随书光盘
4     import android.view.View;                           //引入相关类
5     public abstract class MyMeetableDrawable extends MyDrawable implements View.OnTouch-
      Listener,
6                 Externalizable {
7           int [][] meetableMatrix;                      //可遇矩阵
8           Bitmap bmpDialogBack;                         //对话框背景图片
9           Bitmap bmpDialogButton;                       //对话框的按钮背景图片
10          Hero tempHero;                                //英雄的引用
11          public void writeExternal(ObjectOutput out) throws IOException {}
12          public void readExternal(ObjectInput in) throws IOException,ClassNotFound-
            Exception {}
13          public MyMeetableDrawable(){}                 //无参数构造器
14          public MyMeetableDrawable(){}                 //有参数构造器
15          public abstract void drawDialog(Canvas canvas,Hero hero);
                                                          //抽象方法：在游戏屏幕上绘制对话框
16          public void drawString(Canvas canvas,String string){//绘制给定的字符串到对话框上}
17          public boolean onTouch(View arg0, MotionEvent arg1) {//实现View.OnTouchListener
```

```
18    }                                                    //接口的方法}
```

- 代码第 7 行声明了用于记录可遇对象中的可遇点的二维数组，同 MyDrawable 的不可通过矩阵一样，可遇实体中的可遇点也需要额外记录，meetableMatrix 中记录的同样是可遇部分相对于可遇实体左下角的坐标。
- 代码第 8～9 行声明了两个 Bitmap 对象的引用，分别代表对话框的背景图片和对话框按钮的背景图片，当英雄遇到可遇实体时，是通过对话框的方式来交互的。
- 代码第 15 行为抽象方法 drawDialog，因为不同的可遇实体所执行的逻辑不同，所以要绘制的对话框也不尽相同，各个子类需要独立实现该方法。
- 代码第 16 行为 drawString 方法，该方法负责将接收到的字符串对象绘制到指定的 Canvas 对象中，MyMeetableDrawable 子类在实现 drawDialog 方法时会调用到 drawString 方法。
- 代码第 17 行对 View.OnTouchListener 接口的 onTouch 方法的实现，在 MyMeetableDrawable 类中对此只提供了空实现，该方法将在子类中进行具体实现。

16.10.3 森林类——ForestDrawable 的开发

前面的小节对 MyDrawable 和 MyMeetableDrawable 类进行了介绍，游戏中的真正与英雄相遇的是 MyMeetableDrawable 的子类对象，如 ForestDrawable、PalaceDrawable 等。

不同的可遇实体执行不同的业务逻辑，但其都是通过重写 MyMeetableDrawable 类中的抽象方法 drawDialog 和实现 View.OnTouchListener 接口的 onTouch 方法来完成与玩家交互的。本小节将以 ForestDrawable 为例来说明地图可遇实体的开发过程，其步骤如下。

（1）ForestDrawable 在地图中显示为森林，当英雄与 ForestDrawable 相遇时可以消耗一定的体力值来伐木，伐木会获得一定数额的金钱。为实现这些功能，在 ForestDrawable 类中需要另外声明一些成员变量，这些成员变量的代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 ForestDrawable. java。

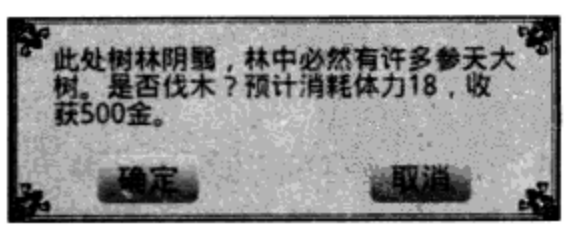
```
1    String dialogMessage[] = {//对话框的提示信息，第一个是可以狩猎，第二个是不可以狩猎显示的
2          "此处树林阴翳，林中必然有许多参天大树。是否伐木？预计消耗体力 xx，收获 yy 金。",
3          "此处是个伐木的好地方，怎奈人困马乏，体力不足，只好改日再来！"
4    };
5    int result;                    //记录英雄可以获得的收益
6    int status;                    //状态，0-显示是否伐木的选择对话框，1-显示不能伐木的提示对话框
```

- 代码第 1 行声明了一个 String 数组，数组中每个元素代表不同情况下对话框需要显示的文本信息。代码第 2 行中的字符串中包含了“xx”和“yy”，这两处将会在绘制对话框之前被计算出的消耗体力值和收获金钱数所替换掉。
- 代码第 5 行声明了用于存储英雄可获得的收益的成员变量，在 ForestDrawable 类中，该成员变量代表收获的金钱数。
- 代码第 6 行声明的 status 代表对话框应处于的状态，0 表示向玩家显示是否伐木的对话框，1 表示向玩家显示不能伐木的对话框。

（2）接下来需要实现父类的抽象方法 drawDialog，在 ForestDrawable 类中，drawDialog 方法代码如下。

```
1    public void drawDialog(Canvas canvas,Hero hero) {     //方法：绘制对话框
2         String showString = null;                         //需要显示到对话框中的字符串
3         tempHero = hero;
4         canvas.drawBitmap(bmpDialogBack, 0, DIALOG_START_Y, null);    //先画背景
5         Skill skill = hero.heroSkill.get(LUMBER);        //获得英雄的伐木技能
6         result = skill.calculateResult(hero);            //计算英雄的收益
7         if(result == -1){                                //体力值不够了
8              status = 1;                                 //设置对话框状态
9              showString = dialogMessage[status];         //设置对话框显示的文本信息
10        }else{                                           //如果体力值够
11             status = 0;                                 //设置对话框状态
12             showString = dialogMessage[status];         //设置对话框显示的文本信息
13             showString = showString.replaceAll("xx", skill.strengthCost+"");
                                                           //替换消耗体力的字符串
14             showString = showString.replaceFirst("yy", result+"");
                                                           //替换掉预计收获金钱数的字符串
15        }
16        drawString(canvas, showString);                  //在对话框中绘制信息
17             //画确定按钮
18        canvas.drawBitmap(bmpDialogButton, DIALOG_BTN_START_X, DIALOG_BTN_START_Y, null);
19        Paint paint = new Paint();                       //创建画笔对象
20        paint.setARGB(255, 42, 48, 103);                 //设置画笔颜色
21        paint.setAntiAlias(true);                        //设置抗锯齿
22        paint.setTypeface(Typeface.create((Typeface)null,Typeface.ITALIC));
                                                           //设置字体
23        paint.setTextSize(18);                           //设置字号
24        canvas.drawText("确定",                          //绘制“确定”文字
25                  DIALOG_BTN_START_X+DIALOG_BTN_WORD_LEFT,
26                  DIALOG_BTN_START_Y+DIALOG_WORD_SIZE+DIALOG_BTN_WORD_UP,
27                  paint);
28        //画取消按钮
29        if(status == 0){                                 //查看是否需要画第二个取消按钮
30             canvas.drawBitmap(bmpDialogButton, DIALOG_BTN_START_X
31                       +DIALOG_BTN_SPAN, DIALOG_BTN_START_Y, null);
32             canvas.drawText("取消",                     //绘制“取消”文字
33                       DIALOG_BTN_START_X+DIALOG_BTN_SPAN+DIALOG_BTN_WORD_LEFT,
34                       DIALOG_BTN_START_Y+DIALOG_WORD_SIZE+DIALOG_BTN_WORD_UP,
35                       paint);
36        }}
```

- 代码第 6 行调用了英雄伐木技能的计算收益的方法，该方法将在随后的内容中介绍。
- 代码第 7～15 行通过判断伐木技能的 calculateResult 方法的返回值来确定对话框的状态。代码第 16 行根据对话框状态绘制相应的信息。
- 代码第 18～27 行将确定按钮绘制到对话框中，首先将按钮背景图片绘制出来，然后创建一个 Paint 画笔对象并对其进行设置，最后将按钮文本绘制到对话框中。
- 代码第 28～36 行判断是否需要绘制取消按钮，因为对于状态 1（即不可伐木的状态）是不需要绘制取消按钮的。图 16-38 显示了当玩家在游戏中遇到



▲图 16-38　遇到 ForestDrawable 时显示的对话框

ForestDrawable 时显示的对话框。

（3）将对话框绘制出来后，当玩家单击按钮后，需要进行相应的处理，这些处理工作放在 onTouch 方法中进行，该方法来自接口 View.OnTouchListener。onTouch 方法代码如下。

```
1    public boolean onTouch(View v, MotionEvent event) {  //对 onTouch 方法的实现
2         int x = (int)event.getX();                      //获得单击位置的 X 坐标
3         int y = (int)event.getY();                          //获得单击位置的 Y 坐标
4         if(event.getAction() == MotionEvent.ACTION_DOWN){
5              if(x>DIALOG_BTN_START_X && x<DIALOG_BTN_START_X+DIALOG_BTN_WIDTH
6                        && y>DIALOG_BTN_START_Y
7                        && y<DIALOG_BTN_START_Y+DIALOG_BTN_HEIGHT){
                                                            //判断点下的是否是确定键
8                   switch(status){                         //根据状态来处理确定键按下事件
9                   case 0:                                 //有木可伐时玩家确认伐木
10                       tempHero.heroSkill.get(LUMBER).useSkill(result);
                                                            //调用技能类的 useSkill 方法
11                       break;
12                  case 1:break;                           //无木可伐时玩家确认放弃
13                  }
14                  recoverGame();                          //恢复游戏状态
15             }else if(x>DIALOG_BTN_START_X+DIALOG_BTN_SPAN
16                  && x<DIALOG_BTN_START_X+DIALOG_BTN_SPAN+DIALOG_BTN_WIDTH
17                  && y>DIALOG_BTN_START_Y
18                  && y<DIALOG_BTN_START_Y+DIALOG_BTN_HEIGHT){//判断点下的是否是取消键
19                  recoverGame();
20             }}
21        return true;
22   }
```

- 代码 5～14 行为当玩家单击确定键时的处理代码，首先判断此时对话框的状态，如果为 0（即玩家确认伐木），则调用伐木技能的 useSkill 方法，如果为 1（玩家不能伐木时按下确认），则什么也不做。代码第 14 行为 recoverGame 方法，该方法用于恢复游戏状态。
- 代码第 15～20 行为当玩家点即取消键时的处理代码，玩家按下取消键的情况只有一种，即可以伐木时选择不伐木。代码第 19 行也调用了 recoverGame 方法来复游戏状态。

16.10.4 可遇实体对象的调用流程

在 16.10.3 节以 ForestDrawable 为例介绍了 MyMeetableDrawable 子类对象的开发，本小节将介绍可遇实体对象在游戏中的调用流程。英雄从检查是否与可遇实体相遇到结束与可遇实体的交互之间要经过如下的流程。

- 调用 HeroGoThread 类的 checkIfMeet 方法判断是否相遇。
- 如果英雄与某个可遇实体相遇，用可遇实体对象的监听方法替换掉 GameView 的 View.OnTouchListener 监听器，并进行设置让 GameView 调用可遇实体对象的 drawDialog 方法。
- 可遇实体对象与玩家交互完毕后，调用可遇实体对象的 recoverGame 方法恢复游戏。

接下来介绍上述的流程在代码中的具体实现，其步骤如下。

（1）HeroGoThread 类的 checkIfMeet 方法的开发，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 HeroGoThread.java。

```
1    public boolean checkIfMeet(){          //方法：检查英雄是否与可遇实体对象相遇
2         MyMeetableDrawable mmd = gv.meetableChecker.check(hero);
3         if(mmd != null && mmd!=gv.previousDrawable){
                                            //如果碰到了可遇物，且该可遇物不是上次遇到的那个
4              gv.setOnTouchListener(mmd);      //设置 GameView 的 OnTouchListener
5              gv.currentDrawable = mmd;        //设置 GameView 的当前可遇实体
6              gv.previousDrawable = mmd;       //记录当前的可遇实体为遇过的
7              if(gv.activity.isEnvironmentSound){gv.playSound(1, 0);}
                                                //如有需要，播放声音
8              return true;                     //返回 true
9         }
10        gv.previousDrawable = mmd;            //将 mmd 记录到 GameView 的成员变量中
11        return false;
12   }
```

- 代码第 2 行调用了 GameView 类中 MeetableLayer 的 check 方法，将英雄停留位置的左右两边的可遇实体对象返回。代码第 3 行对返回值进行判断，如果其不为空，且不是上次遇到的可遇实体，则执行代码第 4～8 行的内容。
- 代码第 4～8 行首先设置 GameView 的 OnTouchListener 为可遇实体对象 mmd，然后将 mmd 赋值给 GameView 中的 currentDrawable 和 previousDrawable。currentDrawable 用于记录当前的可遇实体对象，previousDrawable 用于记录上次的可遇实体对象。

提示

在 GameView 中设置 currentDrawable 和 previousDrawable 的目的是防止两次遇到的可遇实体相同。假设游戏中的森林所占的行列数较多，英雄两次掷骰子走动后仍然与该可遇实体相遇，如果不做如上的设置，将会触发两次伐木事件，这并不符合游戏逻辑。

（2）在 GameView 中绘制可遇实体的对话框，onDraw 方法为 GameView 的绘制方法。可遇实体对象的 drawDialog 也是在 onDraw 方法调用的。该代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 ForestDrawable. java。

```
1    if(currentDrawable != null){                    //判断当前的可遇实体对象是否为 null
2         currentDrawable.drawDialog(canvas, hero);  //调用可遇实体对象的 drawDialog 方法
3    }
```

（3）对话框被显示，监听器被替换后，玩家就可以与可遇实体对象进行交互了，交互完毕之后，通过调用可遇实体对象的 recoverGame 方法来恢复游戏，该方法代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 ForestDrawable.java。

```
1    public void recoverGame(){                      //方法：返还监听器，恢复游戏状态
2         tempHero.father.setOnTouchListener(tempHero.father);//返还监听器
```

```
3            tempHero.father.setCurrentDrawable(null);          //置空记录引用的变量
4            tempHero.father.setStatus(0);                      //设置 GameView 为待命状态
5            tempHero.father.gvt.setChanging(true);             //骰子转起来
6        }
```

说明 在 recoverGame 方法中首先将监听器重新设置为 GameView 自身，然后将 GameView 类的 currentDrawable 引用置空，并将游戏状态设置为 0（待命态），最后通过设置 GameViewThread 类的标志位让骰子重新变换起来。

16.11 英雄技能模块的开发

在 16.10 节介绍 ForestDrawable 类的开发时，曾经提到了英雄的技能，并调用了技能对象的方法，本节就来介绍英雄的技能模块的开发。该模块涉及的类有 Skill、FarmingSkill、SuiXinBuSkill 等，其中 Skill 为所有技能类的基类。

16.11.1 技能抽象类——Skill 的开发

Skill 类为抽象类，每个子类在继承 Skill 类时都需重写父类的抽象方法，Skill 类的代码框架如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 Skill.java。

```
1    package wyf.ytl;                                                    //声明包语句
2    import java.io.Serializable;                                        //引入相关类
3    public abstract class Skill implements Serializable{
4        private static final long serialVersionUID = 5878577464788782434L; //持久化版本号
5        int id;                                          //技能 id，惟一标识技能
6        String name;                                     //技能的名字
7        int proficiencyLevel;                            //技能熟练度
8        int strengthCost;                                //每次释放消耗的体力
9        int basicEarning;                                //基本收入，用于计算英雄的总收入
10       int tempProficiency;                             //临时熟练度，积累可增加技能熟练度
11       int proficiencyToUpgrade;                        //技能升级需要的临时熟练度
12       int skillType;                                   //技能类型，0-生活技能，2-特殊技能
13       Hero hero;                                       //所属英雄的引用
14       public Skill(){}                                 //无参数构造器
15       public Skill(int id,String name,int basicEarning,int skillType,Hero hero){}
                                                          //有参数构造器
16       public abstract int calculateResult();           //抽象方法：返回英雄应得的收益，
                                                          //如果无收益则返回-1；
17       public abstract void useSkill(int skillEarning);     //抽象方法：施放技能
18       ……//此处省略成员变量的 get 和 set 方法，读者可自行查阅随书光盘
19   }
```

- 代码第 5 行声明了 Skill 对象的 ID，每种技能都具有惟一的 ID，其值取自 ConstantUtil 中的常量，如 LUMBER 代表伐木技能的 ID 等。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 ConstantUtil.java。

```
1    public static final int LUMBER = 0;          //代表伐木技能，作为技能 HashMap 的键
2    public static final int FISHING = 1;         //代表打渔技能，作为技能 HashMap 的键
3    public static final int FARMING = 2;         //代表农耕技能，作为技能 HashMap 的键
```

```
4    public static final int MINING = 3;                    //代表采矿技能，作为技能 HashMap 的键
5    public static final int SUI_XIN_BU = 4;                //代表随心步，作为技能 HashMap 的键
6    public static final int HUI_TOU_SHI_AN = 5;            //代表回头是岸，作为技能 HashMap 的键
7    public static final int WU_ZHONG_SHENG_YOU = 6;        //代表无中生有，作为技能 HashMap 的键
```

- 代码第 7 行为记录技能的熟练度的成员变量，随着技能熟练度的提高，英雄获得的收益也会增加。代码第 10 行为记录临时熟练度的成员变量，当临时熟练度达到代码第 11 行定义的 proficiencyToUpgrade 时，技能会升级。
- 代码第 12 行为技能种类，其中 0 代表生活技能，如伐木、采矿等。2 代表普通技能，如随心步、回头是岸等。
- 代码第 15 行为 Skill 类的有参数构造器，在该构造器中对主要的成员变量进行初始化。
- 代码第 16～17 行为抽象成员方法 calculateResult 和 useSkill 的声明。

16.11.2　伐木技能类——LumberSkill 的开发

本小节将以 LumberSkill 为例，介绍如何开发继承自 Skill 类的技能子类对象。LumberSkill 对象代表伐木技能，实现 LumberSkill 主要是对其父类的抽象方法的重写，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 LumberSkill.java。

```
1    public int calculateResult() {                          //方法：计算英雄收益，若无收益返回 0
2         if(hero.strength < strengthCost){                  //如果英雄的体力不够使用本次技能
3              return -1;
4         }else{                                             //利用公式计算英雄收益
5              int result = GameFormula.getSkillEearning(proficiencyLevel, basicEarning);
6              return result;
7         }}
8    public void useSkill(int skillEarning) {            //方法：使用技能
9         hero.setTotalMoney(hero.getTotalMoney() + skillEarning); //英雄金钱增加
10        hero.setStrength(hero.getStrength() - strengthCost);//英雄体力减少
11        tempProficiency += PROFICIENCY_INCREMENT;            //英雄临时熟练度增加
12        if(tempProficiency == proficiencyToUpgrade && proficiencyLevel<SKILL_LEVEL_
          MAX){
                                                               //可以升级
13             proficiencyLevel += 1;                          //熟练度等级加 1
14             tempProficiency = 0;                            //临时熟练度置零
15             strengthCost -= STRENGTH_COST_DECREMENT;        //消耗的体力值减少
16             proficiencyToUpgrade+=PROFICIENCY_UPGRADE_SPAN;//下一级升级上限增加
17        }}
```

- 代码第 1～7 行为 calculateResult 方法的代码，在该方法中首先判断英雄当前的体力值是否足够施放技能，如果不够则返回 0，否则调用 GameFormula 的静态方法来计算英雄通过这次伐木能够收获的金钱。
- 代码第 8～17 为 useSkill 方法的代码，在该方法中将修改与该技能有关的英雄数据，如金钱增加、体力值减少等，同时还增加技能的临时熟练度。
- 代码第 12～17 行对技能的临时熟练度进行判断，如果其值达到升级的上限且还没有升到最大等级，则对技能进行升级操作。

说明　伐木技能的使用流程可以参考 16.10.3 节对 ForestDrawable 类中的 drawDialog 和 onTouch 方法代码的介绍，其他生活类技能的使用流程与此类似。

16.11.3 随心步技能类——SuiXinBuSkill 的开发

在 16.11.2 节介绍了伐木技能 LumberSkill 的开发，伐木技能属于生活技能，游戏中英雄还具有其他的特殊技能，如随心步技能可以让英雄前进指定的地图格子数，回头是岸技能可以让英雄掉转方向，无中生有技能可以让英雄凭空获得一些士兵和金钱等。

本小节将以随心步技能为例来说明特殊技能的开发，特殊技能同样继承自 Skill 类，需要对父类的抽象方法进行实现，SuiXinBuSkill 类的代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 SuiXinBuSkill.java。

```
1    public int calculateResult() {                          //方法：计算技能收益，在此为判断是否能够使用
2         if(hero.strength < strengthCost){                  //判断英雄的体力值够不够
3              return -1;                                    //返回-1 表示不能够使用该技能
4         }
5         return 0;                                          //返回 0 表示能够使用该技能
6    }
7    public void useSkill(int skillEarning) {                //方法：施放技能
8         hero.setStrength(hero.getStrength() - strengthCost);//减少体力
9         int steps = hero.father.suiXinBu;                        //获取要走几步
10        hero.startToGo(steps);                                   //英雄开始走
11        hero.father.setStatus(1);                                //设置 GameView 状态
12   }
```

- 代码第 1～6 行为 calculateResult 方法的代码，由于随心步技能不会产生收益，所以该方法在此的功能是判断英雄能否施放，返回 0 表示能够施放该技能，返回-1 表示不能施放。
- 代码第 7～12 行为 useSkill 方法的代码，施放随心步技能后首先减少英雄的体力值，然后获得 Ga meView 类中用于记录随心步步数的成员变量的值，最后调用 Hero 类的 startToGo 方法激活 HeroGoThread，使英雄开始移动，同时还需要将 GameView 的状态设置为 1（英雄移动状态）。

16.12 游戏提示模块的开发

在 16.7.2 节介绍 HeroBackDataThread 时曾经提到过，游戏中发生粮草危机、敌军突袭等事件时，需要提示玩家进行相应处理，这里就使用到了游戏提示模块的功能，本节就来介绍游戏提示模块的开发。其中涉及的类主要有 GameAlert 及其子类 FoodAlert、WarAlert、PlainAlert 及 GameOverAlert。

16.12.1 提示模块抽象类——GameAlert 的开发

游戏提示模块与可遇实体模块的开发比较类似，在该模块中，抽象类 GameAlert 为所有游戏提示类的父类，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 GameAlert.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import static wyf.ytl.ConstantUtil.*;                     //引入相关类
3    ……//此处省略部分引入相关类的代码，读者可自行查阅随书光盘
4    import android.view.View;                                 //引入相关类
5    public abstract class GameAlert implements View.OnTouchListener{
```

```
6          Bitmap bmpDialogBack;                                  //对话框背景
7          Bitmap bmpDialogButton;                                //对话框按钮
8          GameView gameView;                                     //GameView 对象引用
9          public GameAlert(GameView gameView,Bitmap bmpDialogBack,Bitmap bmpDialog-
           Button){}
10         public abstract void drawDialog(Canvas canvas);//抽象方法:用于获得监听后绘制对话框
11         public void drawString(Canvas canvas,String string){
                                                                //方法:绘制给定的字符串到对话框上}
12     }
```

- 代码第 6～7 行声明了游戏提示中用到的图片对象引用，游戏中的提示也是通过对话框的形式显示到屏幕上的。
- 代码第 10 行为抽象方法 drawDialog，子类除了要实现该方法外，还需要实现 View.OnTouchListener 接口的 onTouch 方法。
- 代码第 11 行为在对话框上绘制文本信息的方法，该方法与 MyMeetableDrawable 类中的 drawString 方法是一样的。

16.12.2　点击确定按钮显示的信息类——PlainAlert 的开发

前面的小节介绍了 GameAlert 类，本小节将以 PlainAlert 类为例来介绍游戏提示对象的开发，PlainAlert 是最简单的一种 GameAlert，其功能是向玩家显示提示信息和一个确定按钮，玩家按下确定按钮后返回到正常游戏的状态。PlainAlert 类的代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 GameAlert.java。

```
1    package wyf.ytl;                                           //声明包语句
2    import android.view.MotionEvent;                           //引入相关类
3    ……//此处省略部分引入相关类的代码，读者可自行查阅随书光盘
4    import android.view.View;                                  //引入相关类
5    public class PlainAlert extends GameAlert{
6         String alertMessage;                                  //要显示的消息
7         public PlainAlert(GameView gameView,String alertMessage,Bitmap bmpDialogBack,
8                                                               Bitmap bmpDialogButton){
                                                                //构造器}
9         public void drawDialog(Canvas canvas) {               //对父类抽象方法drawDialog的实现
10             canvas.drawBitmap(bmpDialogBack, 0, DIALOG_START_Y, null);   //先画背景
11             drawString(canvas, alertMessage);                //绘制提示信息
12             //画确定按钮
13             canvas.drawBitmap(bmpDialogButton,
14                        DIALOG_BTN_START_X, DIALOG_BTN_START_Y, null);
15             Paint paint = new Paint();                           //创建画笔对象
16             paint.setARGB(255, 42, 48, 103);                     //设置画笔颜色
17             paint.setAntiAlias(true);                            //设置抗锯齿
18             paint.setTypeface(Typeface.create((Typeface)null,Typeface.ITALIC));
                                                                    //设置字体
19             paint.setTextSize(18);                               //设置字号
20             canvas.drawText("确定",                              //绘制确定按钮
```

```
21                          DIALOG_BTN_START_X+DIALOG_BTN_WORD_LEFT,
22                          DIALOG_BTN_START_Y+DIALOG_WORD_SIZE+DIALOG_BTN_WORD_UP,
23                          paint
24                          );
25          }
26          public boolean onTouch(View view, MotionEvent event) { //实现 View.OnTouchListener
                                                                   //接口的方法
27              int x = (int)event.getX();                         //获得单击处的 X 坐标
28              int y = (int)event.getY();                         //获得单击处的 Y 坐标
29              if(event.getAction() == MotionEvent.ACTION_DOWN){
30                  if(x>DIALOG_BTN_START_X && x<DIALOG_BTN_START_X+DIALOG_BTN_WIDTH
31                      && y>DIALOG_BTN_START_Y
32                      && y<DIALOG_BTN_START_Y+DIALOG_BTN_HEIGHT){//单击的是确定键
33                      gameView.setStatus(0);            //设置游戏状态为 0(待命状态)
34                      gameView.setOnTouchListener(gameView);
                                                          //将监听器重新设置为 GameView
35                      gameView.currentGameAlert = null;        //置空当前游戏提示
36                  }}
37              return true;
38          }}
```

- 代码第 6 行声明 alertMessage 变量，该变量存储要显示的提示信息，将在代码第 7 行的构造器中被初始化。
- 代码第 9～25 行是对父类抽象方法 drawDialog 的具体实现，由于 PlainAlert 只是用来提示信息，所以，只需要绘制一个确定按钮即可。
- 代码第 26～38 为对 View.OnTouchListener 接口方法 onTouch 的实现，在玩家点击“确定”按钮后，通过执行第 33～35 行的代码来恢复游戏。

下面来介绍如何在程序中使用 PlainAlert，在游戏中如果显示提示信息，创建一个 PlainAlert 对象，然后将其引入赋值给 GameView 类的成员变量 currentGameAlert。在 GameView 的 onDraw 方法中，将通过以下的代码绘制 PlainAlert。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 GameView.java。

```
1    if(currentGameAlert != null && currentDrawable == null){ //判断是否需要绘制游戏提示
2        currentGameAlert.drawDialog(canvas);                 //绘制对话框
3        setOnTouchListener(currentGameAlert);                //设置 GameView 的监听器
4    }
```

说明　在游戏中可遇实体和游戏提示都需要显示及替换 GameView 的监听器，本游戏人为规定可遇实体的优先级比游戏提示的高。因此，在判断是否需要绘制游戏提示时，首先要保证记录当前可遇实体对象的 currentDrawable 引用为 null。

16.12.3　显示粮草危机信息类——FoodAlert 的开发

在游戏提示模块中，除了 PlainAlert 可以在多种场合使用显示提示信息外，其他的 GameAlert 子类都只负责特定的提示信息的显示，如 FoodAlert 用在发生粮草危机的场合，WarAlert 用在敌军

来袭的场合，GameOverAlert 用在英雄破产，游戏结束的场合。本小节将以 FoodAlert 为例介绍特定游戏提示的开发。其步骤如下。

（1）首先开发出 FoodAlert 类的代码框架，代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 FoodAlert.java。

```
1    package wyf.ytl;                                          //声明包语句
2    import android.view.MotionEvent;                          //引入相关类
3    ……//此处省略部分引入相关类的代码，读者可自行查阅随书光盘
4    import android.view.View;                                 //引入相关类
5    public class FoodAlert extends GameAlert{
6          CityDrawable city;                                  //记录是哪个城池粮草危机
7          String alertMessage = "刚才 xx 城守将派人来报,说粮草已经不多了,是否划拨粮草?";
                                                               //提示消息
8         public FoodAlert(GameView gameView,CityDrawable city,
9                              Bitmap bmpDialogBack,Bitmap bmpDialogButton){//构造器}
10        public void drawDialog(Canvas canvas) {}//对父类抽象方法的实现
11        public boolean onTouch(View v, MotionEvent event) {}
                                                  //对接口 View. OnTouchListener 方法的实现
12   }
```

说明　代码第 5 行声明了用于记录粮草危机城市的 CityDrawable 对象引用，该成员变量将在第 8 行的构造器中被初始化。代码第 7 行为将要显示的提示信息，该字符串中的“xx”部分将会在显示之前被替换为发生粮草危机的城市名称。

（2）接下来将对父类的抽象方法 drawDialog 进行具体实现，其代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 FoodAlert.java。

```
1    public void drawDialog(Canvas canvas) {                          //方法：绘制对话框
2          canvas.drawBitmap(bmpDialogBack, 0, DIALOG_START_Y, null);    //画对话框背景
3          alertMessage=alertMessage.replaceFirst("xx", city.getCityName());
                                                                      //替换成实际的城市名称
4          drawString(canvas, alertMessage);                          //画提示文本
5          Paint paint = new Paint();                                 //创建画笔
6          paint.setTextSize(DIALOG_WORD_SIZE);                       //设置字体大小
7          paint.setAntiAlias(true);                                  //设置抗锯齿
8          paint.setARGB(255, 42, 48, 103);                           //设置字体颜色
9          //画划拨按钮
10         canvas.drawBitmap(bmpDialogButton, DIALOG_BTN_START_X, DIALOG_BTN_START_Y, null);
11         canvas.drawText("划拨",                                    //画按钮上的文字
12                   DIALOG_BTN_START_X+DIALOG_BTN_WORD_LEFT,
13                   DIALOG_BTN_START_Y+DIALOG_WORD_SIZE+DIALOG_BTN_WORD_UP,
14                   paint);
15         //画忽略按钮
16         canvas.drawBitmap(bmpDialogButton, DIALOG_BTN_START_X+DIALOG_BTN_SPAN,
17                                              DIALOG_BTN_START_Y, null);
18         canvas.drawText("忽略",                                    //画按钮上的文字
19                   DIALOG_BTN_START_X+DIALOG_BTN_SPAN+DIALOG_BTN_WORD_LEFT,
20                   DIALOG_BTN_START_Y+DIALOG_WORD_SIZE+DIALOG_BTN_WORD_UP,
21                   paint);
22   }
```

- 代码第 3 行将 alertMessage 字符串中的“xx”内容替换为实际的城市名称。
- 代码第 10～21 行分别在对话框中绘制了“划拨”和“忽略”按钮。

（3）当玩家点下 FoodAlert 对话框上的按钮时，onTouch 方法将会对单击事件进行处理，该方

法的代码如下。

```
1    public boolean onTouch(View v, MotionEvent event) {        //方法：处理单击屏幕事件
2         if(event.getAction() == MotionEvent.ACTION_DOWN){     //事件为点下屏幕
3              int x = (int)event.getX();                       //获取屏幕按下的 x 坐标
4              int y = (int)event.getY();                       //获取屏幕按下的 y 坐标
5              if(x>DIALOG_BTN_START_X && x<DIALOG_BTN_START_X+DIALOG_BTN_WIDTH
6                   && y>DIALOG_BTN_START_Y
7                   && y<DIALOG_BTN_START_Y+DIALOG_BTN_HEIGHT){//单击的是划拨键
8                   gameView.setStatus(97);                     //弹出城池管理界面
9                   gameView.setCurrentGameAlert(null);
10                  gameView.setOnTouchListener(gameView);     //监听器重新设置为 GameView
11             }else if(x>DIALOG_BTN_START_X+DIALOG_BTN_SPAN
12                  && x<DIALOG_BTN_START_X+DIALOG_BTN_SPAN+DIALOG_BTN_WIDTH
13                  && y>DIALOG_BTN_START_Y
14                  && y<DIALOG_BTN_START_Y+DIALOG_BTN_HEIGHT){    //点下的是忽略键
15                  gameView.setCurrentGameAlert(null);
16                  gameView.setStatus(0);                      //设置游戏状态为待命
17                  gameView.setOnTouchListener(gameView);     //监听器设置为 GameView
18             }}
19        return true;
20   }
```

- 代码第 5～10 行为当玩家单击“划拨”按钮时的处理代码，由于点下划拨按钮需要弹出城池管理界面，所以，需要将游戏设置为相应的状态，同时将 GameView 中记录当前游戏提示的成员变量 currentGameAlert 赋值为 null，然后将监听器重新设置为 GameView。
- 代码第 11～18 行为当玩家单击“忽略”按钮时的处理代码，单击忽略按钮后执行的代码主要职责是恢复游戏状态。

16.12.4 辅助线程 HeroBackDataThread 中对 FoodAlert 的调用

本书在 16.7.2 节对 HeroBackDataThread 做过简单的介绍，该线程主要负责定时修改英雄的一些后台数据，并根据一定规则产生粮草危机、敌军来袭等事件，这些事件的发生都需要用到 GameAlert，本小节就以 FoodAlert 为例来说明 GameAlert 在游戏中的用法。

在 HeroBackDataThread 的 run 方法中，对于城池粮草的处理是按照如下的流程来进行的。

- 遍历英雄拥有的城池，对每个城池的粮草进行衰减。
- 检查衰减后剩余的粮食，是否小于最小值。如果小于最小值，则判断这是该城池的第几次粮草警报，如果是第二次，则新建一个 PlainAlert 通知玩家该城池已不攻自破。
- 如果这是该城池的第一次粮草警报，则创建一个 FoodAlert 对象提示玩家是否进行处理，并将该城池记录下来。

下面来看 run 方法中对上述逻辑的具体实现代码如下。

代码位置：见随书光盘中源代码/第 16 章/ HDZG/src/wyf/ytl 目录下的 HeroBackDataThread. java。

```
1    public void run(){                                  //HeroBackDataThread 线程的 run 方法
2         while(flag){
```

```
3                foodCount++;
4                if(foodCount == 5){                    //满足计数次数就进行粮食衰减
5                     foodCount = 0;
6                     hero.setFood((int)(hero.getFood()*0.95)); //按比例减少英雄的粮食
7                     ArrayList<CityDrawable> cityList = hero.getCityList();
8                     for(CityDrawable city:cityList){ //遍历英雄城池列表，减少各城池的粮食
9                          city.food -= (int)(city.food*0.15f); //减少城池的粮草
10                         if(city.food <= MIN_FOOD){  //判断城中粮草是否小于某个值
11                              if(ignoredCity == city){   //如果这是该城池的第二次粮草警报
12                                   GameView gv = hero.father;
13                                   city.setBackToInit(); //恢复城市的信息到默认
14                                   String alertMessage = "由于你没有及时输送粮草，"+city.
                                     getCityName()
15                                                  +"由于粮草危机已经不攻自破,将领和老百姓已经
                                                    归顺了"
16                                                  +COUNTRY_NAME[city.getCountry()]+"，不再属
                                                    于你了。";
17                                   PlainAlert pa = new PlainAlert(gv, alertMessage,
                                                                   //创建 PlainAlert 对象
18                                                  GameView.dialogBack, GameView.
                                                    dialogButton);
19                                   hero.father.setCurrentGameAlert(pa);  //设置为当前提示
20                                   emptyCity = city;     //记录失去的城池
21                              }
22                              else{                      //如果是第一次出现粮草危机
23                                   FoodAlert fa = new FoodAlert(hero.father, city,
                                     GameView. dialogBack,
24                                                 GameView.dialogButton);
25                                   hero.father.setCurrentGameAlert(fa);
26                                   ignoredCity = city;   //将第一次粮草危机的城池记录下来
27                              }}}
28                    if(emptyCity != null){                //删掉因粮草危机而造成的空城
29                         hero.cityList.remove(emptyCity);
30                         hero.father.allCityDrawable.add(emptyCity);
31                         emptyCity = null;
32                    }}
33               ……//此处省略 run 方法的其他逻辑代码，读者可自行查阅随书光盘
34               try{Thread.sleep(sleepSpan);}              //休眠一段时间
35               catch(Exception e){e.printStackTrace();}   //打印捕获异常
36          }}
```

- 代码第 3 行为对计数器自加操作，当计数器满 5 次时才对城池的粮草进行衰减。
- 代码第 8～27 行为遍历英雄的城池列表，对每座城池的粮草进行衰减处理的代码，代码第 17 行创建了一个 PlainAlert 对象来提示玩家该城池已经失去。代码第 23 行创建了一个 FoodAlert 对象来提示玩家对粮草告急城池进行处理。
- 代码第 19 行和 25 行为将创建的 GameAlert 对象引用赋值给 GameView 中，用于记录当前 GameAlert 对象的成员变量。
- 代码第 28～32 行为将已经有过两次粮草警报的城池从英雄拥有的城池列表中移除，同时将该城池添加到用于存放英雄城池之外的城池列表中。

16.13 游戏的优化与改进

到此为止，本游戏的开发已经基本完成，游戏中各个模块的功能也都得到实现。但是游戏中仍

然存在一些可以优化和提升的地方，现将这些可改进的地方总结如下。

- 丰富故事情节

本游戏将故事背景设在了战国时代，因此，可以将历史上的重大事件融入到故事中，玩家可以通过对这些事件的不同反应来影响游戏的进程。

- 多处存档

游戏中只提供了一个保存点，有兴趣的读者可以在此进行改进增加几个保存点供玩家选择。这可提高游戏的合理性。

- 公式的增强

游戏中使用的计算公式大都比较简单，如技术攻击力以及防御力的公式，在真实的游戏开发中，此公式可能会非常复杂。

第 17 章　体育游戏——2D 迷你桌球

桌球是一种非常流行的娱乐活动，本章将介绍一款模拟真实的桌球游戏，其简单而流行的操作方式及好的可玩性是本章的一大亮点，下面将介绍 2D 桌球在 Android 平台上的设计与实现，使读者掌握此类游戏的开发全过程。

17.1 2D 桌球的背景及功能概述

本节将要对 2D 桌球游戏进行简单介绍，通过本节的学习，读者可以对 2D 桌球游戏有初步的认识，并了解本章开发案例的具体功能及效果，为之后游戏的开发做好准备。

17.1.1　背景概述

桌球的种类很多，包括英式桌球、美式桌球、中式斯诺克桌球、法式桌球及开伦式桌球等，其中英式桌球中的斯诺克桌球是民办流行的主流桌球项目之一。

桌球游戏的界面比较简单，没有复杂的游戏背景。虽然玩法简单，但能否在规定时间内打进所有的球需要很高的球技，具有一定的挑战性，可玩性极强。

17.1.2　功能简介

上一小节介绍了 2D 桌球游戏的背景，本节将对本游戏的玩法进行简单介绍，使读者了解游戏的控制方式，为后面的学习打好基础，其具体的操作方法如下。

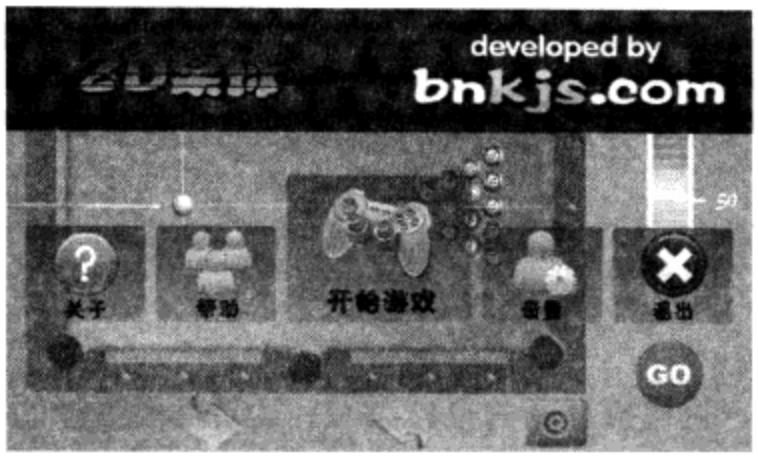
（1）安装游戏后，打开游戏，进入游戏的欢迎动画界面，如图 17-1 和图 17-2 所示。

▲图 17-1　欢迎动画界面 1

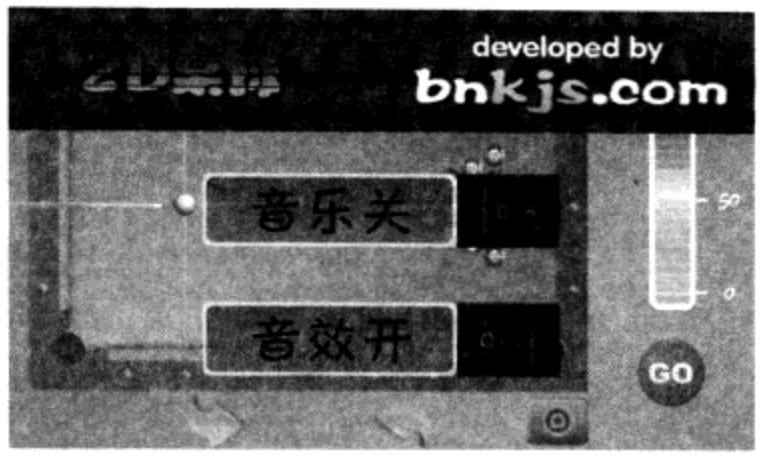
▲图 17-2　欢迎动画界面 2

（2）欢迎动画界面播放完毕后，游戏自动进入主菜单界面，如图 17-3 所示。

（3）将“设置”按钮滑动到屏幕的中间，并单击，进入设置音效界面，如图 17-4 所示。



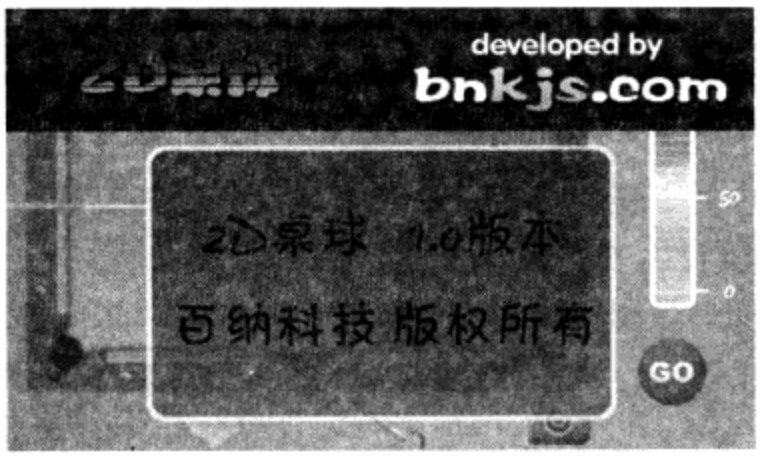
▲图 17-3　主菜单界面



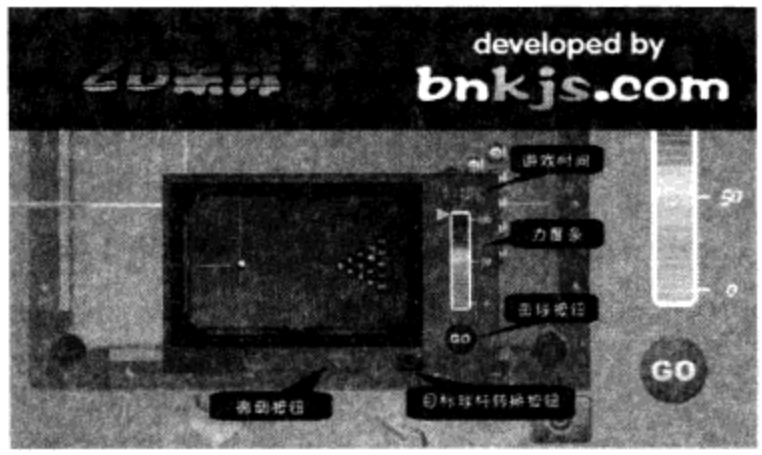
▲图 17-4　设置音效界面

（4）按下“返回”按钮，返回到游戏主菜单界面。将“关于”按钮滑动到屏幕的中间，并单击，进入关于界面，如图 17-5 所示。

（5）再次按下“返回”按钮，返回到游戏主菜单界面。将“帮助”按钮滑动到屏幕的中间，并单击，进入帮助界面，如图 17-6 所示。



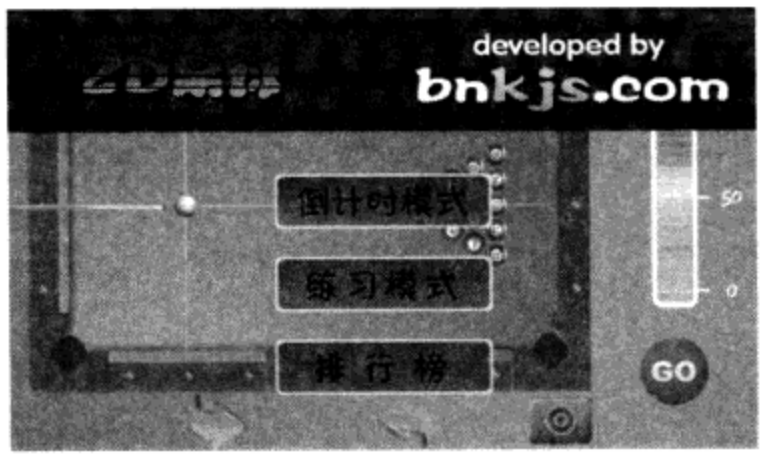
▲图 17-5　排行榜界面



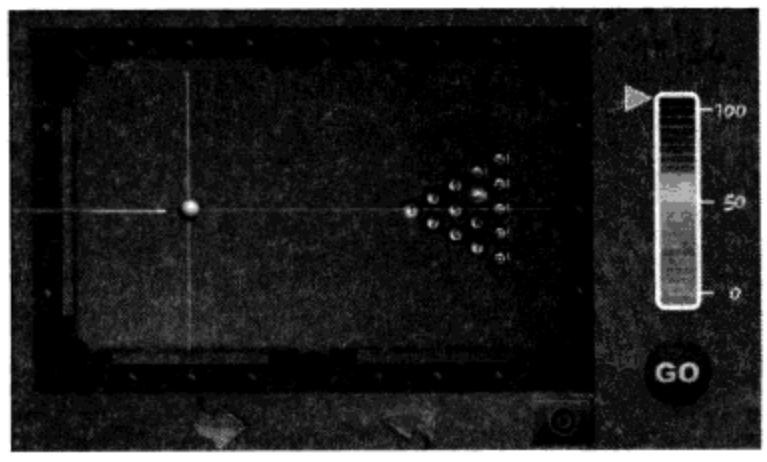
▲图 17-6　帮助界面

（6）按下“返回”按钮，返回到游戏主菜单界面。将“开始游戏”按钮滑动到屏幕的中间，并单击，进入选项界面，如图 17-7 所示。

（7）单击“倒计时模式”按钮，进入倒计时模式游戏界面，如图 17-8 所示。

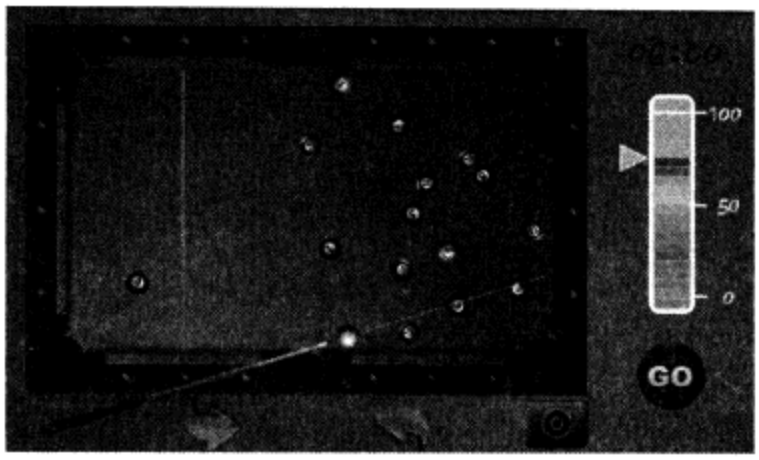


▲图 17-7　选项界面

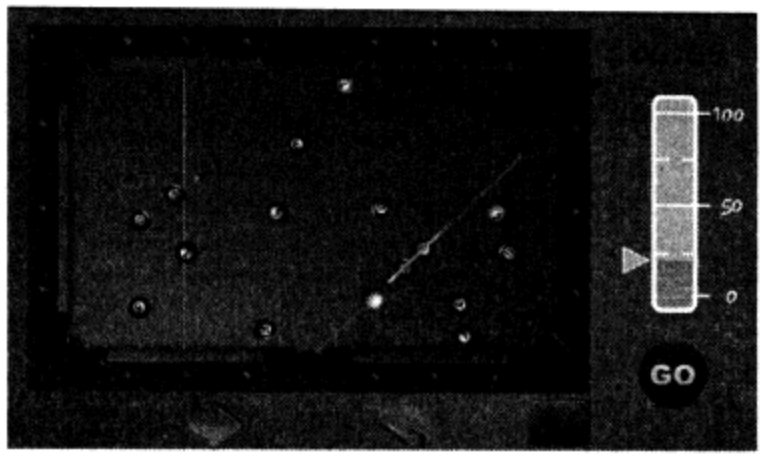


▲图 17-8　游戏界面

(8)触摸屏幕上非按钮的位置或微调按钮可以调节球杆的方向，滑动力度条可以调节击球力度，单击“go”按钮开始击球，效果如图 17-9、图 17-10 所示。



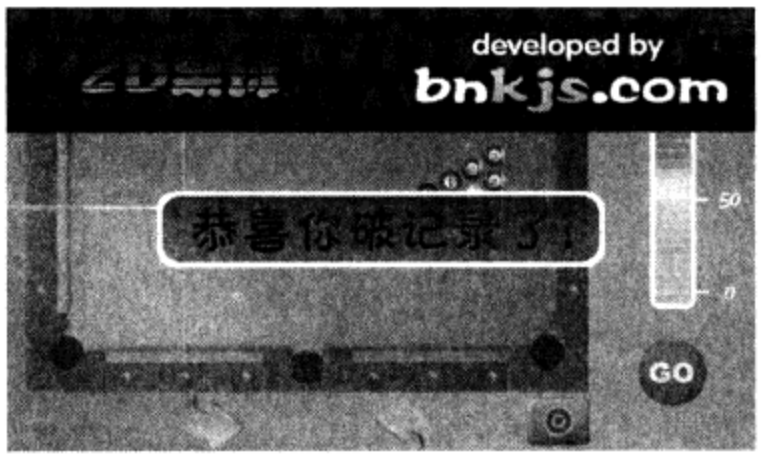
▲图 17-9　游戏进行中界面 1



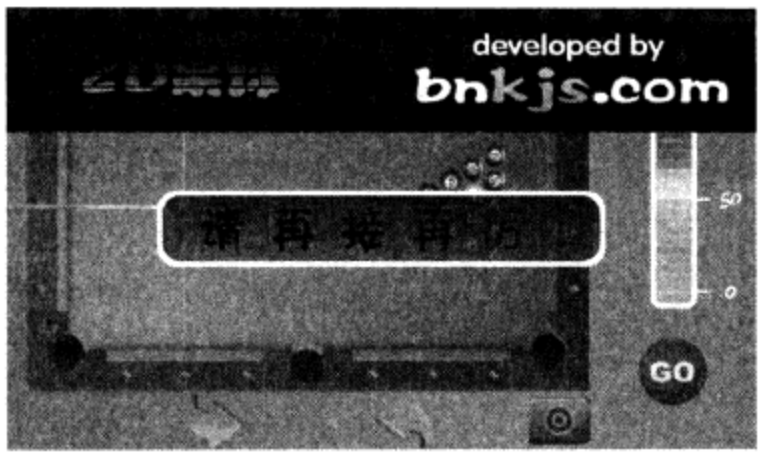
▲图 17-10　游戏进行中界面 2

(9) 游戏界面中有时间倒计时，当时间为 0 时，游戏结束。如果玩家当前得分大于排行榜中最高分，进入“打破纪录”的界面，如图 17-11 所示。

(10) 如果游戏结束后，玩家未能打破纪录，便进入“请再接再厉”的界面，如图 17-12 所示。



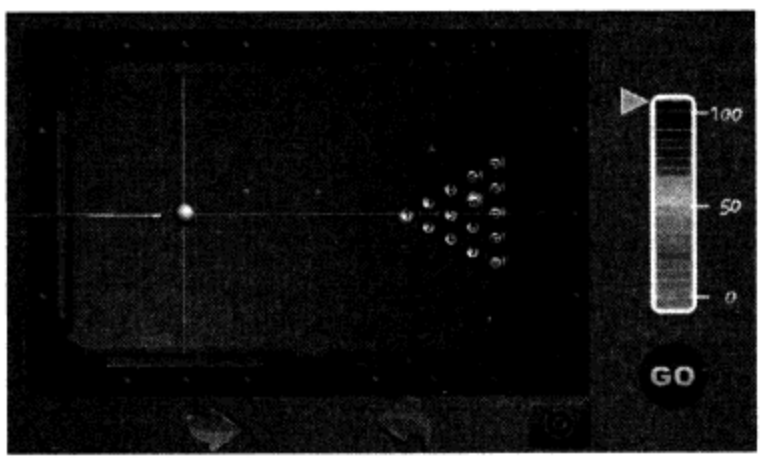
▲图 17-11　打破纪录的界面



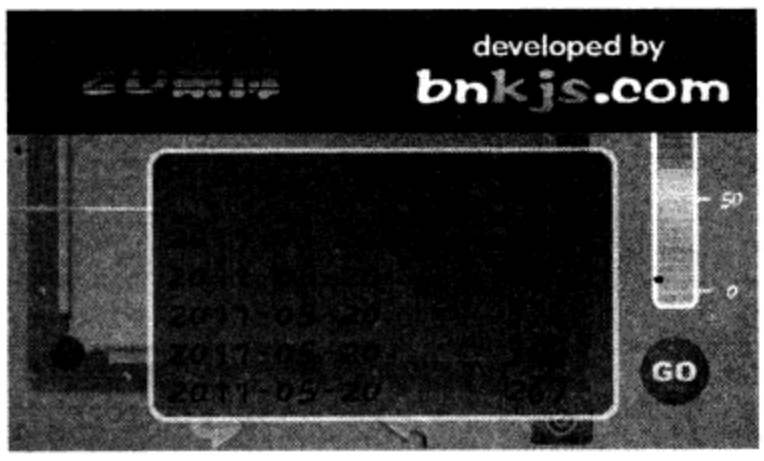
▲图 17-12　请再接再厉的界面

(11) 在选项界面中单击“练习模式”按钮，进入练习模式，如图 17-13 所示。

(12) 在选项界面中单击“排行榜”按钮，进入排行榜界面，如图 17-14 所示。



▲图 17-13　练习模式界面



▲图 17-14　排行榜界面

(13) 在主菜单界面中单击“退出”按钮或“返回”按钮，退出游戏。

提示

了解游戏的规则对游戏的开发是非常重要的，建议读者在学习本游戏开发的详细步骤之前，先运行一下光盘中的该项目。

17.2 游戏的策划及准备工作

上一节介绍了本游戏的背景及功能，本节将要介绍游戏的策划以及开发前的准备工作，这些工作虽然略显枯燥，但是在游戏开发过程中起到的作用很大。

17.2.1 游戏的策划

本小节将对本游戏的策划进行简单介绍。在真实的游戏开发中，对游戏的策划还需要更加细致、具体、全面的准备，在此只列出其中一部分。

- 运行的目标平台

本游戏的运行目标平台为 Android2.1、Android2.2 或者是更高的 Android 版本。

- 操作方式

本游戏通过触摸屏幕进行操作，玩家可通过触摸屏幕上非按钮的位置或微调按钮调节球杆的方向，滑动力度条可以调节击球力度，单击“go”按钮开始击球。

- 呈现技术

本游戏界面采用的是 2D 贴图技术，由于对桌球的动作要求非常细腻，因此贴图位置的改变比其他游戏更加微小。

- 音效设置

为了增强游戏的用户体验，本游戏添加了背景音乐和杆击球、球进洞及球之间碰撞的音效等。

17.2.2 Android 平台下游戏的准备工作

游戏开发前需要做一些必不可少的准备工作，包括游戏中用到的图片、声音素材等。本案例所用到的部分图片资源，如下表 17-1 所列。

表 17-1　　图片清单

图 片 名	大小（kB）	像素（w×h）	用　途	图 片 名	大小（kB）	像素（w×h）	用　途
aim_down.png	6.62	66×51	杆图标	help.jpg	280	800×480	界面背景
aim_up.png	6.73	66×51	目标图标	left_up.png	6.03	64×60	微调按钮
ball00.png-ball152.png	4.34	24×24	桌球	riqi.png	5	40×20	日期文字
bg.png	251	800×480	背景	number0-9.png	4	20×32	数字 0～9
bmp0.png	82.9	504×299	关于	pointer.png	3.28	30×30	力度条指针
bmp.png	53.4	504×299	文字背景	qiu_gan.png	8.35	322×13	球杆
breakmark.png	6	40×31	冒号	riqi.png	7.76	130×40	日期
choice0-3.png	21	233×60	选项按钮	ruler.png	5	184×42	力度条

续表

图 片 名	大小（kB）	像素（w×h）	用　　途	图 片 名	大小（kB）	像素（w×h）	用　　途
menu0-4.png	35.5	200×161	主菜单按钮	table0-12.png	17	265×48	球台
defen.png	8.19	130×40	得分	icon.png	6	48×48	游戏图标
on.png	17.92	80×58	开关	gang.png	3	20×20	日期分隔符
go_up.png	8	104×36	Go 按钮	highscore.png	8	108×33	文字排行榜
help0.png	144	472×300	帮助	breadmark.pn	3	13×20	日间分隔符
yinyuekai.png	13	240×64	音乐开文字	dukeb.png	20	480×320	欢迎背景 2

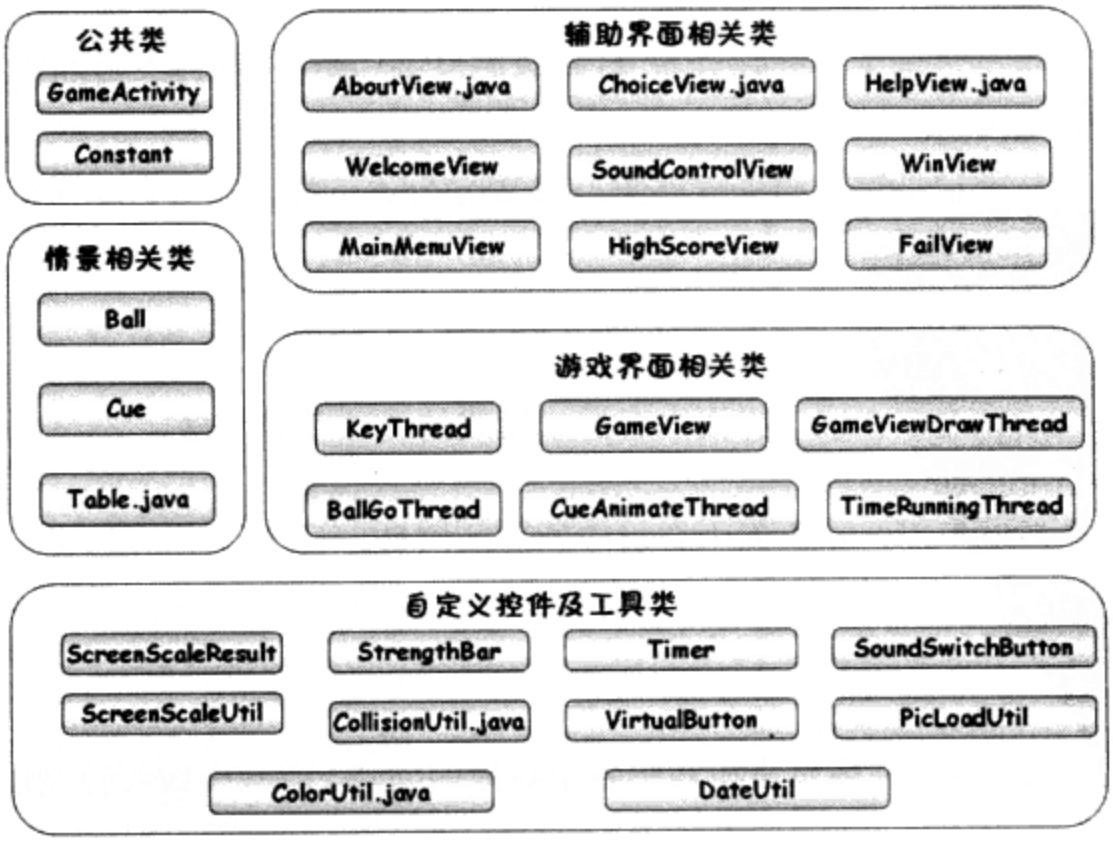
说明　表 17-1 列出了本案例中用到的部分位图资源详细信息，这些位图资源都存储在 res/drawable-nodpi 文件夹下。

17.3 游戏的架构

本节将对 2D 桌球游戏的整体架构进行介绍，让读者了解该休闲游戏的整体框架，这样读者在学习后面的内容时将更加容易理解，达到事半功倍的效果。

17.3.1　游戏的框架简介

本小节先对本游戏的框架进行介绍，使读者对游戏的架构有个整体的把握，以便于后面对每个类的具体的学习。游戏类框架如图 17-15 所示。



▲图 17-15　2D 桌球游戏类框架图

说明	游戏中的类分为公共类、辅助界面相关类、游戏界面相关类、情景相关类和自定义控件及工具类，各个模块之相互关联、相互配合便组成整个项目。其中各个类功能的简介将在下一小节中给出。

17.3.2 各个类的简要介绍

1. 公共类

- Activity 的实现类 GameActivity

该类继承并扩展了 Activity 类，是整个游戏程序的入口。

- 常量类 Constant

该类封装了游戏中用到的全部常量，便于游戏的维护和管理。

2. 辅助界面相关类

- 欢迎界面 WellcomeView

该类实现游戏的欢迎界面，主要负责欢迎界面的绘制工作。

- 主菜单界面 MainMenuView

该类为游戏主菜单界面类，主要负责主菜单中虚拟按钮绘制及监听事件等工作。

- 音效控制界面 SoundControlView

该类为游戏中音效控制界面类，主要负责游戏中背景音乐和音效的打开或关闭。

- 排行榜界面类 HighScoreView

该类为游戏的排行榜界面，主要负责从数据库中读取分数和时间信息，并显示在界面中。

- 胜利界面类 WinView

该类为玩家打破记录后进入的胜利界面，胜利界面显示一定时间后，会自动跳转到主菜单界面。

- 失败界面类 FailView

该类为玩家没有打破记录进入的失败界面，同 WinView 一样，也会自动跳转到主菜单界面。

- 关于界面类 AboutView

该类为游戏中关于界面类，界面中有游戏版本、版权等字样。

- 选项界面类 ChoiceView

该类为游戏中选项界面，在选项界面中有游戏模式的选项及排行榜。

- 帮助界面类 HelpView

该类为游戏中的帮助界面，在界面中有对应的帮助信息。

3. 游戏界面相关类

- 游戏主界面类 GameView

该类为游戏的主界面类，主要负责游戏中各个对象的创建、游戏中线程控制以及事件的监听等工作。

- 主绘制线程类 GameViewDrawThread

该类为游戏界面的主绘制线程类，主要负责游戏中各个对象的不断绘制。

- 倒计时线程类 TimeRunningThread

该类为游戏中倒计时线程类，主要负责更改游戏界面中游戏的时间。

- 触控监听线程 KeyThread

该类为触控监听线程，每隔一定时间读取屏幕的触控状态值，并根据当前屏幕的触控状态值做出对应的操作。

- BallGoThread

该类为控制桌球滚动的类。

- CueAnimateThread

该类为游戏中负责绘制球杆击球时产生击球动画的类。

4. 情景相关类

- 桌球 Ball

该类为游戏中的桌球类，游戏中所有的球都是该类的一个对象。

- 球杆 Cue

该类为游戏中的球杆类，用于玩家瞄准并打击母球。

- 球台 Table

该类为游戏中的球台类，球台是将拆散的各个球台碎片重新组装起来形成的。

5. 自定义控件及工具类

- 力度条 StrengthBar

该类为控制炮弹发射速度的力度条封装类，滑动力度条可以很方便地调节击球力度。

- 计时器 Timer

该类为游戏中显示游戏倒计时的封装类，负责将游戏进行时间用相应的图片显示在游戏界面中。

- 声音开关按钮 SoundSwitchButton

该类为游戏音效控制界面中的声音开关按钮封装类，负责根据开关标志来绘制相应开关的图片。

- 虚拟按钮 VirtualButton

该类为游戏中所有虚拟按钮类，游戏中的所有单个的虚拟按钮都是该类的一个对象。

- 获取系统时间的工具类 DateUtil

该类为封装好的得到系统时间的工具类。调用其相应的静态工具方法即可得到系统时间。

- 自适应屏 ScreenScaleResult、ScreenScaleUtil

ScreenScaleResult 类和 SrceenScaleUtil 类负责自动判断横竖屏，之后便可加载横竖屏各自的常量。

- CollisionUtil

该类为游戏中负责处理桌球之间的碰撞检测的类。

- PicLoadUtil

该类为游戏中负责更改图片的大小以自动适应屏的类。

- ColorUtil

该类为游戏中用到的力度条中色彩值的类。

17.4 公共类的实现

从本节开始将正式介绍游戏代码的开发。为了使代码的结构更加清晰，也为了使读者了解游戏的整个开发过程，本小节将尽量按原开发顺序对各个类一一进行讲解。

17.4.1 主控制类——GameActivity 的代码框架

首先介绍的是 Activity 的实现。Activity 的作用是对各个界面进行管理、切换以及对线程发送来的请求做出响应，其代码框架如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1    package com.bn.d2.bill;                                    //声明包
2    import android.app.Activity;                               //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.widget.Toast;                               //引入相关类
5    class WhatMessage{                                         //标识所有 SurfaceView 的常量类
6         public static final int GOTO_WELLCOME_VIEW=0;         //欢迎界面
7         public static final int GOTO_MAIN_MENU_VIEW=1;        //主菜单界面
8         public static final int GOTO_GAME_VIEW=2;             //游戏界面
9         public static final int GOTO_SOUND_CONTORL_VIEW=3;    //声音控制界面
10        public static final int GOTO_WIN_VIEW=4;              //胜利界面
11        public static final int GOTO_FAIL_VIEW=5;             //失败界面
12        public static final int GOTO_HIGH_SCORE_VIEW=6;       //排行榜界面
13        public static final int GOTO_HELP_VIEW=7;             //帮助界面
14        public static final int GOTO_ABOUT_VIEW=8;            //关于界面
15        public static final int GOTO_CHOICE_VIEW=9;           //选项界面
16        public static final int OVER_GAME=20;                 //游戏结束
17   }
18   public class GameActivity extends Activity {
19        int currentView;                                      //标识当前的界面
20        WellcomeView wellcomeView;                            //显示欢迎动画界面
21        MainMenuView mainMenuView;                            //主菜单界面
22        HelpView helpView;                                    //帮助界面
23        AboutView aboutView;                                  //关于界面
24      GameView gameView;                                      //游戏界面
25      WinView winView;                                        //胜利界面
26        FailView failView;                                    //失败界面
27        SoundControlView soundControlView;                    //音效控制界面
28        HighScoreView highScoreView;                          //排行榜界面
29        ChoiceView choiceView;
30        boolean coundDownModeFlag=true;                       //是否为倒计时模式的标志
31        private boolean backGroundMusicOn=false;              //背景音乐是否开启的标志
32        private boolean soundOn=true;                         //音效是否开启的标志
33        static int initTime=0;                                //初始化的次数
34        int currScore;                                        //游戏结束后的得分
35        int highestScore;
36        SQLiteDatabase sld;                                   //SQLiteDatabase 数据库
37      Handler myHandler = new Handler(){
```

```
38          ……//此处省略了对此成员变量的初始化，将在后面章节中给出};
39      public void onCreate(Bundle savedInstanceState){
40          super.onCreate(savedInstanceState);
41              requestWindowFeature(Window.FEATURE_NO_TITLE); //全屏幕
42              getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN ,
43                          WindowManager.LayoutParams.FLAG_FULLSCREEN);
44          this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                                                                  //设置为横屏
45          setVolumeControlStream(AudioManager.STREAM_MUSIC);    //只允许调整多媒体音量
46          DisplayMetrics dm=new DisplayMetrics();               //获取分辨率
47          getWindowManager().getDefaultDisplay().getMetrics(dm);
48          if(initTime==0){
49              Constant.initConst(dm.widthPixels, dm.heightPixels);    //初始化常量
50              initTime++;
51          }
52          gotoWellcomeView();                                   //切换到欢迎界面
53      }
54      public boolean onKeyDown(int keyCode, KeyEvent e){    //重写 onKeyDown 方法
55             ……//此处省略了此方法的具体实现，将在后面章节中给出}
56
57      public void sendMessage(int what){                    //向 Handler 发送信息的方法
58        Message msg1 = myHandler.obtainMessage(what);       //获得信息对象
59        myHandler.sendMessage(msg1);                        //发送信息
60      }
61      public boolean isBackGroundMusicOn() {                //判断背景音乐是否开启的方法
62             return backGroundMusicOn;
63        }
64        public void setBackGroundMusicOn(boolean backGroundMusicOn) {
                                                              //开启背景音乐的方法
65             this.backGroundMusicOn = backGroundMusicOn;
66        }
67        public boolean isSoundOn() {                        //判断音效是否开启的方法
68             return soundOn;
69        }
70        public void setSoundOn(boolean soundOn) {           //关闭音效的方法
71             this.soundOn = soundOn;
72        }
73        //以下方法都省略了方法体，其具体实现将在后面章节中给出
74      private void gotoWellcomeView(){}                     //切换到欢迎界面的方法
75      private void gotoMainMenuView(){}                     //切换到主菜单界面的方法
76      private void gotoGameView(){}                         //切换到游戏界面
77      private void gotoHelpView(){}                         //切换到帮助界面的方法
78      private void gotoAboutView(){}                        //切换到关于界面的方法
79      private void gotoSoundControlView(){}                 //切换到设置声音界面的方法
80      private void gotoWinView(){}                          //切换到胜利界面的方法
81      private void gotoFailView(){}                         //切换到失败界面的方法
82      private void gotoChoiceView(){}                       //切换到选项界面的方法
83      private void gotoHighScoreView(){}                    //切换到排行榜界面的方法
84      public void openOrCreateDatabase(){}                  //打开或创建数据库的方法
85      public void closeDatabase(){}                         //关闭数据库的方法
86      public void insert(int score,String date){}           //插入记录的方法
87      public String query(int posFrom,int length){}         //查询的方法
88      public int getRowCount(){}                            //得到数据库中记录条数的方法
89      private void goToOverView(){}     //将得分和时间插入数据库，并跳转到相应的结束界面
90  }
```

- 代码第 5-16 行定义了标识所有界面的常量类，以常量代表数字的目的是增加代码的可维护性。
- 代码第 18-38 行为 GameActiviy 类中成员变量的声明，这些成员大部分是整个游戏开发过程中都会用到的全局变量。

- 代码第 61-72 行实现了对私有成员变量 backGroundMusicOn 和 soundOn 的封装。

说明 以上代码只是 GameActiviy 类的代码框架，其中大部分方法和成员变量都是空实现，不过读者不用担心，这些方法在下一小节将一一实现。

17.4.2 GameActivity 类主要成员变量及方法的实现

在前面小节中介绍了 GameActivity 类的代码框架，本小节将会对其中主要成员变量及方法的开发进行详细介绍，其步骤如下。

（1）成员变量 myHandler 的实现。myHandler 主要负责处理其他类发送来的信息，其代码如下。将下列代码插入到 GameActivity 类的第 37 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1     Handler myHandler = new Handler(){                          //处理各个 View 发送的消息
2           public void handleMessage(Message msg) {              //处理消息的方法
3               switch(msg.what)                                  //消息内容
4               {
5               case WhatMessage.GOTO_MAIN_MENU_VIEW:
6                    gotoMainMenuView();                          //转到主菜单界面
7               break;
8               case WhatMessage.GOTO_GAME_VIEW:
9                    gotoGameView();                              //转到游戏界面
10              break;
11              case WhatMessage.GOTO_SOUND_CONTORL_VIEW:
12                   gotoSoundControlView();                      //转到音效设置界面
13              break;
14              case WhatMessage.GOTO_WIN_VIEW:
15                   gotoWinView();                               //转到胜利界面
16              break;
17              case WhatMessage.GOTO_FAIL_VIEW:
18                   gotoFailView();                              //转到失败界面
19              break;
20              case WhatMessage.GOTO_HIGH_SCORE_VIEW:
21                   gotoHighScoreView();                         //转到排行榜
22              break;
23              case WhatMessage.GOTO_WELLCOME_VIEW:
24                   gotoWellcomeView();                          //转到欢迎界面
25              break;
26              case WhatMessage.GOTO_HELP_VIEW:
27                   gotoHelpView();                              //转到帮助界面
28              break;
29              case WhatMessage.GOTO_ABOUT_VIEW:
30                   gotoAboutView();                             //转到关于界面
31              break;
32              case WhatMessage.GOTO_CHOICE_VIEW:
33              gotoChoiceView();                                 //转到选项界面
34              break;
35              case WhatMessage.OVER_GAME:
36                   goToOverView();                              //游戏结束选择方法
37              break;
38        }}};
```

- myHandler 是继承了 Handler 类的匿名内部类的对象，主要负责发送信息，当收到信息后，根据不同的信息调用其他成员方法。

- msg.what 是整型的，因此，WhatMessage 类中的常量是整型的，而不应是枚举类型。

（2）onKeyDown 方法的实现。onKeyDown 方法主要负责监听在游戏中按下返回键转到相应界面。其代码如下。将下列代码插入到 GameActivity 类的第 54 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1    public boolean onKeyDown(int keyCode, KeyEvent e)
2        {
3          switch(keyCode)
4          {
5              case 4://返回键
6                     switch(currentView){//当前界面
7                          case WhatMessage.GOTO_WELLCOME_VIEW:     //欢迎界面
8                          break;
9                          case WhatMessage.GOTO_MAIN_MENU_VIEW:    //主菜单界面
10                              System.exit(0);                     //退出游戏
11                         break;
12                         case WhatMessage.GOTO_HIGH_SCORE_VIEW:   //排行榜界面
13                              gotoChoiceView();
14                         break;
15                         case WhatMessage.GOTO_GAME_VIEW:         //游戏界面
16                         case WhatMessage.GOTO_SOUND_CONTORL_VIEW://音效控制界面
17                         case WhatMessage.GOTO_WIN_VIEW:          //胜利界面
18                         case WhatMessage.GOTO_FAIL_VIEW:         //失败界面
19                         case WhatMessage.GOTO_HELP_VIEW:         //帮助界面
20                         case WhatMessage.GOTO_ABOUT_VIEW:        //关于界面
21                         case WhatMessage.GOTO_CHOICE_VIEW:       //选项界面
22                              gotoMainMenuView();                 //切换到主菜单界面
23                         }
24                 return true;
25         }
26             return false;
27     }
```

- 每个 View 中也有 onKeyDown 方法，但在跳转界面时只能用 Activity 中的此方法。
- 成员变量 currentView 的作用是标记当前界面是哪个 SurfaveView，当在相应界面中按下返回键时的此方法会根据 currentView 的值判断具体转到哪一个界面。

（3）gotoMainMenuView 、gotoWellcomeView 等方法的实现，gotoMainMenuView 方法的代码如下。将下列代码插入到 GameActivity 类的第 75 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1    private void gotoMainMenuView(){                          //切换到主菜单界面的方法
2          if(mainMenuView==null){
3              mainMenuView = new MainMenuView(this);         //创建主菜单界面对象
4          }
5          this.setContentView(mainMenuView);                 //将界面设置为主菜单界面
6          currentView=WhatMessage.GOTO_MAIN_MENU_VIEW;       //标记主菜单界面为当前界面
7      }
```

提示　这些方法都是私有的，在其他类中想要实现跳转功能必须通过 Handler 发送请求消息来实现。其他方法和此方法非常类似，由于本书篇幅有限，这里不再赘述，读者可自行查阅随书光盘中源代码。

（4）openOrCreateDatabase 方法的实现，其代码如下。将下列代码插入到 GameActivity 类的第 84 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1    public void openOrCreateDatabase(){                          //打开或创建数据库的方法
2         try{
3              sld=SQLiteDatabase.openDatabase(                   //打开数据库
4                        "/data/data/bn.qiulei/mydb",             //数据库所在路径
5                        null,                                    //CursorFactory
6                        SQLiteDatabase.OPEN_READWRITE|SQLiteDatabase.CREATE_IF_
                         NECESSARY
7              );
8              String sql="create table if not exists highScore" +//创建表的 SQL 语句
9                        "( " +
10                       "score integer," +
11                       "date varchar(20)" +
12                       ");";
13             sld.execSQL(sql);                                          //执行 SQL 语句
14        }catch(Exception e){
15             Toast.makeText(this, "数据库错误: "+e.toString(), Toast.LENGTH_SHORT).
               show();                                                    //显示异常
16        }}
```

- 代码第 3-7 行为创建 SQLite 数据库方法，第三个参数的含意为读写，若不存在则创建数据库。
- 代码第 8-13 行为开发创建表的 SQL 语句并执行的代码，表中只有分数和时间两个列。

（5）insert 方法的实现。该方法是对数据库中创建的 highScore 表中插入一条记录的方法，其代码如下。将下列代码插入到 GameActivity 类的第 86 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1    public void insert(int score,String date){                  //插入记录的方法
2         try{
3              String sql="insert into highScore values("+score+",'"+date+"');";
                                                                  //插入一条记录语句
4              sld.execSQL(sql);                                  //执行 SQL 语句
5              sld.close();                                       //关闭数据库
6         }catch(Exception e){
7                   Toast.makeText(this, "数据库错误: "+e.toString(), Toast.LENGTH_SHORT).
                    show();;
8              }
9        }
```

提示　该方法和打开数据库的方法类似，也是执行一条 SQL 语句即可实现插入一条数据。由于关闭数据库的方法比此方法还要简单，这里不再赘述，读者可自行查阅随书光盘中源代码。

（6）query 方法的开发。该方法主要负责将数据库中所有的记录以特定格式的字符串的形式返回。其代码如下。将下列代码插入到 GameActivity 类的第 70 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1    public String query(int posFrom,int length){          //查询的方法
2         StringBuilder sb=new StringBuilder();            //要返回的结果
```

```
3          Cursor cur=null;                                    //声明游标引用
4          openOrCreateDatabase();                             //打开数据库
5           String sql="select score,date from highScore order by score desc;";
                                                               //查询 SQL 语句
6           cur=sld.rawQuery(sql, null);                       //执行 SQL 语句
7          try{
8               cur.moveToPosition(posFrom);                   //将游标移动到指定的开始位置
9               int count=0;                                   //当前查询记录条数
10              while(cur.moveToNext()&&count<length){         //循环查询结果临时表
11                   int score=cur.getInt(0);                  //获得分数
12                   String date=cur.getString(1);             //获得日期
13                   sb.append(score);                         //追加分数
14                   sb.append("/");                           //将日期和时间用"/"分隔开
15                   sb.append(date);                          //追加日期
16                   sb.append("/");                           //将记录用"/"分隔开
17                   count++;                                  //当前查询记录数加 1
18              }
19         }catch(Exception e){
20                   Toast.makeText(this, "数据库错误: "+e.toString(), Toast.LENGTH_SHORT).
                     show();
21              }finally{
22                   cur.close();                              //关闭游标
23                   closeDatabase();                          //关闭数据库
24              }
25              return sb.toString();                          //转换成字符，并返回
26       }
```

- 代码第 4-5 行为打开数据库并执行查询语句的代码，注意查询结果是按得分降序排列的，这样才能保证排行榜中最前面的记录数据是最高分。
- 代码第 8-18 行将临时表中必要的数据信息取出并加入到 StringBuilder 类的对象 sb 中。其中参数 posFrom 和 length 分别代表要开始查询的位置及要查询记录的条数。
- 代码第 21-24 行为关闭已经打开的资源。注意，已经打开的资源一定要关闭，所以，这两行代码放到了 finally 语句中。

（7）getRowCount 方法的开发。该方法的作用是得到数据库中记录的总行数。其代码如下。将下列代码插入到 GameActivity 类的第 88 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1    public int getRowCount(){                                 //得到数据库中记录条数的方法
2         int result=0;                                        //声明返回结果
3         Cursor cur=null;//游标引用
4         openOrCreateDatabase();                              //打开数据库
5         try{
6              String sql="select count(score) from highScore;";   //查询 SQL 语句
7                 cur=sld.rawQuery(sql, null);                     //执行 SQL 语句
8              if(cur.moveToNext())                                //游标移动到下一行
9                   result=cur.getInt(0);                          //得到记录总条数
10             }
11        }
12        catch(Exception e){
13                  Toast.makeText(this, "数据库错误:"+e.toString(), Toast.LENGTH_SHORT).
                    show();
14             }finally{
15                  cur.close();                                   //关闭游标
16                  closeDatabase();                               //关闭数据库
17             }
18        return result;
```

```
19        }
```

- 对于数据库中数据的查询应尽量用一句 SQL 语句完成。如果先把数据取出来，用 Java 代码实现选最大值，执行速度会慢很多。
- 代码第 8 行中用到 cur.moveToNext()方法的原因是，.rawQuery 方法返回的 Cursor 类的对象指向的是结果集中第一条记录的前一行。

（8）goToOverView 方法的开发。该方法的作用是，当游戏结束时将玩家游戏记录插入到数据库中，并根据玩家的得分跳转到相应结束界面，其代码如下。将下列代码插入到 GameActivity 类的第 89 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameActivity.java。

```
1    private void goToOverView(){          //将得分和时间插入数据库，并跳转到相应的结束界面
2         Cursor cur=null;                 //游标引用
3         openOrCreateDatabase();          //打开或创建数据库
4         try{                             //从数据库中选出最高分
5              String sql="select max(score) from highScore;";//SQL 语句
6              cur=sld.rawQuery(sql, null);                  //执行 SQL 语句
7              if(cur.moveToNext()){                         //如果结果集不为空，移到下一行
8                   this.highestScore=cur.getInt(0);         //获得最高分
9              }
10             insert(currScore,DateUtil.getCurrentDate());//获得当前分数和日期并插入数据库
11        }catch(Exception e){
12                  Toast.makeText(this, "数据库错误:"+e.toString(), Toast.LENGTH_SHORT).
                    show();
13             }finally{
14                  cur.close();                        //关闭游标
15                  closeDatabase();                    //关闭数据库
16             }
17             if(currScore>highestScore){             //如果当前得分大于排行榜中最高分
18             this.gotoWinView();                     //进入胜利的界面
19             }
20             else{                                   //如果当前得分不大于排行榜中最高分
21                  this.gotoFailView();               //进入失败的界面
22        }}
```

提示　方法体中用到了 DateUtil 类中静态方法 getCurrentDate 来获取当前日期，读者不必担心不理解，在这里只要知道其返回值为满足一定格式的日期字符串即可，DateUtil 类会在后面的章节中做详细介绍。

17.4.3　常量类——Constant 的实现

Constant 类的作用是将代码中用到的常量全部集中起来，封装成一个类。这样便于游戏开发时的调试和管理，这里也将碰撞检测的方法封装进了此常量类中，其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 Constant.java。

```
1    package com.bn.d2.bill;                                  //声明包
2    public class Constant {
3         public static ScreenScaleResult ssr;                //屏幕的结果
4         public static float wRatio;                         //适应全屏的缩放比例
5         public static float hRatio;
6         public static int SCREEN_WIDTH;                     //屏幕的宽度
7         public static int SCREEN_HEIGHT;                    //屏幕的高度
```

```
8          //关于球台的常量
9          public static float BOTTOM_WIDTH=85*2+168*2+95;      //整个球台面宽度
10         public static float BOTTOM_HEIGHT=85*2+228;          //整个球台面高度
11         //……此处省略了部分源代码，请读者自行查看随书光盘中的源代码
12         //关于小背景的常量
13         static float BMP_Y=150;                              //小背景位置
14         static float HELP_Y=150;                             //帮助图片的位置
15         //初始化常量的方法
16         public static void initConst(int screenWidth,int screenHeight){
17              SCREEN_WIDTH=screenWidth;                       //屏幕的尺寸
18              SCREEN_HEIGHT=screenHeight;
19              wRatio=screenWidth/(float)screenWidthTest;      //适应全屏的缩放比例
20              hRatio=screenHeight/(float)screenHeightTest;
21              ssr=ScreenScaleUtil.calScale(screenWidth, screenHeight); //计算屏幕的结果
22              X_OFFSET=ssr.lucX;
23              Y_OFFSET=ssr.lucY;
24              //关于球台的常量
25              BOTTOM_WIDTH*=ssr.ratio;                        //整个球台面宽度
26              BOTTOM_HEIGHT*=ssr.ratio;                       //整个球台面高度
27              //此处省略了部分源代码，请读者自行查看随书光盘中的源代码
28              BMP_Y*=ssr.ratio;
29              HELP_Y*=ssr.ratio;
30         }
31         public static boolean IsTwoRectCross                 //判断两个矩形是否交叉的方法
32         (
33                   float xLeftTop1,float yLeftTop1,float length1,float width1,
                                                                //左上点 x,y 坐标，长，宽
34                   float xLeftTop2,float yLeftTop2,float length2,float width2
35         )
36         {
37              if
38              (
39                        isPointInRect(xLeftTop1,yLeftTop1,xLeftTop2,yLeftTop2,
                          length2,width2)||
40                        isPointInRect(xLeftTop1+length1,yLeftTop1,xLeftTop2,yLeftTop2,
                          length2,width2)||
41                        isPointInRect(xLeftTop1,yLeftTop1+width1,xLeftTop2,yLeftTop2,
                          length2,width2)||
42                   isPointInRect(xLeftTop1+length1,yLeftTop1+width1,xLeftTop2,yLeftTop2,
                     length2,width2)||
43
44                        isPointInRect(xLeftTop2,yLeftTop2,xLeftTop1,yLeftTop1,
                          length1,width1)||
45                        isPointInRect(xLeftTop2+length2,yLeftTop2,xLeftTop1,yLeftTop1,
                          length1,width1)||
46                        isPointInRect(xLeftTop2,yLeftTop2+width2,xLeftTop1,yLeftTop1,
                          length1,width1)||
47                   isPointInRect(xLeftTop2+length2,yLeftTop2+width2,xLeftTop1,yLeftTop1,
                     length1,width1)
48              )
49              {
50                   return true;                               //两个矩形有交叉，返回 true
51              }
52              return false;                                   //两个矩形没有交叉，返回 false
53         }
54         public static boolean isPointInRect(                //一个点是否在矩形内（包括边界）
55                   float pointx,float pointy,                 //点的坐标
56                   float xLeftTop,float yLeftTop,float length,float width
                                                                //矩形的左上点 x,y 坐标，长，宽
57         )
58         {
```

```
59                if(
60                        pointx>=xLeftTop&&pointx<=xLeftTop+length&&
                                                        //判断点是否在矩形区域内
61                        pointy>=yLeftTop&&pointy<=yLeftTop+width
62                  )
63                  {
64                      return true;                    //在矩形区域内返回 true
65                  }
66                return false;                         //不在矩形区域内返回 false
67            }
68    }
```

- 代码第 3-14 行为游戏中用到常量的定义，适当调节这些变量的值可以很方便地调试游戏。
- 代码第 16-30 行为判断屏幕缩放比例后，按比例缩放常量值的方法。
- 代码第 31-53 行为判断两个矩形是否相交的方法，基本思想是，只要其中一个矩形的 4 个顶点之一在另一个矩形内，两个矩形就是相交的，反之不相交。

17.5 辅助界面相关类的实现

上一节介绍了公共类的实现，本节开始介绍辅助界面相关类的实现。辅助界面相关类指，除游戏界面之外其他辅助界面，包括欢迎动画界面、主菜单界面、音效设置界面、排行榜界面、胜利界面和失败界面等。下面先来介绍欢迎动画界面。

17.5.1 欢迎动画界面 WelcomeView 类的实现

本游戏欢迎动画界面的实现比较简单，主要是靠单独开启的线程来不断地调节画笔的不透明度，并不断地用该画笔绘制图片来实现。其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 WelcomView.java。

```
1     package com.bn.d2.bill;                                  //声名包
2     import android.graphics.Bitmap;                          //引入相关类
3     ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4     import android.view.View;                              //引入相关类
5     public class WellcomeView extends View
6     implements Holder.Callback  {                          //实现生命周期回调接口
7          GameActivity activity;
8          Paint paint;                                        //画笔
9          int currentAlpha=0;                                 //当前的不透明值
10         int screenWidth=Constant.SCREEN_WIDTH;              //屏幕宽度
11         int screenHeight=Constant.SCREEN_HEIGHT;            //屏幕高度
12         int sleepSpan=50;                                   //动画的时延 ms
13         Bitmap[] logos=new Bitmap[2];                       //Logo 图片数组
14         Bitmap currentLogo;                                 //当前 Logo 图片引用
15         int currentX;                                       //当前位置
16         int currentY;
17         public WellcomeView(GameActivity activity) {        //构造器
18              super(activity);
19              this.activity = activity;
20              this.getHolder().addCallback(this);            //设置生命周期回调接口的实现者
21              paint = new Paint();                           //创建画笔
22              paint.setAntiAlias(true);                      //打开抗锯齿
23              logos[0]=BitmapFactory.decodeResource(activity.getResources(),  R.drawable.
                dukea);
```

```
24              logos[1]=BitmapFactory.decodeResource(activity.getResources(), R.drawable.
                dukeb);
25          }
26          public void onDraw(Canvas canvas){                  //重写 onDraw 方法
27              paint.setColor(Color.BLACK);                    //设置画笔颜色
28              paint.setAlpha(255);                            //设置画笔透明度
29              canvas.drawRect(0, 0, screenWidth, screenHeight, paint); //绘制黑填充矩形，
                                                                          //清除背景
30              if(currentLogo==null)return;
31              paint.setAlpha(currentAlpha);                   //设置画笔透明度
32              canvas.drawBitmap(currentLogo, currentX, currentY, paint); //进行平面贴图
33          }
34          public void SurfaceChanged(Holder arg0, int arg1, int arg2, int arg3) {}
35          public void SurfaceCreated(Holder holder) {         //创建时被调用
36              new Thread(){                                   //创建线程对象
37                  public void run(){                          //实现 run 方法
38                      for(Bitmap bm:logos){                   //循环两张欢迎图片
39                          currentLogo=bm;                     //获得当前显示图片
40                          currentX=screenWidth/2-bm.getWidth()/2  //计算图片位置
41                          currentY=screenHeight/2-bm.getHeight()/2;
42                          for(int i=255;i>-10;i=i-10)     {
                                                    //动态更改图片的透明度值并不断重绘
43                              currentAlpha=i;   //获得当前透明度
44                              if(currentAlpha<0){   //保证透明度大于 0
45                                  currentAlpha=0;
46                              }
47                              Holder myholder=WellcomeView.this.getHolder();
48                              Canvas canvas = myholder.lockCanvas(); //获取画布
49                              try{
50                                  synchronized(myholder){   //锁定 myholder
51                                      onDraw(canvas);       //绘制
52                                  }
53                              }
54                              catch(Exception e){
55                                  e.printStackTrace(); //打印异常
56                              }
57                              finally{
58                                  if(canvas != null){  //如果画布引用不为空
59                                      myholder.unlockCanvasAndPost(canvas);
                                                            //释放画布
60                                  }
61                              }
62                              try{
63                                  if(i==255){           //若是新图片，多等待一会
64                                      Thread.sleep(1000);
65                                  }
66                                  Thread.sleep(sleepSpan); //使线程睡眠一段时间
67                              }
68                              catch(Exception e){
69                                  e.printStackTrace();      //打印异常
70                              }
71                          }
72                      }
73                      activity.sendMessage(WhatMessage.GOTO_MAIN_MENU_VIEW);
                                                                  //切换到主菜单界面
74                  }
75              }.start();                                        //启动线程
76          }
77          public void Destroyed(Holder arg0) {}                 //销毁时被调用
78      }
```

- 代码第 7-16 行为成员变量的定义，其中 paint 和 currentLogo 要定义为成员变量的原因是，使用和改变两个变量的值不在同一个方法内。
- 代码第 26-33 行为 onDraw 方法的实现。onDraw 方法主要负责用成员变量 paint 来绘制图片。
- 代码第 36-75 行创建并启动了一个线程，负责动态改变成员变量 paint 和 currentLogo 的值，其中第 73 行，当动画播放完后向 Activity 发送了一个切换到主菜单界面的消息。

17.5.2 主菜单界面 MainMenuView 的代码框架

上一小节中讲到动画播放完毕后，Activity 会将界面转到主菜单界面，本小节就将介绍主菜单界面的开发与实现，其代码框架如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 MainMenuView.java。

```
1    package com.bn.d2.bill;                                  //声明包
2    import android.content.res.Resources;                    //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import static com.bn.d2.bill.Constant.*;                 //引入相关类
5    public class MainMenuView extends SurfaceView implements SurfaceHolder.Callback{
6         GameActivity activity;                              //Activity 的引用
7         Paint paint;                                        //画笔
8         Bitmap[] menu;                                      //菜单图片数组
9         Bitmap bj;                                          //背景图片
10        int currentIndex=2;                                 //当前选中的菜单编号
11        float mPreviousX;                                   //上次触控的 X 坐标
12        float mPreviousY;                                   //上次触控的 Y 坐标
13        float changePercent=0;                              //动画进行的百分比
14        int anmiState=0;                                    //0-没有动画，1-向右走，2-向左走
15        float currentWidth;                                 //当前菜单项宽度
16        float currentHeight;                                //当前菜单项高度
17        float currentX;                                     //当前菜单项 X 位置
18        float currentY;                                     //当前菜单项 Y 位置
19        float leftWidth;                                    //紧邻当前菜单项左侧菜单项的宽度
20        float leftHeight;                                   //紧邻当前菜单项左侧菜单项的高度
21        float tempxLeft;                                    //紧邻当前菜单项左侧菜单项的 X 坐标
22        float tempyLeft;                                    //紧邻当前菜单项左侧菜单项的 Y 坐标
23        float rightWidth;                                   //紧邻当前菜单项右侧菜单项的宽度
24        float rightHeight;                                  //紧邻当前菜单项右侧菜单项的高度
25        float tempxRight;                                   //紧邻当前菜单项右侧菜单项的 X 坐标
26        float tempyRight;                                   //紧邻当前菜单项右侧菜单项的 Y 坐标
27        static float initial_Width;                         //菜单项图标初始宽度高度
28        static float initial_Height;
29        final int ABOUT_VIEW=0;                             //关于
30        final int HELP_VIEW=1;                              //帮助
31        final int START_VIEW=2;                             //开始游戏
32        final int SETUP_VIEW=3;                             //设置
33        final int EXIT_VIEW=4;                              //退出
34        public MainMenuView(GameActivity activity){//构造器
35             super(activity);                               //调用父类构造器
36             this.activity=activity;
37             this.getHolder().addCallback(this);            //添加回调接口的监听
38             paint=new Paint();                             //创建画笔
39             paint.setAntiAlias(true);                      //抗锯齿
40             initBitmap(activity.getResources());
41             init();                              //初始化当前及紧靠其左右的菜单项的位置大小参数
42        }
43        public boolean onTouchEvent(MotionEvent e) {}          //重写 onTouchEvent 方法
44        public void onDraw(Canvas canvas){}                    //onDraw 方法
```

```
45          public void surfaceSurfaceChanged(SurfaceHolder holder, int format, int width,
            int height){}
46          public void surfaceSurfaceCreated(SurfaceHolder holder){//重写 surfaceCreated
                                                                    //方法
47                repaint();
48          }
49          public void surfaceDestroyed(SurfaceHolder holder){}//重写 surfaceDestroyed
                                                                //方法
50          public void repaint(){}                             //重新绘制的方法
51          public void initBitmap(Resources r){}  //初始化图片资源的方法
52          public void init(){}
53          public void drawBitmap                 //在指定的位置按照指定的缩放比绘制图片
54          (
55                Canvas c,                        //画布
56                float x,float y,                 //绘制位置 xy 坐标
57                float xRatio,float yRatio,       //XY 方向的缩放比
58                Bitmap bm                        //待绘制的图片
59          )
60          {
61                Matrix m1=new Matrix();          //设置缩放矩阵
62                m1.setScale(xRatio, yRatio);
63                Matrix m2=new Matrix();          //设置平移矩阵
64                m2.setTranslate(x, y);
65                Matrix mz=new Matrix();          //计算总矩阵
66                mz.setConcat(m2, m1);
67                c.drawBitmap(bm, mz, paint);     //绘制当前的菜单项
68    }}
```

- 代码第 6-33 行为成员变量的定义，基中主要包括各个图片及图片的位置。
- 代码第 43-52 行大部分方法均为空实现，这些空实现的方法将在下一小节中介绍。
- 代码第 53-68 行为在指定的位置按照指定的缩放比绘制图片的方法，该方法是通过矩阵来实现的，创建一个缩放矩阵，再创建一个平衡矩阵，再以两个矩阵的乘积绘制图片。

说明　音效控制等界面实现非常简单，由于本书篇幅所限，这里不再赘述，请读者自行查看随书光盘中的源代码。

17.5.3　主界面类——MainMenuView 部分成员方法的实现

上一小节中介绍了 MainMenuView 类的代码框架，本小节将介绍其部分成员方法的具体实现。

（1）onDraw 方法的实现。该方法主要负责根据当前成员变量的值绘制相应的图片，其代码如下。将下列代码插入到 MainMenuView 类的第 44 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 MainMenuView.java。

```
1     public void onDraw(Canvas canvas){
2               canvas.drawColor(Color.GRAY);
3               canvas.drawBitmap(bj, 0, 0, paint);                  //绘制背景
4               float ratioX=currentWidth/initial_Width;             //计算缩放比
5               float ratioY=currentHeight/initial_Height;
6               drawBitmap(canvas,currentX+Constant.X_OFFSET,currentY+Constant.Y_
                OFFSET,
7                         ratioX,ratioY,menu[currentIndex]);         //绘制当前菜单项
8               if(currentIndex>0){  //若当前菜单项不是第一项，则绘制紧邻当前菜单项左侧的菜单项

9                    ratioX=leftWidth/initial_Width;                 //计算缩放比
```

```
10                  ratioY=leftHeight/initial_Height;
11                  drawBitmap(canvas,tempxLeft+Constant.X_OFFSET, tempyLeft+Constant.
                    Y_OFFSET,
12                          ratioX,ratioY,menu[currentIndex-1]);     //绘制当前菜单项
13              }
14              if(currentIndex<menu.length-1){//若当前菜单项不是最后一项，则绘制紧邻当前菜单
                                               //项右侧的菜单项
15                  ratioX=rightWidth/initial_Width;                 //计算缩放比
16                  ratioY=rightHeight/initial_Height;
17                  drawBitmap(canvas,tempxRight+Constant.X_OFFSET,tempyRight+
                    Constant.Y_OFFSET,
18                          ratioX,ratioY,menu[currentIndex+1]);     //绘制当前菜单项
19              }
20                                                                   //向左绘制其他
                                                                     //未选中的菜单
21              for(int i=currentIndex-2;i>=0;i--){
22                  float tempx=tempxLeft-(span+smallWidth)*(currentIndex-1-i);
                                                                     //计算 X 值
23                  if(tempx<-smallWidth){                           //若不再屏幕上，
                                                                     //则不用绘制
24                      break;
25                  }
26                  float tempy=selectY+(bigHeight-smallHeight);     //计算 Y 值
27                  ratioX=smallWidth/initial_Width;                 //计算缩放比
28                  ratioY=smallHeight/initial_Height;
29                  drawBitmap(canvas,tempx+Constant.X_OFFSET,tempy+Constant.Y_
                    OFFSET,
30                          ratioX,ratioY,menu[i]);          //绘制当前菜单项
31              }
32              for(int i=currentIndex+2;i<menu.length;i++){   //向右绘制其他未选中的菜单
33                  float tempx=tempxRight+rightWidth+span+(span+smallWidth)*(i-
                    (currentIndex+1)-1);
34                    if(tempx>screenWidthTest){               //若不再屏幕上，则不用绘制
35                          break;
36                  }
37                  float tempy=selectY+(bigHeight-smallHeight);     //计算 Y 值
38                  ratioX=smallWidth/initial_Width;                 //计算缩放比
39                  ratioY=smallHeight/initial_Height;
40                  drawBitmap(canvas,tempx+Constant.X_OFFSET,tempy+Constant.Y_OFFSET,
41                              ratioX,ratioY,menu[i]);              //绘制当前菜单项
42          }}
```

- 代码第 6-19 行为绘制当前菜单项和与其紧邻的两边的菜单项，由于其大小位置很特殊，所以，不能和其他菜单统一绘制。
- 代码第 21-42 行为绘制其他菜单项的代码，这些菜单的大小相同，距离有规律，所以，用两个 for 循环即可实现所有菜单的绘制。

（2）onTouchEvent 方法的实现。该方法主要负责接受屏幕的触控事件，其代码如下。将下列代码插入到 MainMenuView 类的第 43 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 MainMenuView.java。

```
1       public boolean onTouchEvent(MotionEvent e) {
2         if(anmiState!=0){                                  //若动画播放中，则触控无效
3             return true;
4         }
5           float x = e.getX();                              //获取当前触控点的 XY 坐标
6           float y = e.getY();
7           switch (e.getAction()){                    //根据触控的不同动作执行不同的业务逻辑
8               case MotionEvent.ACTION_DOWN:
9                 mPreviousX=x;                        //记录触控笔 X 位置
10                mPreviousY=y;                        //记录触控笔 Y 位置
```

```
11              break;
12                case MotionEvent.ACTION_UP:
13                  float dx=x- mPreviousX;              //计算 X 位移
14                  if(dx<-slideSpan){                   //若 X 位移小于阈值则向左滑动
15                   if(currentIndex<menu.length-1){ //若当前菜单项不是最后一个菜单项则向左滑动
16                        int afterCurrentIndex=currentIndex+1;//计算滑动完成后的当前菜单项编号
17                        anmiState=2;                   //动画状态值设置为 2，向左走
18                        new ViewDrawThread(this,afterCurrentIndex).start();
                                                         //启动线程播放动画更新状态值
19                   }
20                  }
21                  else if(dx>slideSpan){               //若 X 位移大于阈值则向右滑动
22                   if(currentIndex>0){
23                        int afterCurrentIndex=currentIndex-1;//计算滑动完成后的当前菜单项编号
24                        anmiState=1;                   //动画状态值设置为 1，向右走
25                        new ViewDrawThread(this,afterCurrentIndex).start();
                                                         //启动线程播放动画更新状态值
26                   }
27                  }
28                          else{                        //若 X 位移在阈值内则判断有否选中某菜单项
29                               if(//若按下与抬起的触控点都在当前菜单的范围内,则执行按下某菜单项
                                    //的业务逻辑
30                            Constant.isPointInRect(mPreviousX, mPreviousY,
31                                   selectX+Constant.X_OFFSET, selectY+Constant.Y_
                                     OFFSET,
32                                   bigWidth, bigHeight)&&
33                            Constant.isPointInRect(x, y,
34                                   selectX+Constant.X_OFFSET, selectY+Constant.Y_
                                     OFFSET,
35                                   bigWidth, bigHeight)
36                               )
37                               {
38                                   switch(currentIndex)            //当前菜单索引
39                                   {
40                                        case ABOUT_VIEW:           //关于
41                                             activity.sendMessage(WhatMessage.GOTO_
                                               ABOUT_VIEW);
42                                        break;
43                                        case HELP_VIEW:
44                                             activity.sendMessage(WhatMessage.GOTO_
                                               HELP_VIEW);//帮助
45                                        break;
46                                        case START_VIEW:           //模式选择
47                                             activity.sendMessage(WhatMessage.GOTO_
                                               CHOICE_VIEW);
48                                        break;
49                                        case SETUP_VIEW:
50          activity.sendMessage(WhatMessage.GOTO_SOUND_CONTORL_VIEW);//设置
51                                        break;
52                                        case EXIT_VIEW:
53                                             System.exit(0);            //退出程序
54                                        break;
55                          }}}
56                    break;
57            }
58            return true;
59        }
```

● 代码第 8-10 行为当按下屏幕中的某位置时，记录按下的位置坐标，以便判断抬起时是否滑动。

- 代码第 12-26 行为当在屏幕中抬起时，根据按下的位置和抬起的位置差判断是否为滑动动作，如果是滑动，对应改变当前索引值 currentIndex。
- 代码第 28-56 行为当点击在当前可触控的图片上时，根据前索引值 currentIndex 向 Activity 中的 Handler 发送不同的消息，进入相应界面。

17.5.4 主界面动画线程类 ViewDrawThread 的实现

该类负责根据当前索引和动画后的索引来动态改变图片的位置和大小以形成动画，其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 MainMenuView.java。

```
1    class ViewDrawThread extends Thread{
2         MainMenuView mv;                                    //MainMenuView 引用
3         int afterCurrentIndex;                              //移动后的当前索引
4         static boolean flag;                                //循环标志位
5         public ViewDrawThread(MainMenuView mv,int afterCurrentIndex){
6              this.mv=mv;
7              this.afterCurrentIndex=afterCurrentIndex;
8         }
9         public void run(){
10             for(int i=0;i<=totalSteps;i++){                 //循环指定的次数完成动画
11                  mv.changePercent=percentStep*i;            //计算此步的百分比
12                  mv.init();                                 //初始化各个位置值
13                  //动画总的思想就是，根据进度百分比计算出当前菜单项的位置、大小
14                  //并根据当前菜单项的位置大小计算出其左右侧紧邻菜单项的位置、大小
15                  //非紧邻项的大小是固定的，只需要计算位置
16                  if(mv.anmiState==1){                       //向右的动画
17                       mv.currentX=mv.currentX+(bigWidth+span)*mv.changePercent;
18                       mv.currentY=mv.currentY+(bigHeight-smallHeight)*mv.change-
                         Percent;
19                       mv.currentWidth=(int)(smallWidth+(bigWidth-smallWidth)*
                         (1-mv.changePercent));
20                       mv.currentHeight=(int)(smallHeight+(bigHeight-smallHeight)*
                         (1-mv.changePercent));
21                       //由于动画向右，左侧菜单项会变大，因此，计算左侧菜单项大小
22                       mv.leftWidth=(int)(smallWidth+(bigWidth-smallWidth)*mv.
                         changePercent);
23                       mv.leftHeight=(int)(smallHeight+(bigHeight-smallHeight)*mv.
                         changePercent);
24                  }
25                  else if(mv.anmiState==2){          //向左的动画
26                       mv.currentX=mv.currentX-(smallWidth+span)*mv.changePercent;
27                       mv.currentY=mv.currentY+(bigHeight-smallHeight)*mv.change-
                         Percent;
28                       mv.currentWidth=(int)(smallWidth+(bigWidth-smallWidth)*(1-
                         mv.changePercent));
29                       mv.currentHeight=(int)(smallHeight+(bigHeight-smallHeight)*
                         (1-mv.changePercent));
30                       //由于动画向左，右侧菜单项会变大，因此，计算右侧菜单项大小
31                       mv.rightWidth=(int)(smallWidth+(bigWidth-smallWidth)*mv.
                         changePercent);
32                       mv.rightHeight=(int)(smallHeight+(bigHeight-smallHeight)*mv.
                         changePercent);
33                  }
34                  mv.tempxLeft=mv.currentX-(span+mv.leftWidth);
                                                            //计算出紧邻左侧的菜单项位置
35                  mv.tempyLeft=mv.currentY+(mv.currentHeight-mv.leftHeight);
36                  mv.tempxRight=mv.currentX+(span+mv.currentWidth);
```

```
                                                        //计算出紧邻右侧的菜单项位置
37                  mv.tempyRight=mv.currentY+(mv.currentHeight-mv.rightHeight);
38                  mv.repaint();                       //重绘画面
39                  try{
40                       Thread.sleep(timeSpan);        //休眠一定时间
41                  }
42                  catch(Exception e){
43                       e.printStackTrace();           //打印异常
44              }}
45              mv.anmiState=0;                         //动画完成后设置动画状态为 0，无动画
46              mv.currentIndex=afterCurrentIndex;      //动画完成后更新当前的菜单项编号
47              mv.init();                              //初始化各个位置值
48              mv.repaint();                           //重绘画面
49   }}
```

- 代码第 2-8 行为该类的成员变量及构造器，创建该线程时需要传入移动后的索引值 afterCurrentIndex，以确定播放的是哪一段动画。
- 代码第 9-49 行用一个 for 循环动态图片的大小及位置，主要思想是用几个关键帧代替整个的图片变化的动画。

17.5.5　排行榜界面 HighScoreView 的代码框架

排行榜界面主要负责从数据库中取出数据，得到一定格式的字符串，再将其用图片显示在界面中。HighScoreView 的代码框架如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 HighScoreView.java。

```
1    package com.bn.d2.bill;                                        //声明包
2    import android.graphics.Bitmap;                                //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.view.View;                                  //引入相关类
5    public class HighScoreView extends View implements Holder.Callback{
6         GameActivity activity;                                //Activity 的引用
7         Paint paint;                                          //画笔引用
8         DrawThread drawThread;                                //绘制线程引用
9         Bitmap bgBitmap;                                      //背景图片
10        Bitmap highScoreBitmap;                               //文字的图片
11        Bitmap defenBitmap;
12        Bitmap riqiBitmap;
13        Bitmap[] numberBitmaps;                               //数字图片
14        Bitmap gangBitmap;                                    //符号"-"对应的图片
15        int bmpx;                                             //文字位置
16        String queryResultStr;                                //查询数据库的结果
17        String[] splitResultStrs;                             //将查询结果切分后的数组
18        private int numberWidth;                              //数字图片的宽度
19        private int posFrom=-1;                               //查询的开始位置
20        private int length=6;                                 //查询的最大记录个数
21        int downY=0;                                          //按下和抬起的 y 坐标
22        int upY=0;
23        public HighScoreView(GameActivity activity) {
24        ……//此处省略了构造方法体中代码，读者可自行查看随书光盘的源代码
25        }
26        protected void onDraw(Canvas canvas) {
27        ……//此处省略了绘制信息的代码，读者可自行查看随书光盘的源代码
28        }
29                                                              //画日期图片的方法
30        public void drawDateBitmap(String numberStr,float endX,float endY,Canvas
          canvas,Paint paint){
```

```
31          ……//此处省略了绘制信息的代码，将在下一小节中给出
32          }
33          public boolean onTouchEvent(MotionEvent event) {     //重写 onTouchEvent 方法
34          ……//此处省略了处理触控信息的代码，将在下一小节中给出
35          }
36          public void SurfaceCreated(Holder holder){           //重写 Created 方法
37          ……//此处省略了部分成员变量初始化的代码，将在下一小节中给出
38          }
39      ……//以下方变量或方法与前面介绍过的界面类类似，这里只给出其框架，读者可自行查看随书光盘的源代码
40          public void SurfaceChanged(Holder holder, int format, int width,int height) {}
41          public void Destroyed(Holder holder) {}
42          public void initBitmap(){}                           //初始化位图的方法
43          void createAllThreads(){}                            //创建所有线程的方法
44          void startAllThreads(){}                             //开启所有线程的方法
45          void stopAllThreads(){}                              //停止所有线程的方法
46          private class DrawThread extends Thread{}            //绘制线程类
47      }
```

17.5.6 HighScoreView 类的部分方法的实现

上一小节中介绍了 HighScoreView 类的代码框架，本小节将介绍其部分成员方法的具体实现。这里按方法被调用的先后顺序进行介绍。

（1）SurfaceCreated 方法的实现。该方法主要负责成员变量的初始化工作，其代码如下。将下列代码插入到 HighScoreView 类的第 36 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 HighScoreView.java。

```
1     public void SurfaceCreated(Holder holder){
2             paint=new Paint();                                  //创建画笔
3             paint.setAntiAlias(true);                           //打开抗锯齿
4             createAllThreads();                                 //创建所有线程
5             initBitmap();                                       //初始化位图资源
6             numberWidth=numberBitmaps[0].getWidth()+3;          //得到数字图片的宽度
7             bmpx=(Constant.SCREEN_WIDTH-highScoreBitmap.getWidth())/2;
                                                                  //初始化图片的位置
8             posFrom=-1;                                         //查询的开始位置，置-1
9             queryResultStr=activity.query(posFrom,length);      //得到数据库中的数据
10            splitResultStrs=queryResultStr.split("/", 0);       //用"/"切分，且舍掉空串
11            startAllThreads();                                  //开启所有线程
12        }
```

- 代码第 8 行中虽然成员变量 posFrom 的初值是–1，但是，仍要在该方法中将其重新置为–1，否则排行榜中会记录上次翻页情况，这不符合游戏的要求。
- 代码第 9-10 行是从数据库中得到一定格式返回的字符串形式的数据，所以，要用"/"将时间字符串分开，放到成员变量 splitResultStrs 中，以供其他方法中使用。
- 代码第 11 行中线程的开启要放在最后，保证所有初始化工作全部结束后再开启线程工作。

（2）drawDateBitmap 方法的实现。该方法的作用是将成员变量 splitResultStrs 中日期信息用图片的形式显示在排行榜上，其代码如下。将下列代码插入到 HighScoreView 类的第 30 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 HighScoreView.java。

```
1     public void drawDateBitmap(String numberStr,float endX,float endY,Canvas canvas,
Paint paint){
2             for(int i=0;i<numberStr.length();i++){      //循环字日期符串
3                 char c=numberStr.charAt(i);             //从左往右得到每个字符
```

```
4                    if(c=='-'){                                  //如果得到的字符为日期分隔符
5                         canvas.drawBitmap(gangBitmap,endX-numberWidth*(numberStr.
                          length()-i),
                             endY, paint);                        //绘制图片
6                    }
7                    else{                                        //得到的字符为数字
8                         canvas.drawBitmap(                      //用对应的数字图片绘制每个字符
9                             numberBitmaps[c-'0'],               //图片数组
10                            endX-numberWidth*(numberStr.length()-i),  //图片 x 位置
11                            endY,                                     //图片 y 位置
12                            paint                                     //画笔
13                        );
14        }}}
```

● 代码第 1 行中方法的参数 numberStr 只能是规定以'-'分开的日期格式的字符串，该方法在 onDraw 方法里被调用。

● 代码第 8-13 行为用图片绘制数字的方法，该方法是通用的，在本游戏的游戏界面中绘制时间时也用到了此方法。

（3）onTouchEvent 方法的实现。该方法主要负责排行榜中数据的翻页，其代码如下。将下列代码插入到 HighScoreView 类的第 33 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 HighScoreView.java。

```
1    public boolean onTouchEvent(MotionEvent event) {
2              int y = (int) event.getY();                        //得到事件发生的 y 坐标
3         switch(event.getAction()){
4         case MotionEvent.ACTION_DOWN:                           //如果是按下动作
5              downY=y;                                           //获取动作的 y 坐标
6         break;
7         case MotionEvent.ACTION_UP:                             //如果是抬起动作
8              upY=y;                                             //再次获取动作的 y 坐标
9              if(Math.abs(downY-upY)<20){                        //在域值范围内，不翻页
10                  return true;
11             }
12             else if(downY<upY){                                //向下拖动
13                  if(this.posFrom-this.length>=-1){             //如果移动到最前页，不可再移动
14                       this.posFrom-=this.length;
15                  }
16             }
17             else{                                              //向上拖动
18
19                  if(this.posFrom+this.length<activity.getRowCount()-1){
                                                                   //如果移动到最后页，不可再移动
20                       this.posFrom+=this.length;
21                  }
22             }
23             queryResultStr=activity.query(posFrom,length);     //得到数据库中的数据
24                  splitResultStrs=queryResultStr.split("/", 0);//用"/"切分，且舍掉空串
25             break;
26        }
27             return true;
28        }
```

● 代码第 4-11 行判断按下和抬起的位置是否超过了阈值，如果没有超过阈值，判断为没有翻页。如果不设定阈值，用手机点击屏幕也可能会翻，这显然不合要求。

● 代码第 12-25 行为当判断为翻页后，再判断是向上翻还是向下翻，注意，向上翻到第 1 页

或向下翻到最后一页时都不可以再翻。

说明　由于胜利界面和失败界面非常简单，这里不再赘述，请读者自行查看随书光盘中的源代码。

17.6 游戏界面相关类的实现

上一节中介绍了辅助界面相关类的实现，本节将介绍游戏中最重要的游戏界面相关类的实现。游戏界面相关类是游戏的主体，也是实现起来最复杂的部分，下面对其进行详细介绍。首先介绍的是最后一个界面 GameView 类的设计与实现。

17.6.1 游戏界面 GameView 类的代码框架

GameView 类是游戏的核心类，其他类都是为该类服务而做的准备工作。由于 GameView 的逻辑相对复杂，本小节先介绍该类的代码框架，其代码框架如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameView.java。

```
1    package com.bn.d2.bill;                                    //声明包
2    import java.util.ArrayList;                                //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.view.SurfaceView;                           //引入相关类
5    public class GameView extends SurfaceView implements SurfaceHolder.Callback{
6         GameActivity activity;                                //Activity 的引用
7         Paint paint;                                          //画笔引用
8         Bitmap[] tableBmps;                                   //球台图片
9         ……//此处省略了部分图片引用声名的代码，读者可自行查看随书光盘的源代码
10        Bitmap breakMarkBitmap;
11        //对象引用
12        List<Ball> alBalls;                                   //所有球的列表
13        Table table;                                          //球台
14        Cue cue;                                              //球杆
15        StrengthBar strengthBar;                              //力度条
16        VirtualButton goBtn;                                  //GO 图标按钮
17        VirtualButton leftBtn;                                //左按钮
18        VirtualButton rightBtn;                               //右按钮
19        VirtualButton aimBtn;                                 //目标按钮
20        Timer timer;                                          //计时器
21        //线程引用
22        GameViewDrawThread drawThread;                        //绘制线程
23        BallGoThread ballGoThread;                            //球前进的线程
24        KeyThread keyThread;                                  //按键监听线程
25        TimeRunningThread timeRunningThread;                  //负责计时的线程
26        //状态值
27        int keyState=0;//键盘状态  1-left 2-right 4-null 8-null 16-change bar 32-button
          press time
28        float btnPressTime=0;                                 //按下按钮的时间
29        //声音相关变量
30        SoundPool soundPool;                                  //声音池
31        HashMap<Integer, Integer> soundPoolMap;
32        MediaPlayer mMediaPlayer;                             //音乐播放器引用
33        public static final int SHOOT_SOUND=0;                //声音常量
34        public static final int HIT_SOUND=1;
35        public static final int BALL_IN_SOUND=2;
```

```
36        public GameView(GameActivity activity) {
37             super(activity);
38             this.activity=activity;
39             this.requestFocus();                          //获得焦点并设置为可触控
40          this.setFocusableInTouchMode(true);
41             getHolder().addCallback(this);                //注册回调接口
42        }
43        protected void onDraw(Canvas canvas) {}            //重写 onDraw 方法
44        public boolean onTouchEvent(MotionEvent event) {}  //重写 onTouchEvent 方法
45        public void surfaceCreated(SurfaceHolder holder){} //重写surfaceCreated方法
46        public void surfaceChanged(SurfaceHolder holder, int format, int width,
47                  int height) {}                           //重写surfaceChanged方法
48        public void surfaceDestroyed(SurfaceHolder holder) {}
49        public void initBitmap(){}                         //将图片加载
50        public void changeBmpsRatio(float ratio){}         //按比例改变图片大小的方法
51        public void changeBmpsRatioFullScreen(float wRatio,float hRatio){}
52        void createAllThreads(){}                          //创建所有线程的方法
53        void startAllThreads(){}                           //开启所有线程的方法
54        void stopAllThreads(){}                            //停止所有线程的方法
55        void joinAllThreads(){}                            //加入所有线程的方法
56        public void overGame(){}                           //结束游戏的方法
57      public void repaint(){}                              //重新绘制的方法
58        public void initSounds(){}                         //初始化声音的方法
59        public void playSound(int sound, int loop) {}      //播放声音的方法
60    }
```

- 代码第 11-20 行声明了游戏界面中用到的自定义类的对象的引用。
- 代码第 22-25 行声明了游戏中用到的 4 个线程，分别用于绘制游戏界面、使桌球运动、监听按键事件和游戏倒计时。
- 代码第 32-34 行定义了游戏中用到的常量和状态值。其中 keyState 的第 1、2 低到高分别代表球杆向左转、球杆向右转其他位未定义，留待扩充游戏玩法。

说明　代码框架中的大部分方法均为空实现，部分方法的实现将在下一小节中详细介绍。

17.6.2　GameView 类部分成员方法的实现

上一小节中介绍了 GameView 类的代码框架，本小节将介绍其部分成员方法的具体实现。这里依然按方法被调用的先后顺序进行介绍。

（1）SurfaceCreated 方法的实现。该方法主要负责成员变量的初始化工作，其代码如下。将下列代码插入到 HighScoreView 类的第 45 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameView.java。

```
1    public void surfaceCreated(SurfaceHolder holder){
2          paint=new Paint();                          //创建画笔
3          paint.setAntiAlias(true);                   //打开抗锯齿
4          createAllThreads();                         //创建所有线程
5          initBitmap();                               //初始化位图资源
6          changeBmpsRatio(Constant.ssr.ratio);        //改变图片的大小，尽量全屏，但不改变宽高比
7          changeBmpsRatioFullScreen(Constant.wRatio,Constant.hRatio);
                                                       //可以变形的图片适应全屏
8          initSounds();                               //初始化声音资源
9          mMediaPlayer = MediaPlayer.create(activity, R.raw.backsound);//初始化背景音乐
```

```
10          mMediaPlayer.setLooping(true);          //背景音乐循环播放
11          table=new Table(tableBmps);             //球台
12          alBalls=new ArrayList<Ball>();          //创建球列表，并加入所有球
13          for(int i=0;i<Table.AllBallsPos.length;i++){
14               alBalls.add(new Ball(ballBmps[i],this,0,0,Table.AllBallsPos[i]));
15          }
16          cue=new Cue(cueBmp,alBalls.get(0));     //球杆
17          strengthBar=new StrengthBar(barDownBmp,barUpBmp);    //力度条
18          goBtn=new VirtualButton(goDownBmp,goUpBmp,
19                    Constant.GO_BTN_X,Constant.GO_BTN_Y);      //GO 图标按钮
20          leftBtn=new VirtualButton(leftDownBmp,leftUpBmp,
21                    Constant.LEFT_BTN_X,Constant.LEFT_BTN_Y); //左按钮
22          rightBtn=new VirtualButton(rightDownBmp,rightUpBmp,
23                    Constant.RIGHT_BTN_X,Constant.RIGHT_BTN_Y);//右按钮
24          aimBtn=new VirtualButton(aimDownBmp,aimUpBmp,
25                    Constant.AIM_BTN_X,Constant.AIM_BTN_Y);    //右按钮
26          timer=new Timer(this,breakMarkBitmap,numberBitmaps);//创建计时器对象
27          if(activity.isBackGroundMusicOn()){                  //开启背景音乐
28               mMediaPlayer.start();
29          }
30          startAllThreads();                                   //开启线程
31     }
```

● 代码第 2-26 行是各成员变量的初始化，其中自定义的对象创建时需要为各个类分配必要的图片资源，以及 this 引用。

● 代码第 27-28 行为开启背景音乐的代码，控制是否开启音乐的 boolean 型变量没有写在该类中而是在 Activity 中的原因是，在音效控制界面中要能设置该值。

（2）onDraw 方法的实现。该方法主要负责各个对象的绘制工作，其代码如下。将下列代码插入到 HighScoreView 类的第 43 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameView.java。

```
1     protected void onDraw(Canvas canvas) {
2          super.onDraw(canvas);
3          canvas.drawColor(Color.BLACK);                          //整个屏幕背景色
4          canvas.drawBitmap(bgBmp, 0, 0, paint);                  //游戏界面背景
5          table.drawSelf(canvas, paint);                          //绘制球台
6          List<Ball> alBallsTemp=new ArrayList<Ball>(alBalls);//复制所有球的列表
7          for(Ball b:alBallsTemp){                                //绘制所有球
8               b.drawSelf(canvas, paint);
9          }
10         cue.drawSelf(canvas, paint);                            //绘制球杆
11         strengthBar.drawSelf(canvas,paint);                     //绘制力度条
12         goBtn.drawSelf(canvas, paint);                          //绘制 GO 图标按钮
13         leftBtn.drawSelf(canvas, paint);                        //绘制左按钮
14         rightBtn.drawSelf(canvas, paint);                       //绘制右按钮
15         aimBtn.drawSelf(canvas, paint);                         //绘制目标按钮
16         if(activity.coundDownModeFlag){
17              timer.drawSelf(canvas, paint);                     //绘制时间
18         }
19    }
```

● 由于各个类都已经封装好，onDraw 方法中的工作原理非常简单，直接调用各个对象的 drawSelf 方法即可，无须关心每个类具体绘制方法的实现，这充分体现了封装的优点。

● 遍历列表中对象之前先要复制列表的原因是，由于其他类中会对列表中对象同时进行删除操作，可能会产生异常。

（3）onTouchEvent 方法的实现。该方法主要负责监听触控事件，并更改相应标志位，其代码如下。将下列代码插入到 HighScoreView 类的第 44 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameView.java。

```
1    public boolean onTouchEvent(MotionEvent event) {
2         float x = event.getX();
3         float y = event.getY();
4                                   //如果正在播放击球动画或不显示球杆，所有触控事件全都不接收
5         if(cue.isShowingAnimFlag() || !cue.isShowCueFlag()){
6              return true;
7         }
8    switch(event.getAction())                                    //事件类型
9    {
10   case MotionEvent.ACTION_DOWN:
11        if(goBtn.isActionOnButton(x, y)){                       //按在 GO 按钮上
12             goBtn.pressDown();                                 //按下 GO 按钮
13        }
14        else if(leftBtn.isActionOnButton(x, y)){                //如果按在左按钮上
15             leftBtn.pressDown();                               //按下左按钮
16             keyState=keyState|0x20;    //和 0010,0000 按位或，第 6 位置 1，标志增加按键时间
17             keyState=keyState|0x1;     //和 00001 按位或，第 1 位置 1，标志向左移
18        }
19        else if(rightBtn.isActionOnButton(x, y)){               //如果按在右按钮上
20             rightBtn.pressDown();                              //按下右按钮
21             keyState=keyState|0x20;    //和 0010,0000 按位或，第 6 位置 1，标志增加按键时间
22             keyState=keyState|0x2;     //和 00010 按位或，第 2 位置 1，标志向右移
23        }
24        else if(aimBtn.isActionOnButton(x, y)){                 //如果按在目标按钮上
25             cue.setAimFlag(!cue.isAimFlag());                  //切换两种瞄准方式
26             if(cue.isAimFlag()){                               //设置不同瞄准方式的按钮图
27                  aimBtn.releaseUp();
28             }
29             else{
30                  aimBtn.pressDown();
31             }
32        }
33        else if(strengthBar.isActionOnBar(x, y)){               //如果按在力度条上
34             strengthBar.changeCurrHeight(x, y);
35        }
36        else{                                                   //没有按在虚拟按钮上
37        cue.calcuAngle(x,' y);                                  //计算球杆旋转角度
38        }
39        break;
40   case MotionEvent.ACTION_MOVE:
41        if(strengthBar.isActionOnBar(x, y)){                    //如果按在力度条上
42             strengthBar.changeCurrHeight(x, y);
43        }
44        else if(!goBtn.isActionOnButton(x, y) &&
45                  !leftBtn.isActionOnButton(x, y) &&
46                  !rightBtn.isActionOnButton(x, y) &&
47                  !aimBtn.isActionOnButton(x, y)
48        )                                                       //没有在力度条和按钮上
49        {
50             goBtn.releaseUp();                                 //松开 GO 按钮
51        keyState=keyState&0xFFEF;       //和 01111 按位与，第 5 位清 0，停止改变力度条
52        leftBtn.releaseUp();                                    //松开左按钮
53        keyState=keyState&0xFFFE;       //和 1110 按位与，第 1 位清 0，清除左移标志
54        rightBtn.releaseUp();                                   //松开右按钮
55        keyState=keyState&0xFFFD;       //和 1101 按位或，第 2 位清 0，清除右移标志
56        btnPressTime=0;                                         //按键时间清 0
```

```
57          keyState=keyState&0xFFDF;        //和 1101,1111 按位与，第 6 位清 0，清除按键时间标志
58          cue.calcuAngle(x, y);                          //计算球杆旋转角度
59          }
60    break;
61    case MotionEvent.ACTION_UP:
62          if(goBtn.isActionOnButton(x, y)){              //按在 GO 按钮上
63               goBtn.releaseUp();                        //松开 GO 按钮
64          new CueAnimateThread(this).start();            //创建播放击球动画的线程，并开启线程
65          }
66          else if(leftBtn.isActionOnButton(x, y)){       //如果按在左按钮上
67               leftBtn.releaseUp();                      //松开左按钮
68          keyState=keyState&0xFFFE;        //和 1110 按位与，第 1 位清 0，清除左移标志
69          btnPressTime=0;                                //按键时间清 0
70          keyState=keyState&0xFFDF;        //和 1101,1111 按位与，第 6 位清 0，清除按键时间标志
71          }
72          else if(rightBtn.isActionOnButton(x, y)){      //如果按在右按钮上
73               rightBtn.releaseUp();                     //松开右按钮
74          keyState=keyState&0xFFFD;        //和 1101 按位或，第 2 位清 0，清除右移标志
75          btnPressTime=0;                                //按键时间清 0
76          keyState=keyState&0xFFDF;        //和 1101,1111 按位与，第 6 位清 0，清除按键时间标志
77          }
78          break;
79    }
80          return true;
81    }
```

- 代码第 14-23 行为按下的是球杆左右转动按钮时，更改增加按键时间和球杆左右转动的状态位。因为有线程一直在监听 keyState 的状态，当状态位改变后，按键时间和球杆角度就会改变。
- 第 24-32 行为按在游戏界面中的目标转换按钮上时切换球杆和目标两种不同瞄准方式的代码。
- 第 40-60 行为触控事件为移动时，动态改变力度条的力度或球杆旋转角度的代码。
- 第 40-60 行为触控事件为抬起时清除各标志位、抬起相应按钮或球杆击球的代码。球杆击球前开启一个线程，播放击球动画。

（4）overGame 方法的实现。该方法在适当的位置被调用后会立即停止游戏，其代码如下。将下列代码插入到 GameView 类的第 56 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 GameView.java。

```
1         public void overGame(){                          //结束游戏的方法
2           stopAllThreads();                              //停止所有线程
3           if(mMediaPlayer.isPlaying()){                  //停止背景音乐
4               mMediaPlayer.stop();
5           }
6           if(activity.coundDownModeFlag){                //如果是倒计时模式
7               activity.currScore=timer.getLeftSecond(); //将总得分赋值给 Activity 中的 score
8               activity.sendMessage(WhatMessage.OVER_GAME);    //发送结束游戏的请求
9           }
10          else{
11              activity.sendMessage(WhatMessage.GOTO_CHOICE_VIEW);//发送到积分榜界面的请求
12        }}
```

提示　游戏结束的方法代码虽然简短，但其实并不简单。因为方法都已封装好，这里直接对方法调用，读者可以查看各方法的源代码，理清其中逻辑。

17.6.3　键盘监听线程 KeyThread 类的实现

KeyThread 类主要负责不断读取 GameView 类中的整型变量 keyState 各个位的值，根据各个位值的不同做出不同的反应。其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 KeyThread.java。

```
1    package com.bn.d2.bill;                                    //声明包
2    public class KeyThread extends Thread {
3         private boolean flag=true;
4         GameView gameView;                                    //gameView 的引用
5         private int sleepSpan=40;                             //线程睡眠时间
6         private float changeSpeedTime=80f;                    //改变速度的时间点
7         public KeyThread(GameView gameView){
8              this.gameView=gameView;
9         }
10        public void run(){                                    //重写 run 方法
11           while(flag){
12                 if(!((gameView.keyState&0x20)==0)){//第 6 位是 1，可以改变按下按钮时间
13                      gameView.btnPressTime+=3.5f;
14                 }
15                 if(!((gameView.keyState&0x1)==0)){ //和 00001 按位或，判断第 1 位是否为
                                                      //1，标志向左移
16                      if(gameView.btnPressTime<changeSpeedTime){//如果按键时间不足
17                           gameView.cue.rotateLeftSlowly();    //将球杆慢速左转
18                      }
19                      else{                                   //如果按键超过规定时间
20                           gameView.cue.rotateLeftFast();  //将球杆快速左转
21                      }
22                 }
23                 else if(!((gameView.keyState&0x2)==0)){   //和 00010 按位或，判断
                                                             //第 2 位是否为 1，标志右移
24                      if(gameView.btnPressTime<changeSpeedTime){//如果按键时间不足
25                           gameView.cue.rotateRightSlowly();   //将球杆慢速右转
26                      }
27                      else{                                   //如果按键超过规定时间
28                           gameView.cue.rotateRightFast();//将球杆快速右转
29                      }
30                 }
31                 try{
32                 Thread.sleep(sleepSpan);                     //睡眠指定毫秒数
33              }
34              catch(Exception e){
35                 e.printStackTrace();                         //打印堆栈信息
36              }
37           }
38        }
39        public void setFlag(boolean flag) {                  //设置循环标志位的方法
40           this.flag = flag;
41   }}
```

- 第 12-14 行为 keyState 的第 6 位为 1 时，增加按下按钮时间的代码。
- 第 15-30 行为 keyState 的第 1 位或第 2 位为 1 时，更改球杆位置的操作。当按下的时间短时球杆转动速度慢，但较精细；当按下的时间超过某个值后，球杆会快速转动，但精度很粗。

17.6.4 球运动的线程 BallGoThread 类的实现

BallGoThread 类主要负责所有桌球运动，处理球运动后状态。其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 BallGoThread.java。

```
1    package com.bn.d2.bill;                                          //声明包
2    import java.util.ArrayList;                                      //引入相关类
3    public class BallGoThread extends Thread{                        //桌球运动的线程
4        GameView gameView;
5        private boolean flag=true;                                   //线程是否继续工作的标志位
6        ArrayList<Ball> ballsToDelete=new ArrayList<Ball>();//要删除的球的列表
7        private int sleepSpan=7;                                     //线程睡眠时间
8        public BallGoThread(GameView gameView){                      //构造器
9            this.gameView=gameView;
10       }
11       public void run(){                                           //重写 run 方法
12           while(flag){
13               ballsToDelete.clear();                               //清除要删除的列表
14               for(Ball b:gameView.alBalls){                        //让所有球运动
15                   b.go();
16                   if(b.isInHole()){                                //每走一步判断是否进洞
17                       if(b==gameView.alBalls.get(0)){//如果是白球进洞
18                           b.hide();                                //将白球隐藏起来，但不删除
19                       }
20                       else{                                        //如果普通球进洞
21                           ballsToDelete.add(b);                    //将进洞的球加入删除列表
22                       }
23                   }
24               }
25               gameView.alBalls.removeAll(ballsToDelete);   //从列表中删除进洞的球
26               //判断是否可以显示球杆并击球
27               boolean allBallsStoppedFlag=true;                    //所有球是否都停了的标志
28               for(Ball b:gameView.alBalls){
29                   if(!b.isStoped()){      //只要有一个球还没有停，标志位置为 false
30                       allBallsStoppedFlag=false;
31                       break;
32                   }
33               }
34               if(allBallsStoppedFlag){                             //如果所有球都停了
35                   if(gameView.alBalls.get(0).isHided()){ //如果白球因进洞而被隐藏
36                       gameView.alBalls.get(0).reset();  //将白球重新复位
37                   }
38                   gameView.cue.setShowCueFlag(true);     //显示球杆
39                   if(gameView.alBalls.size()<=1){        //如果球全部进洞，游戏结束
40                       gameView.overGame();
41                   }
42               }
43               try {
44                   Thread.sleep(sleepSpan);               //线程休眠一定的时间
45               } catch (InterruptedException e) {
46                   e.printStackTrace();                   //打印异常
47               }
48           }
49       }
50       public void setFlag(boolean flag) {                          //设置循环标志位的方法
51           this.flag = flag;
52   }}
```

- 代码第 13-24 行使所有的球运动，球运动之后判断球是否进洞，如果是普通球进洞，从列

表中删除该球，如果是白球进洞，将白球隐藏进来，并不删除。

- 代码第 26-41 行判断是否可以显示球杆并击球，如果所有的球都静止了，查看白球是否隐藏，并显示白球和球杆，再判断球是否全部进洞，如果全部进洞，结束游戏。

说明

CueAnimateThread 、TimeRunningThread 和 GameViewDrawThread 3 个线程类非常简单，因此这里不再赘述，请读者自行查看随书光盘中的源代码。

17.7 情景相关类的实现

前面章节中用到了很多读者并不熟悉的情景中的实体类，从本节开始将对这些类一一进行讲解。首先介绍的是球台类。

17.7.1 球台 Table 类的实现

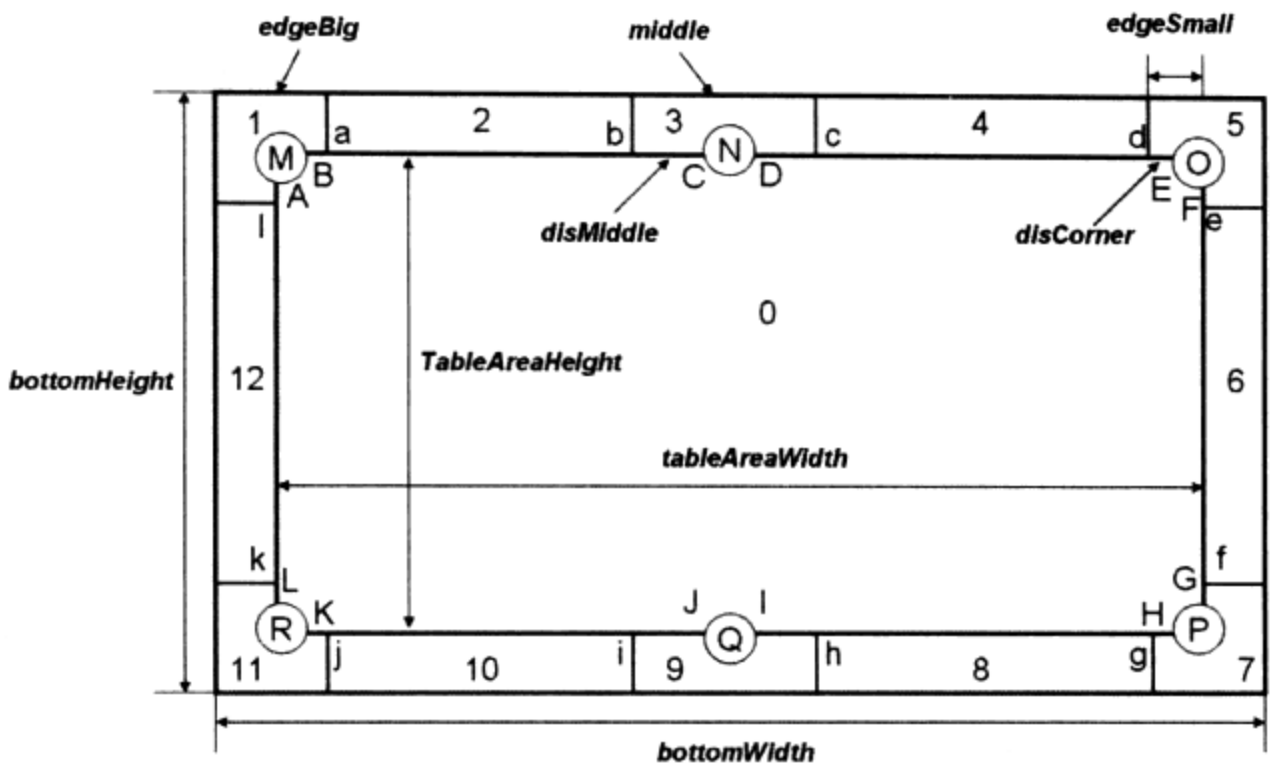
Table 类里主要封装了和球台相关的常量值，由于这些值在各个类中都可能用到，因此，这些常量值大都是静态的，其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 Table.java。

```
1    package com.bn.d2.bill;                                          //声明包
2    import android.graphics.Bitmap;                                  //引入相关类
3    import android.graphics.Canvas;                                  //引入相关类
4    import android.graphics.Paint;                                   //引入相关类
5    //球台类
6    public class Table {
7         static final float x=Constant.TABLE_X;                      //球台左上角位置
8         static final float y=Constant.TABLE_Y;
9         Bitmap[] bitmaps;
10                                                                    //Constant 类中的常量值
11        static final float bottomWidth=Constant.BOTTOM_WIDTH;    //整个球台面宽度
12        static final float bottomHeight=Constant.BOTTOM_HEIGHT;
13        static final float edgeBig=Constant.EDGE_BIG;            //大边缘尺寸
14        static final float edgeSmall=Constant.EDGE_SMALL;
15        static final float middle=Constant.MIDDLE;
16        static final float disCorner=Constant.DIS_CORNER; //图片左上顶点和角上 4 个洞的距离
17        static final float disMiddle=Constant.DIS_MIDDLE;
18                                                                    //桌面尺寸
19        static final float tableAreaWidth=bottomWidth-edgeBig*2;
20        static final float TableAreaHeight=bottomHeight-edgeBig*2;
21                                                                    //各直线长度
22        static final float ab=(tableAreaWidth-middle)/2-edgeSmall;
23        static final float ef=TableAreaHeight-edgeSmall*2;
24                                                                    //各直线的坐标
25        static final float lkx=x+edgeBig;                          //直线 lk 的 x 坐标
26        static final float efx=lkx+tableAreaWidth;
27        static final float ady=y+edgeBig;
```

```
28          static final float jgy=ady+TableAreaHeight;
29                                                                  //各角的坐标
30          static final float A[]={lkx+edgeSmall-disCorner, ady};
31          static final float B[]={lkx+edgeSmall+ab+disMiddle, ady};
32          ……//此处省略了部分代码，读者可自行查看随书光盘的源代码
33          static final float L[]={lkx, ady+edgeSmall-disCorner};
34          static final float collisionPoints[][]={A,B,C,D,E,F,G,H,I,J,K,L};//所有碰撞点
35          //所有球洞的中心坐标
36          static final float holeCenterRevise=Constant.HOLE_CENTER_REVISE;
                                                                    //球洞中心坐标修正值
37          static final float M[]={lkx,ady};
38          static final float N[]={lkx+edgeSmall+ab+middle/2, ady-holeCenterRevise};
39          static final float O[]={efx,ady};
40          static final float P[]={efx,jgy};
41          static final float Q[]={lkx+edgeSmall+ab+middle/2, jgy+holeCenterRevise-2};
42          static final float R[]={lkx,jgy};
43          static final float holeCenterPoints[][]={M,N,O,P,Q,R};   //所有洞的中心点
44          static final float cornerHoleR=Constant.CORNER_HOLE_R;   //洞的半径
45          static final float middleHoleR=Constant.MIDDLE_HOLE_R;   //洞的半径
46                                                          //关于球台上球初始位置的常量
47          static final float xBall1=lkx+Constant.X_OFFESET_BALL1; //一号球的位置
48          static final float yBall1=ady+TableAreaHeight/2-Ball.r;
49          static final float xBallDis=(float) ((Ball.d+Constant.GAP_BETWEEN_BALLS)*Math.
            sin(Math.PI/3));
50          static final float yBallDis=(float) ((Ball.d+Constant.GAP_BETWEEN_BALLS)*Math.
            cos(Math.PI/3));
51          static final float yDis=Ball.d+Constant.GAP_BETWEEN_BALLS;   //每一列球的距离
52                                          //所有球的位置，0 为母球，以后依次从左到右，从下到上
53          static final float[][] AllBallsPos={
54               new float[]{xBall1-Constant.DIS_WITH_MAIN_BALL,yBall1},
                                            //母球位置,ady+1,jgy-Ball.d-1
55               new float[]{xBall1,yBall1},                        //1 列
56               ……//此处省略了部分代码，读者可自行查看随书光盘的源代码
57               new float[]{xBall1+xBallDis*4,yBall1+yBallDis*4-yDis*4},
58          };
59          final float[][] bmpsPos={                               //所有图片位置坐标
60                    new float[]{lkx,ady},                         //0
61                    new float[]{x,y},                             //1
62                    ……//此处省略了部分代码，读者可自行查看随书光盘的源代码
63                    new float[]{x, ady+edgeSmall},                //12
64               };
65          public Table(Bitmap[] bitmaps){                         //构造器
66               this.bitmaps=bitmaps;
67          }
68          public void drawSelf(Canvas canvas,Paint paint){        //绘制球台的方法
69               for(int i=0;i<bitmaps.length;i++){
70                    canvas.drawBitmap(bitmaps[i], bmpsPos[i][0]+Constant.X_OFFSET,
71                              bmpsPos[i][1]+Constant.Y_OFFSET, paint);
72     }}}
```

- 代码第 7-45 行为球台尺寸及各点坐标，球台示意图如图 17-16 所示。



▲图 17-16　球台示意图

说明　数字 0～12 是球台面分解后的各小部件的编号；小写字母 a～i 是各小部件顶点的坐标；大写字母 A～L 是碰撞点的坐标；M～R 是 4 个球洞的中心点坐标。

- 代码第 46-58 行为球台上球初始位置的坐标，注意，球与球之间的初始位置是有空隙的，否则开杆时的碰撞检测可能不成功，调节 GAP_BETWEEN_BALLS 值可以改变初始空隙的大小。
- 代码第 68-71 行是绘制球台的方法。从示意图中可以看到，球台是拆散了，然后重新组合起来的，这样做的目的是方便单独调节某个组成部分的大小，可以更精准地测量各个量的长度。

17.7.2　桌球 Ball 类的代码框架

上一小节中介绍了 Table 类的实现，本小节将介绍 Ball 类的代码框架。Ball 类的成员方法有很多，其中有一部分方法只是为白球服务的，其代码框架如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 Ball.java。

```
1    package com.bn.d2.bill;                                  //声明包
2    import android.graphics.Bitmap;                          //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.graphics.Paint;                           //引入相关类
5    public class Ball {
6         private float x;                                    //球的外接正方形左上角顶点坐标
7         private float y;
8         static final float r=Constant.BALL_SIZE/2;          //球的半径
9         static final float d=Constant.BALL_SIZE;            //球的直径
10        static final float vMax=Constant.V_MAX;             //球的最大速度
11        float vx;                                           //球的运动速度
12        float vy;
```

```
13          float timeSpan=Constant.TIME_SPAN;                  //球运动的模拟时间间隔
14          float vAttenuation=Constant.V_ATTENUATION;          //速度衰减比例
15          float vMin=Constant.V_MIN;                  //速度最小值，当速度小于此值时球停止运动
16          Bitmap[] bitmaps;
17          private int bmpIndex=0;
18          float bmpIndexf=0;                                  //辅助索引
19          Table table;                                        //球台
20          GameView gameView;
21          boolean InHoleflag=false;                           //球是否进洞的标志
22          private float rotateX;
23          private float rotateY;
24          private float angdeg=0;                             //旋转角度
25          public Ball(Bitmap[] bitmaps,GameView gameView,float vx,float vy,float[] pos){
26               this.bitmaps=bitmaps;                          //成员变量赋值
27               this.gameView=gameView;
28               this.vx=vx;
29               this.vy=vy;
30               this.x=pos[0];                                 //成员变量赋值
31               this.y=pos[1];
32               this.table=gameView.table;
33               rotateX=bitmaps[0].getWidth()/2;               //成员变量赋值
34               rotateY=bitmaps[0].getHeight()/2;
35          }
36          public void drawSelf(Canvas canvas,Paint paint){}//绘制桌球的方法
37          public void calAngle(){}                            //计算旋转角度的方法
38          public float getX() {                               //获取 x 值的方法
39               return x;
40          }
41          public float getY() {                               //获取 y 值的方法
42               return y;
43          }
44          public boolean canGo(float tempX,float tempY){} //判断是否能走的方法
45          public void go(){}
46          public boolean isStoped(){                          //判断球是否停止的方法
47               return (vx==0 && vy==0);
48          }
49          public void changeVxy(float v,float angle){         //根据速度和方向改变球速度的方法
50               double angrad=Math.toRadians(angle);           //将角度转换成弧度
51               vx=(float) (v*Math.cos(angrad));               //计算 x,y 方向的速度
52               vy=(float) (v*Math.sin(angrad));
53          }
54          public boolean isInHole(){                          //判断球是否进洞的方法
55               return InHoleflag;
56          }
57          public void hide(){                                 //隐藏球的方法
58               vx=vy=0;
59               x=y=-100000;
60          }
61          public boolean isHided(){                           //判断球是否隐藏起来的方法
62               return y==-100000;
63          }
64          public void reset(){}                               //将球重新复位的方法
65     }
```

- 代码第 6-24 行为 Ball 类成员变量的定义，其中包括了球的所有属性以及球台对象的引用，这些属性值都是为后面的成员方法服务的。
- 代码第 49-53 行为根据速度和方向改变球速度的方法，该方法的参数 v 是通过力度条的力度计算出的，angle 是球杆的旋转角度，通过这两个量就可以算出击球后球的 xy 方向上的分速度。
- 代码第 54-63 行是关于白球进洞的 3 个方法，注意，白球和其他的球进洞的处理方式不一

样，白球进洞后只是将其位置设在一个很远的地方，并没有真正意义上的删除，只是隐藏了起来。

说明　代码框架中的部分方法为空实现，部分方法的实现将在下一小节中详细介绍，其中有些方法很简单，后面没有给出具体代码，请读者自行查看随书光盘中的源代码。

17.7.3　Ball 类部分成员方法的实现

上面介绍了 GameView 类的代码框架，本小节将介绍其部分成员方法的具体实现。

（1）drawSelf 和 calAngle 方法的实现。这两个方法是密不可分的，把两个方法分开来写是为了使代码更加清晰，其代码如下。将下列代码插入到 Ball 类的第 36 和 37 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 Ball.java。

```
1    public void drawSelf(Canvas canvas,Paint paint){
2          calAngle();                                          //计算角度
3          Matrix m1=new Matrix();                              //平移矩阵
4          m1.setTranslate(x+Constant.X_OFFSET,y+Constant.Y_OFFSET);
5          Matrix m2=new Matrix();                              //旋转矩阵
6          m2.setRotate(angdeg, rotateX, rotateY);
7          Matrix mz=new Matrix();                              //综合矩阵
8          mz.setConcat(m1, m2);
9          canvas.drawBitmap(bitmaps[bmpIndex], mz,paint);      //绘制球
10   }
11   public void calAngle(){                                    //计算角度的方法
12         if(vx==0&&vy==0){
13               bmpIndex=0;
14               return;
15         }
16                                                              //根据球的速度、方向，计算球应旋转的角度
17         float dirX=vx;
18         float dirY=vy;
19         if(dirY>=0){
20               angdeg=(float) Math.toDegrees((Math.atan(-dirX/dirY)+Math.PI/2));
21         }
22         else if(dirY<0){
23               angdeg=(float) Math.toDegrees((Math.atan(-dirX/dirY)+Math.PI*3/2));
24         }
25   }
```

- 代码第 1-10 行为 drawSelf 方法的实现，在该方法中首先计算图片应旋转的角度，再用矩阵将图片移到相应的位置，并旋转前面计算出的角度，并绘制索引 bmpIndex 对应的图片。
- 代码第 11-25 行为计算图片旋转角度的方法，首先，如果球的速度是 0，将图的索引值设为 0，保证球静止时有数字的一面朝上。然后再根据球的运动方向计算球应旋转的角度。

（2）go 方法的实现。该方法主要负责球的位置和速度的改变，以及球图片的换帧，其代码如下。将下列代码插入到 Ball 类的第 45 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 Ball.java。

```
1    public void go(){
2          if(isStoped()||Math.sqrt(vx*vx+vy*vy)<vMin){     //速度小于阈值，球停止运动
3                vx=0;                                       //速度置为 0
4                vy=0;
```

```
5                bmpIndex=0;
6                return;
7            }
8            float tempX=x+vx*timeSpan;                          //球要运动到的下一个位置
9            float tempY=y+vy*timeSpan;
10           if(this.canGo(tempX, tempY)){
11               x=tempX;
12               y=tempY;
13               float v=(float) Math.sqrt(vx*vx+vy*vy);     //根据速度大小计算换图频数
14               bmpIndexf=bmpIndexf+Constant.K*v;
15               bmpIndex=(int)(bmpIndexf)%bitmaps.length;  //采样
16               vx*=vAttenuation;                           //每移动一步，速度衰减
17               vy*=vAttenuation;
18           }
19       }
```

- 代码第 2-7 行如果球的速度小于阈值，将球的速度设为 0，使球停止运动。
- 代码第 8-17 行首先，算出球将要运动到的下一个位置，再判断球是否能走到该位置，如果可以到达，再将要运动到的位置赋值给球真正的位置，然后用采样法，根据速度大小计算换帧的频数，最后，球每移动一步，将速度衰减。

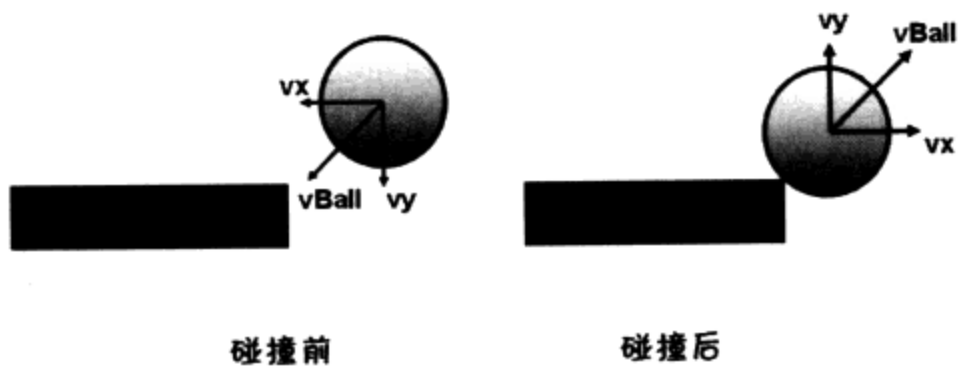
（3）canGo 方法的实现。该方法负责判断球是否能到达一个指定的位置，如果不能到达，根据不同的情况，改变球的速度，其代码如下。将下列代码插入到 Ball 类的第 44 行。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 Ball.java。

```
1    public boolean canGo(float tempX,float tempY){        //判断球是否能达到指定位置的方法
2        boolean canGoFlag=true;
3        for(Ball b:gameView.alBalls){
4            if(b!=this && CollisionUtil.collisionCalculate(new float[]{tempX,tempY},
             this, b)){
5                if(gameView.activity.isSoundOn()){    //播放球碰撞的声音
6                    gameView.playSound(GameView.HIT_SOUND, 0);
7                }
8                canGoFlag=false;
9        }}
10       if(canGoFlag==false){                    //如果和球碰撞了就不再考虑和球台的碰撞
11           return false;
12       }
13       //判断球是否进洞
14       for(int i=0;i<Table.holeCenterPoints.length;i++){    //循环洞的中心坐标数组
15           float disWithHole=CollisionUtil.mould(          //计算球心和洞中心的距离
16                   new float[]{tempX+Ball.r-Table.holeCenterPoints[i][0],
17                               tempY+Ball.r-Table.holeCenterPoints[i][1]});
18           //判断球是否进中洞
19           if(Table.holeCenterPoints[i]==Table.N  ||  Table.holeCenterPoints[i]==
             Table.Q){
20               if(disWithHole<=Table.middleHoleR){  //如果球心在洞内，判定为进洞
21                   InHoleflag=true;
22                   break;
23               }
24           }else{                                   //判断球是否进角上的洞
25               if(disWithHole<=Table.cornerHoleR){  //如果球心在洞内，判定为进洞
```

```
26                      InHoleflag=true;
27                      break;
28          }}}
29          if(InHoleflag==true){                    //如果球进洞，返回 true，不再做其他判断
30              if(gameView.activity.isSoundOn()){          //播放球进洞的声音
31                  gameView.playSound(GameView.BALL_IN_SOUND, 0);
32              }
33              return true;
34          }
35          //判断球是否和角有碰撞
36          float center[]={tempX+r,tempY+r};                //球心位置
37          boolean collisionWithCornerFlag=false;
38          for(float[] p:Table.collisionPoints){
39              if(CollisionUtil.calcuDisSquare(center, p)<=r*r){   //判断球是否和角碰撞
40                  collisionWithCornerFlag=true;
41                  canGoFlag=false;
42                  vx=-vx;                          //与角碰撞，x,y 速度全反向
43                  vy=-vy;
44                  break;
45          }}
46          if(!collisionWithCornerFlag){                //如果没有与角碰撞，再判断是否与边碰撞
47              if(tempX<=Table.lkx||tempX+d>=Table.efx){  //球的位置超出球台
48                  vx=-vx;                              //与竖边碰撞，x 速度反向
49                  canGoFlag=false;
50              }else if(tempY<=Table.ady||tempY+d>=Table.jgy){
51                  vy=-vy;                              //与横边碰撞，y 速度反向
52                  canGoFlag=false;
53          }}
54          return canGoFlag;                            //返回标志位
55      }
```

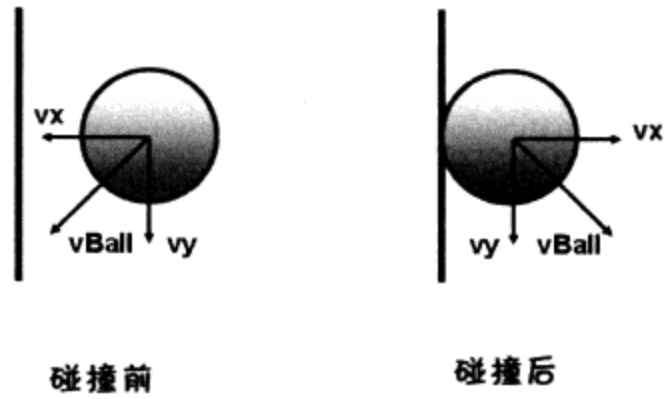
- 代码第 3-12 行先判断了球和球之间是否有碰撞，其具体细节将在讲解 CollisionUtil 类时给出具体介绍，在这里读者只需要知道代码的大致含义即可。
- 代码第 13-28 行判断球是否进洞，由于洞的中心坐标是在 Table 类中二维数组中存放着，所以，只要用一个循环就可以判断是否和所有的洞是否有“碰撞”，注意，这里用的球进洞的算法是判断球的中心是否在洞里，判断时还设了调整值，以保证球进洞的难易程度适中。
- 代码第 35-45 行判断球是否和角有碰撞，示意图如图 17-17 所示。



▲图 17-17　球与角碰撞示意图

提示　球与角碰撞后处理的算法为，将球的两个分速度 *vx*、*vy* 分别置反，就可以实现球的反向。

- 代码第 46-55 行判断球是否和边碰撞，示意图如图 17-18 所示。



▲图 17-18　球与边碰撞示意图

提示　球与边碰撞后处理的算法为，将球的相应方向的分速度置反，就实现了球和边碰撞的效果。

17.7.4　球杆 Cue 类的实现

Cue 类主要负责绘制球杆和找准线，用来控制击球的方向，其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 Cue.java。

```
1    package com.bn.d2.bill;                                          //声明包
2    import android.graphics.Bitmap;                                  //引入相关类
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.graphics.Paint;                                   //引入相关类
5    public class Cue {                                               //球杆类
6         float x;                                                    //左上角位置
7         float y;
8         float rotateX;                                              //旋转中心点
9         float rotateY;
10        private float angdeg=0;                                     //旋转角度
11        float width;                                                //球杆宽度
12        float height;                                               //球杆高度
13        float disWithBall=Constant.DIS_WITH_BALL;                   //球杆与母球边缘的距离
14        Bitmap bitmap;                                              //位图
15        Ball mainBall;                                              //母球
16        private boolean showCueFlag=true;                           //是否显示球杆的标志
17        private final float angleSpanSlow=0.2f;                     //微调角度步进
18        private final float angleSpanFast=1f;                       //粗调角度步进
19        private boolean aimFlag=true;
20        private final float lineLength=Table.tableAreaWidth;//瞄准线的长度
21        //关于击球动画的量
```

```
22          private final  float backSpan=3;                    //球杆向后退的步进
23          private final float forwardSpan=10;                 //球杆前进时的步进
24          private final float maxDis=50;                      //球杆向后退的最大距离
25          private float span=backSpan;                        //击球时球杆步进
26          private boolean showingAnimFlag=false;              //是否正在播放击球动画的标志
27          public Cue(Bitmap bitmap,Ball mainBall){            //构造器
28               this.bitmap=bitmap;
29               this.mainBall=mainBall;
30               this.width=bitmap.getWidth();                  //成员变量赋值
31               this.height=bitmap.getHeight();
32          }
33          public void drawSelf(Canvas canvas,Paint paint){
34               if(!showCueFlag){                              //如果不能显示球杆，直接返回
35                    return;
36               }
37               this.rotateX=this.width+this.disWithBall+Ball.r;//计算球杆旋转中心点位置
38               this.rotateY=this.height/2;
39               x=mainBall.getX()-width-disWithBall;                //计算球杆的位置
40               y=mainBall.getY()+Ball.r-height/2;
41               //绘制球杆
42               Matrix m1=new Matrix();                             //平移矩阵
43               m1.setTranslate(x+Constant.X_OFFSET,y+Constant.Y_OFFSET);
44               Matrix m2=new Matrix();                             //旋转矩阵
45               m2.setRotate(angdeg, rotateX, rotateY);
46               Matrix mz=new Matrix();                             //综合矩阵
47               mz.setConcat(m1, m2);
48               canvas.drawBitmap(bitmap, mz,paint);                //绘制球杆
49               //绘制瞄准线
50               canvas.save();                                      //保存画布
51               canvas.clipRect(Table.lkx+Constant.X_OFFSET, Table.ady+Constant.Y_OFFSET,
52                         Table.efx+Constant.X_OFFSET, Table.jgy+Constant.Y_OFFSET);
                                                                     //划定矩形区域
53               float angrad=(float) Math.toRadians(angdeg);        //将角度转换成弧度
54               float startX=(float) (mainBall.getX()+Constant.X_OFFSET+Ball.r+Ball.
                 r*Math.cos(angrad));
55               float startY=(float) (mainBall.getY()+Constant.Y_OFFSET+Ball.r+Ball.
                 r*Math.sin(angrad));
56               float stopX=startX+(float)(lineLength*Math.cos(angrad));//瞄准线终点坐标
57               float stopY=startY+(float)(lineLength*Math.sin(angrad));
58               paint.setColor(Color.YELLOW);                       //画笔为黄色
59               paint.setAlpha(240);                                //设置画笔不透明度
60               canvas.drawLine(startX, startY, stopX, stopY, paint);   //绘制瞄准线
61               canvas.restore();                                   //恢复画布
62               paint.setAlpha(255);                                //恢复画笔透明度为不透明
63          }
64          public void calcuAngle(float pressX,float pressY){    //根据触控屏幕的位置计算球
                                                                  //杆旋转角度的方法
65               float dirX=pressX-(mainBall.getX()+Ball.r+Constant.X_OFFSET);
                                                                  //球杆的方向向量
```

```
66              float dirY=pressY-(mainBall.getY()+Ball.r+Constant.Y_OFFSET);
67              if(!aimFlag){                          //如果不是按照目标确定旋转角度，将方向向量置反
68                   dirX = -dirX;
69                   dirY = -dirY;
70              }
71                                                     //根据球杆的方向计算球杆应旋转的角度
72              if(dirY>=0){
73                   angdeg=(float) Math.toDegrees((Math.atan(-dirX/dirY)+Math.PI/2));
74              }
75              else if(dirY<0){
76                   angdeg=(float) Math.toDegrees((Math.atan(-dirX/dirY)+Math.PI*3/2));
77         }}
78         ……//此处省略了部分简单的成员方法，请读者自行查看随书光盘中的源代码。
79    }
```

- 代码第 6-32 行为该类中成员方法和构造器，构造器中为成员变量赋值，参数 mainBall 指白球的引用，因为球杆只会击白球，所以没有必要把球的列表传递进来。
- 代码第 33-63 行为绘制球杆和瞄准线的方法，首先，计算出球杆旋转中心点位置和球杆位置，然后，通过矩阵绘制球杆，再通过圈定画布的某个范围绘制球杆的瞄准线。
- 代码第 64-79 行为根据触控屏幕的位置计算球杆旋转角度的方法，其中 65-70 行计算出球杆的方向向量后，判断目标和球杆的切换按钮的状态，相应地更改方向向量的值。

17.8 自定义控件及工具类的实现

上一节介绍了情景相关类的实现，本节开始介绍自定义控件及工具类的实现。其中包括球之间碰撞检测的工具类、自适应屏的工具类、虚拟按钮、定时器、力度条以及获取系统日期的工具类等，下面对各类进行详细介绍。

17.8.1 球与球碰撞检测的工具类 CollisionUtil 的实现

CollisionUtil 类的作用是检测球和球之间的碰撞，并在碰撞后改变球的速度，其代码如下。

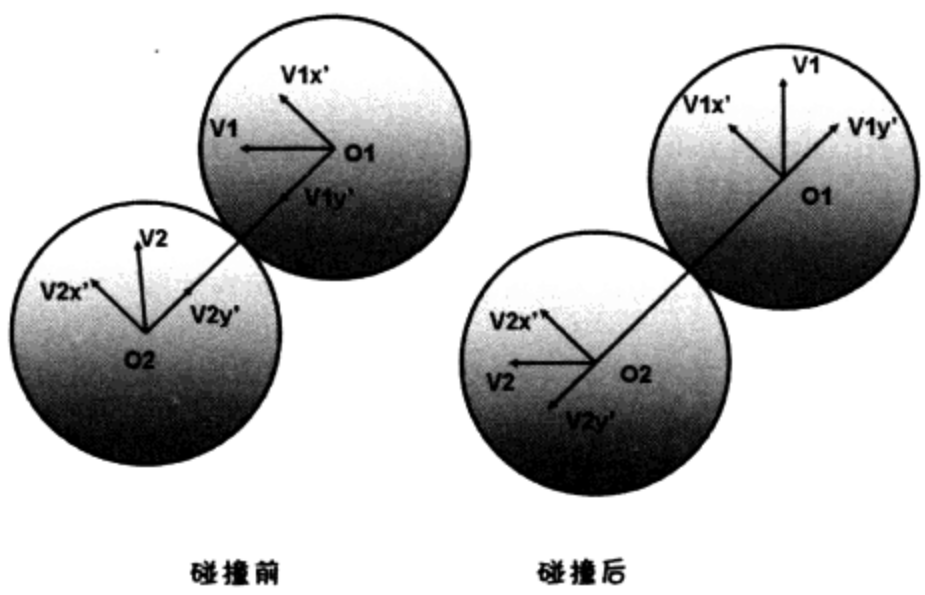
代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 StrengthBar.java。

```
1     package com.bn.d2.bill;                          //声明包
2     public class CollisionUtil{                      //进行碰撞检测与物理计算的工具类
3          public static float dotProduct(float[] vec1,float[] vec2){//求两个平面向量的点积
4               return
5                    vec1[0]*vec2[0]+
6                    vec1[1]*vec2[1];
7          }
8          public static float mould(float[] vec){     //求平面向量的模
9               return (float)Math.sqrt(vec[0]*vec[0]+vec[1]*vec[1]);
10         }
11         public static float angle(float[] vec1,float[] vec2){    //求两个平面向量的夹角
12              float dp=dotProduct(vec1,vec2);        //先求点积
13              float m1=mould(vec1);                  //再求两个向量的模
```

```
14              float m2=mould(vec2);
15              float acos=dp/(m1*m2);
16              if(acos>1){                                  //为了避免计算误差带来的问题
17                   acos=1;
18              }
19              else if(acos<-1){
20                   acos=-1;
21              }
22              return (float) Math.acos(acos);              //返回的是弧度
23         }
24         public static float calcuDisSquare(float[] p1,float[] p2){ //球两点距离的平方
25              return (p1[0]-p2[0])*(p1[0]-p2[0])+(p1[1]-p2[1])*(p1[1]-p2[1]);
26         }
27         //检测两个球是否碰撞的方法
28         public static boolean isTwoBallsCollided(float ballaTempXY[],Ball ballb){
29              float BAx=ballaTempXY[0]-ballb.getX();
30              float BAy=ballaTempXY[1]-ballb.getY();
31              float mvBA=mould(new float[]{BAx,BAy});    //求上述向量的模
32              return (mvBA<Ball.d);
33         }
34         //两个球进行碰撞物理计算的方法
35         public static boolean collisionCalculate(float ballaTempXY[],Ball balla,Ball
           ballb){
36              float BAx=ballaTempXY[0]-ballb.getX();
37              float BAy=ballaTempXY[1]-ballb.getY();
38              float mvBA=mould(new float[]{BAx,BAy});    //求上述向量的模
39              if(mvBA>Ball.d){                             //两球距离大于等于球直径则没有碰撞
40                   return false;
41              }
42              float vB=(float)Math.sqrt(ballb.vx*ballb.vx+ballb.vy*ballb.vy);
                                                             //求 B 球的速度大小
43              float vbCollX=0;                             //平行方向的 X 分速度
44              float vbCollY=0;                             //平行方向的 y 分速度
45              float vbVerticalX=0;                         //垂直方向的 X 分速度
46              float vbVerticalY=0;                         //垂直方向的 y 分速度
47              //若球速度小于阈值则认为速度为零，不用进行分解计算了
48              if(balla.vMin<vB){
49                                      //求 B 球的速度方向向量与碰撞直线向量 B→A 的夹角(弧度)
50                   float bAngle=angle(
51                        new float[]{ballb.vx,ballb.vy},
52                      new float[]{BAx,BAy}
53                   );
54                   float vbColl=vB*(float)Math.cos(bAngle);//求 B 球在碰撞方向的速度大小
55                   vbCollX=(vbColl/mvBA)*BAx;              //求 B 球在碰撞方向的速度向量
56                   vbCollY=(vbColl/mvBA)*BAy;
57                   vbVerticalX=ballb.vx-vbCollX;           //求 B 球在碰撞垂直方向的速度向量
58                   vbVerticalY=ballb.vy-vbCollY;
59              }
60              float vA=(float)Math.sqrt(balla.vx*balla.vx+balla.vy*balla.vy);
                                                             //求 A 球的速度大小
```

```
61              float vaCollX=0;                          //平行方向的 Xy 分速度
62              float vaCollY=0;
63              float vaVerticalX=0;                      //垂直方向的 Xy 分速度
64              float vaVerticalY=0;
65              //若球速度小于阈值则认为速度为零，不用进行分解计算了
66              if(balla.vMin<vA){
67                   float aAngle=angle(//求 A 球的速度方向向量与碰撞直线向量 B→A 的夹角(弧度)
68                        new float[]{balla.vx,balla.vy},
69                      new float[]{BAx,BAy}
70                   );
71                   float vaColl=vA*(float)Math.cos(aAngle); //求 A 球在碰撞方向的速度大小
72                   vaCollX=(vaColl/mvBA)*BAx;           //求 A 球在碰撞方向的速度向量
73                   vaCollY=(vaColl/mvBA)*BAy;
74                   vaVerticalX=balla.vx-vaCollX;        //求 A 球在碰撞垂直方向的速度向量
75                   vaVerticalY=balla.vy-vaCollY;
76              }
77              balla.vx=vaVerticalX+vbCollX;             //求碰撞后 A、B 球的速度
78              balla.vy=vaVerticalY+vbCollY;
79              ballb.vx=vbVerticalX+vaCollX;
80              ballb.vy=vbVerticalY+vaCollY;
81              return true;
82    }}
```

- 代码第 3-26 行是关于向量的几个工具方法，这些方法用到的全是数学知识，在这里不做过多介绍，读者要知道其大致含意。
- 代码第 35-82 行为检测两个球是否碰撞，并根据碰撞的物理知识，改变两球速度的方法，示意图如图 17-19 所示。



▲图 17-19　球与球碰撞示意图

提示　两球进行碰撞的算法为：1、计算两球球心连线的向量 2、将每个球的速度分解为平行与垂直两部分 3、根据完全弹性碰撞的知识，碰撞后平行于向量的速度两球交换，垂直于向量的速度不变 4、重新组装两球的新速度。

17.8.2　定时器 Timer 类的实现

Timer 类的主要功能是游戏倒计时，并将时间以图片的形式显示在游戏界面中。当时间为 0 时游戏结束，其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 Timer.java。

```
1    package com.bn.d2.bill;                                    //声明包
2    import android.graphics.Bitmap;                            //引入相关类
3    import android.graphics.Canvas;                            //引入相关类
4    import android.graphics.Paint;                             //引入相关类
5    public class Timer {
6         GameView gameView;
7         private final int totalSecond=60;                     //总秒数
8         private Bitmap breakMarkBitmap;                       //时间分隔符的位图
9         private Bitmap[] numberBitmaps;                       //数字位图
10        private int leftSecond=totalSecond;
11        int endX=Constant.SCREEN_WIDTH*2/3;                   //数字的右上点坐标
12        int endY=(int) Constant.BAR_Y;
13        int numberWidth;
14        int numberHeight;
15        int breakMarkWidth;
16        int breakMarkHeight;
17        public Timer(GameView gameView,Bitmap breakMarkBitmap,Bitmap[] numberBitmaps){
                                                                //构造器
18             this.gameView=gameView;                          //成员变量赋值
19             this.breakMarkBitmap=breakMarkBitmap;
20             this.numberBitmaps=numberBitmaps;
21             numberWidth=numberBitmaps[0].getWidth();         //成员变量赋值
22             numberHeight=numberBitmaps[0].getHeight();
23             breakMarkWidth=breakMarkBitmap.getWidth();       //成员变量赋值
24             breakMarkHeight=breakMarkBitmap.getHeight();
25        }
26        public void drawSelf(Canvas canvas,Paint paint){     //绘制时间的方法
27             int second=this.leftSecond%60;                   //得到秒数
28             int minute=this.leftSecond/60;                   //得到分数
29             drawNumberBitmap(second,numberBitmaps,endX,endY,canvas, paint);//绘制秒数
30             int secondLength=(second+"").length()<=1 ? (second+"").length()+1 :
               (second+"").length();
31             int breakMarkX=endX-secondLength*numberWidth-breakMarkWidth;//秒数的位置
32             int breakMarkY=endY;
33             canvas.drawBitmap(breakMarkBitmap, breakMarkX, breakMarkY,paint);
                                                                //绘制时间分隔符图片
34             int miniteEndX=breakMarkX;                       //计算分钟图片的位置
35             int miniteEndY=endY;
36             drawNumberBitmap(minute,numberBitmaps,miniteEndX,miniteEndY,canvas,
               paint);                                          //绘制分钟
37        }
```

```
38        public void subtractTime(int second){                    //减少时间的方法
39             if(this.leftSecond>0){
40                  this.leftSecond-=second;
41             }else{                                               //如果时间为 0，结束游戏
42                  gameView.overGame();
43             }
44        }
45        public void drawNumberBitmap(
                       int number,Bitmap[] numberBitmaps,float endX,float endY,
                       Canvas canvas,Paint paint)
46        {                                                         //画数字图片的方法
47             String numberStr=number+"";                          //将数字转化成字符串
48             if(number<10){                                       //保证至少有两位
49                  numberStr="0"+numberStr;
50             }
51             for(int i=0;i<numberStr.length();i++){               //循环绘制每个字符
52                  char c=numberStr.charAt(i);                     //得到其中一个字符
53                  canvas.drawBitmap(
54                            numberBitmaps[c-'0'],                 //位图
55                            endX-numberWidth*(numberStr.length()-i),    //位置
56                            endY,
57                            paint                                 //画笔
58                  );
59   }}
```

- 代码第 26-37 行为绘制时间的方法。由于要绘制的时间为总秒数，因此，要将其换算成分钟数和秒数，再对分钟和秒钟以及中间的分隔符分别进行绘制。
- 代码第 38-44 行为倒计时的方法，循环调用此方法便可实现时间的倒计时，当时间为 0 时调用 GameView 中游戏结束的方法。

17.8.3 主菜单按钮 MainMenuButton 类的实现

MainMenuButton 类为主菜单中的虚拟按钮控件，该类只负责按钮按下和抬起时图片的切换，按钮负责的动作其实是在该类外适当的位置被执行的。MainMenuButton 类的代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 MainMenuButton.java。

```
1    package com.bn.d2.bill;                                        //声明包
2    import android.graphics.Bitmap;                                //引入相关类
3    import android.graphics.Canvas;                                //引入相关类
4    import android.graphics.Paint;public class VirtualButton {
5         float x;                                                  //按钮位置
6         float y;
7         int width;                                                //按钮宽度
8         int height;                                               //按钮高度
9         Bitmap downBmp;                                           //按钮位图
10        Bitmap upBmp;
11        boolean isDown=false;                                     //按钮是否被按下的标志
```

```
12        private int extendSize=20;                          //虚拟按钮触控面积扩大的尺寸
13        public VirtualButton(Bitmap downBmp,Bitmap upBmp,float x,float y){ //构造器
14             this.downBmp=downBmp;
15             this.upBmp=upBmp;
16             this.x=x+Constant.X_OFFSET;                    //将相对位置转换成实际位置
17             this.y=y+Constant.Y_OFFSET;
18             this.width=upBmp.getWidth();
19             this.height=upBmp.getHeight();
20        }
21        public void drawSelf(Canvas canvas,Paint paint){     //绘制虚拟按钮的方法
22             if(isDown){                                     //如果按钮被按下
23                  canvas.drawBitmap(downBmp, x, y, paint);
24             }
25             else{                                           //按钮没有被按下
26                  canvas.drawBitmap(upBmp, x, y, paint);
27             }
28        }
29        public void pressDown(){                             //按下按钮的方法
30             isDown=true;
31        }
32        public void releaseUp(){                             //放开按钮的方法
33             isDown=false;
34        }
35        public boolean isActionOnButton(float pressX,float pressY){
                                                               //判断按钮是否有触控事件的方法
36             return Constant.isPointInRect(pressX, pressY,
37                       x-extendSize, y-extendSize, width+2*extendSize, height+2*
                         extendSize);
38   }}
```

- 代码第 5-12 行为成员变量的定义。按钮的成员变量包括按钮图片、尺寸、位置以及按钮是否被按下的标志位等。
- 代码第 21-27 行为绘制按钮图片的方法，绘制按钮时，根据 isDown 标志位，当标志位为按下时，绘制按下的按钮图片，反之则绘制未被按下的图片。
- 代码第 35-38 行为判断按钮上是否有触控事件发生的方法，方法的参数为屏幕触控的位置，如果触控的位置在虚拟按钮的区域内则判断为按下，返回 true，反之返回 false。

说明　SoundSwitchButton 类也是游戏中的虚拟按钮类，其实现形式上大同小异，这里不再赘述，请读者自行查看随书光盘中的源代码。

17.8.4　获取日期的工具 DateUtil 类的实现

DateUtil 类中只有一个静态方法 getCurrentDate，其主要负责获取当前系统时间，并以特定格式的字符串的形式返回，其代码如下。

代码位置：见随书光盘中源代码/第 17 章/Billiard_2D/src/com/bn/d2/bill 目录下的 DateUtil.java。

```
1    package com.bn.d2.bill;                                          //声明包
2    import java.util.Date;                                           //引入相关类
3    public class DateUtil{
4         public static String getCurrentDate(){                      //得到当前日期的静态方法
5              String dateStr=null;                                   //定义返回的结果
6              Date date=new Date();                                  //创建日期对象
7              int year=date.getYear()+1900;                          //获得年份
8              int month=date.getMonth()+1;                           //获得月份
9              int dt=date.getDate();                                 //获得日期
10             dateStr=year+"-"+((month<10)?"0"+
11                       month:month)+"-"+((dt<10)?"0"+dt:dt);//将日期组织成特定格式的字符串
12             return dateStr;                                        //返回字符串
13   }}
```

提示

该方法非常简单，但又很实用，在需要获得系统时间的地方直接调用该方法即可获得特定格式的字符串，之后再将字符串解析成所需的格式并显示到游戏界面中即可。由于篇幅所限，在此只介绍了一部分自定义控件及工具类，其他类的实现请读者自行查看随书光盘中的源代码。

17.9 游戏的优化与改进

到此为止，本游戏的功能开发完毕，但是还有很多地方值得优化与改进。下面提出几点可以改进的地方，有兴趣的读者可以对本游戏进行必要的完善。

- 加入桌球的游戏规则

桌球的玩法多种多样，有很多的游戏规则，本游戏只简单地实现了最基本的规则，没有实现多种玩法，有兴趣的读者可以把复杂的游戏规则加入代码中，增强游戏的可玩性。

- 去除游戏中的 Bug

本游戏的球与球碰撞时，偶而会出现两球的粘滞现象，读者可深入研究碰撞部分的代码，将其中逻辑理清，去除球与球碰撞时的粘滞现象。

- 排行榜界面的优化

在游戏排行榜界面中实现了得分的上下翻页，但没有显示当前页的位置。有兴趣的读者可以在排行榜界面的适当位置加入页码或进度条，使其更加人性化。

第 18 章　益智游戏——3D 版推箱子

随着移动互联网时代的到来，移动手持设备功能日益强大，从而使曾经只能在个人电脑上运行的 3D 游戏，逐渐移植到了日常使用的手机上。本章通过一款经典益智游戏——3D 版推箱子游戏在 Android 平台上的设计与实现，使读者掌握此类游戏的开发全过程。

18.1 3D 版推箱子的背景及功能概述

本节将对 3D 游戏开发进行简单介绍，通过本节的学习，读者可以对 3D 版推箱子游戏有初步的认识，并了解本章开发案例游戏效果以及具体功能。

18.1.1　背景概述

推箱子是一款来自日本的古老游戏，目的是训练逻辑思考能力。要求在一个狭小的空间把木箱放在指定的位置，稍不小心就会出现箱子无法移动或者通道被堵的情况。

推箱子游戏是一款老少皆宜的益智类游戏，虽然游戏画面与玩法简单，但是趣味无穷。

18.1.2　功能简介

本节将对本游戏的玩法、游戏的功能和运行效果进行简单介绍，使读者了解游戏的功能和控制方式，其具体的操作方法如下。

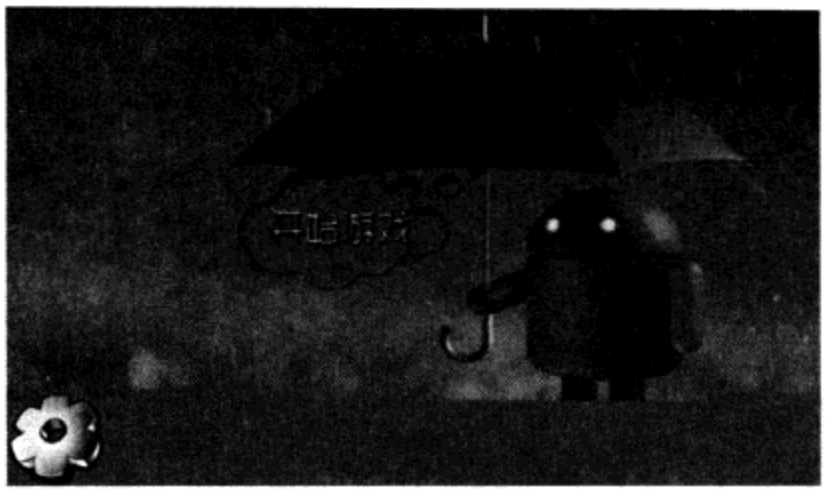
（1）安装游戏后，打开游戏，进入游戏的欢迎动画界面，如图 18-1 和图 18-2 所示。

▲图 18-1　欢迎动画界面 1

▲图 18-2　欢迎动画界面 2

（2）欢迎动画界面播放完毕后，游戏自动进入主菜单界面，如图 18-3 所示。

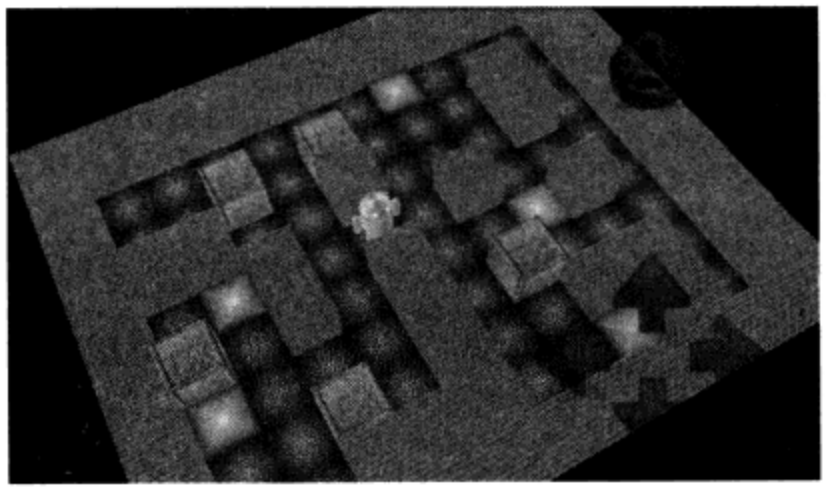
（3）单击主菜单中左下角的“设置”按钮，进入设置音效界面，如图 18-4 所示。



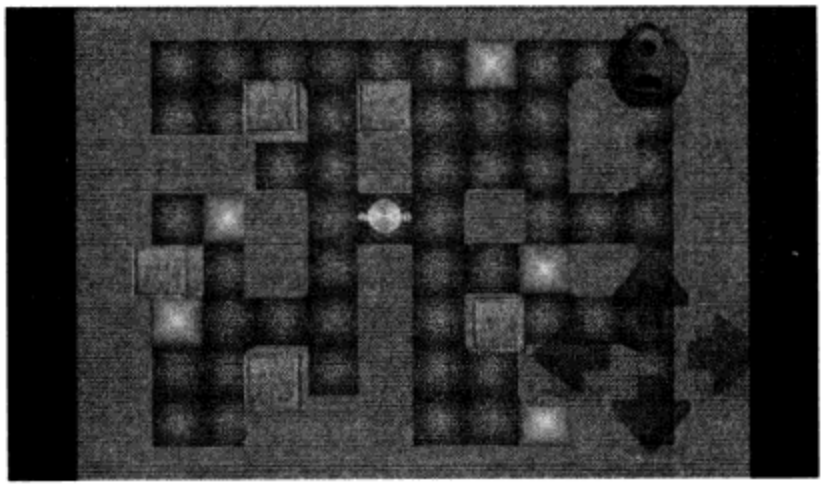
▲图 18-3　主菜单界面

▲图 18-4　设置音效界面

（4）按下“返回”按钮，回到游戏主菜单界面。单击主菜单中“开始游戏”按钮，进入游戏界面，如图 18-5 和图 18-6 所示。

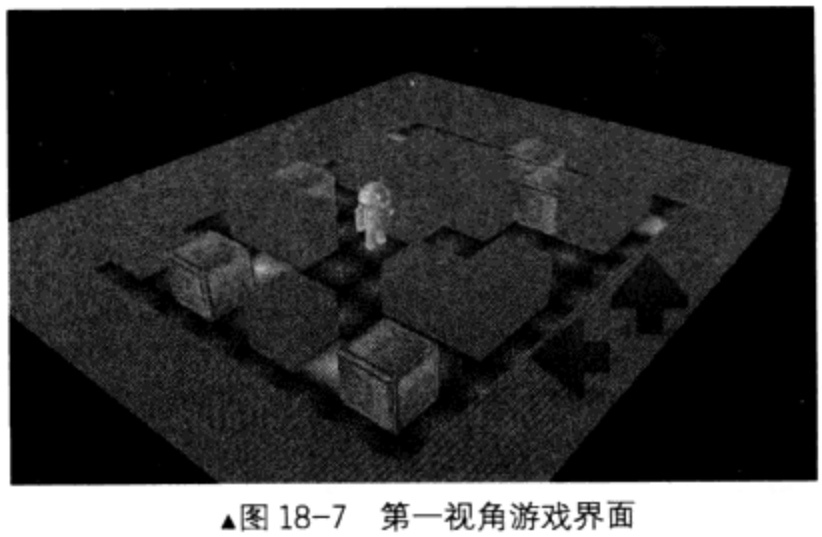


▲图 18-5　游戏开始界面 1



▲图 18-6　游戏开始界面 2

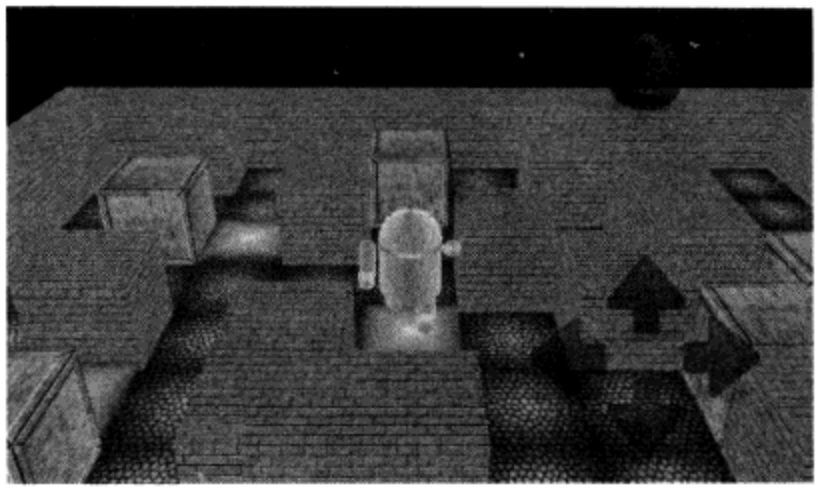
（5）按下游戏界面右上角的“切换视角”按钮，玩家可以实现游戏界面第一视角和第三视角的切换功能，如图 18-7 和图 18-8 所示。



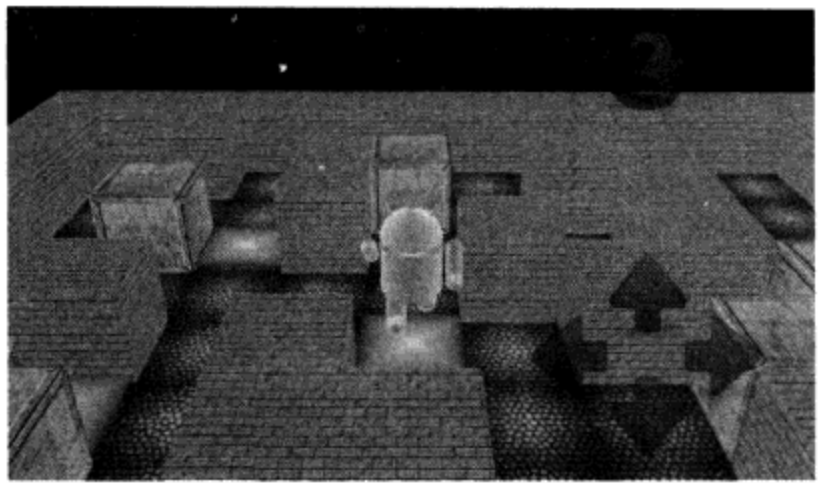
▲图 18-7　第一视角游戏界面

▲图 18-8　第三视角游戏界面

（6）按下游戏界面“前进”虚拟按钮时，游戏界面中 Android 小机器人执行前进的走路动画，如图 18-9 和图 18-10 所示。

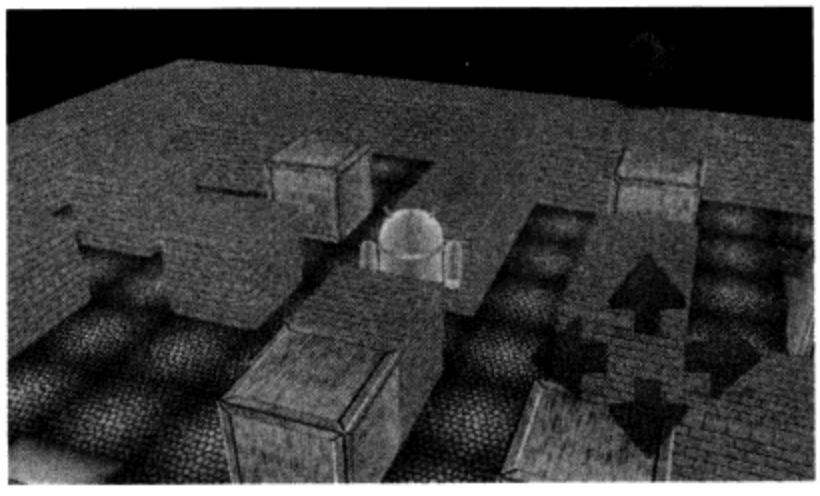


▲图 18-9　机器人前进动画 1

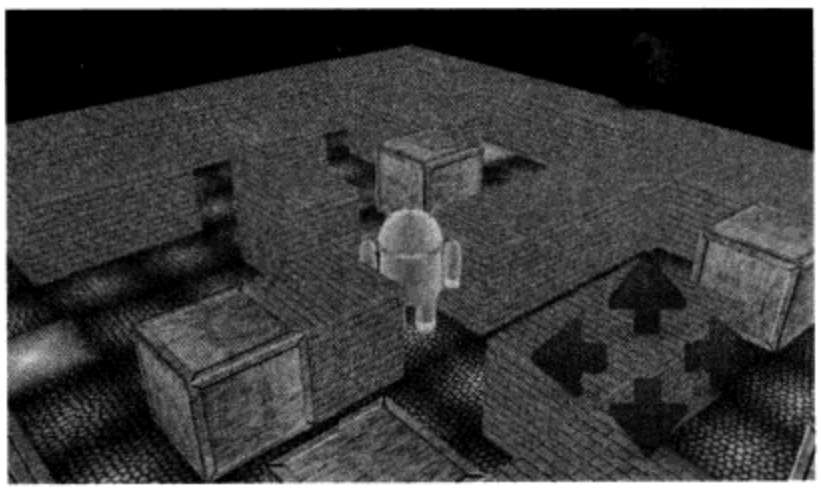


▲图 18-10　机器人前进动画 2

（7）按下游戏界面“左转”虚拟按钮时，游戏界面中 Android 小机器人执行左转身动画，如图 18-11 和图 18-12 所示。

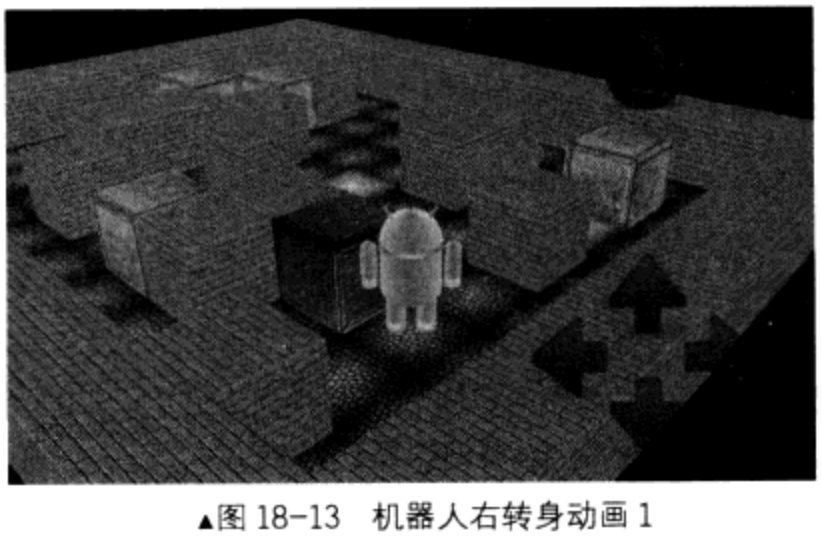


▲图 18-11　机器人左转身动画 1

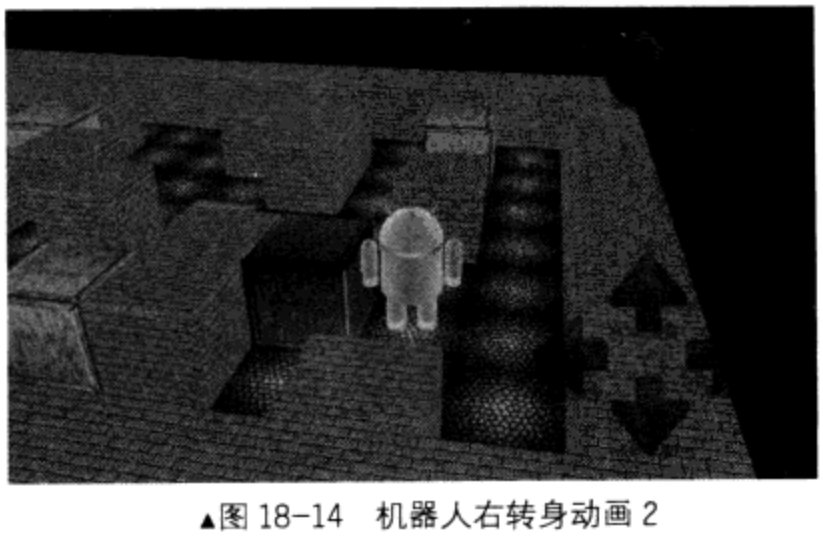


▲图 18-12　机器人左转身动画 2

（8）按下游戏界面“右转”虚拟按钮时，游戏界面中 Android 小机器人执行右转身动画，如图 18-13 和图 18-14 所示。

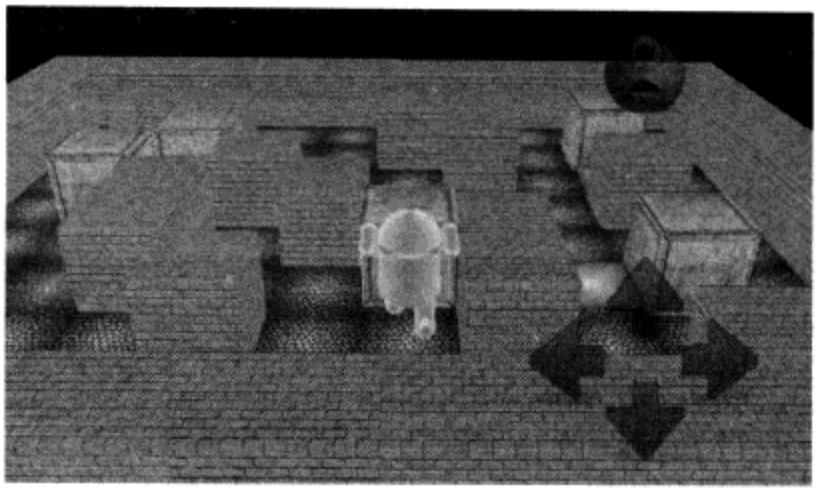


▲图 18-13　机器人右转身动画 1

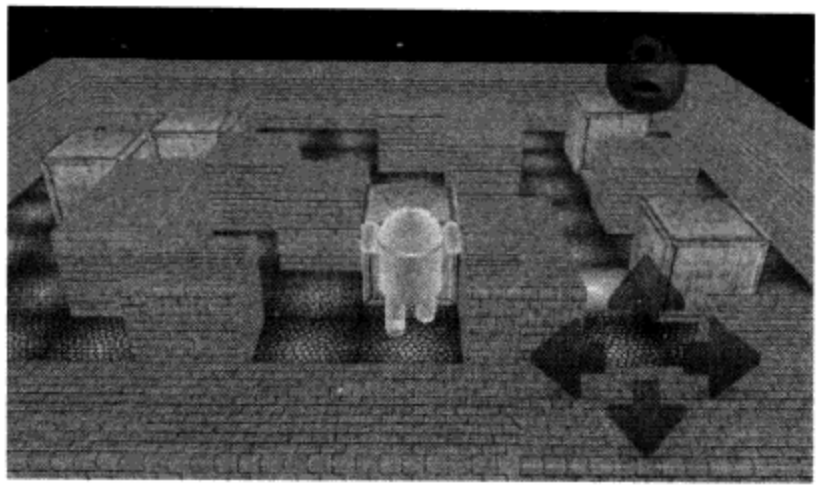


▲图 18-14　机器人右转身动画 2

（9）当机器人遇到可推动的箱子时，按下游戏界面中“前进”虚拟按钮，游戏界面中机器人执行推箱子前进动画，如图 18-15 和图 18-16 所示。

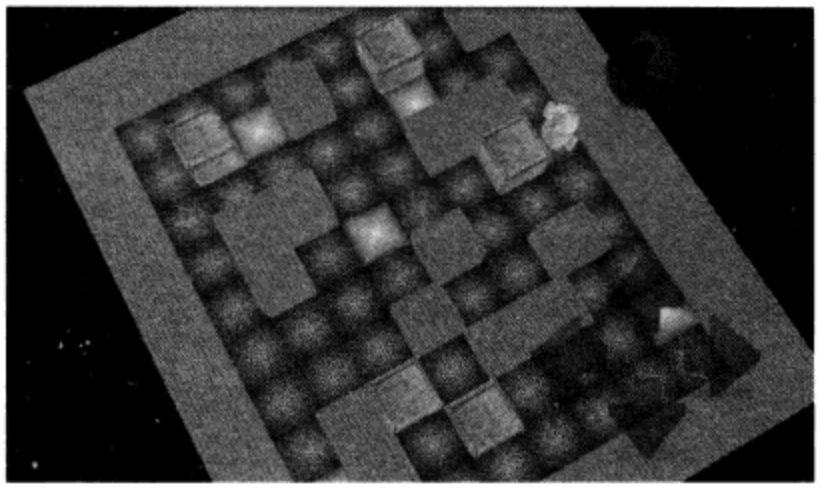


▲图 18-15　机器人推箱子前进动画 1

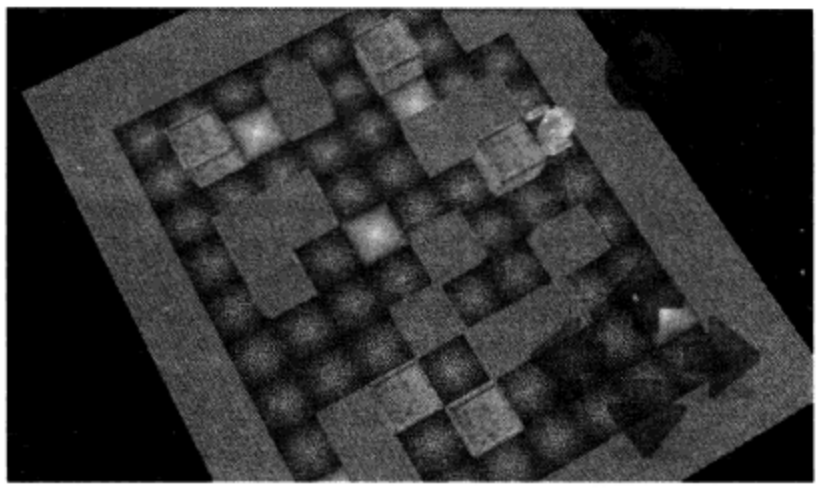


▲图 18-16　机器人推箱子前进动画 2

（10）当界面为第一视角显示时，机器人遇到可推动的箱子，按下游戏界面“前进”虚拟按钮，游戏界面中机器人执行推箱子前进动画，如图 18-17、图 18-18 和图 18-19 所示。

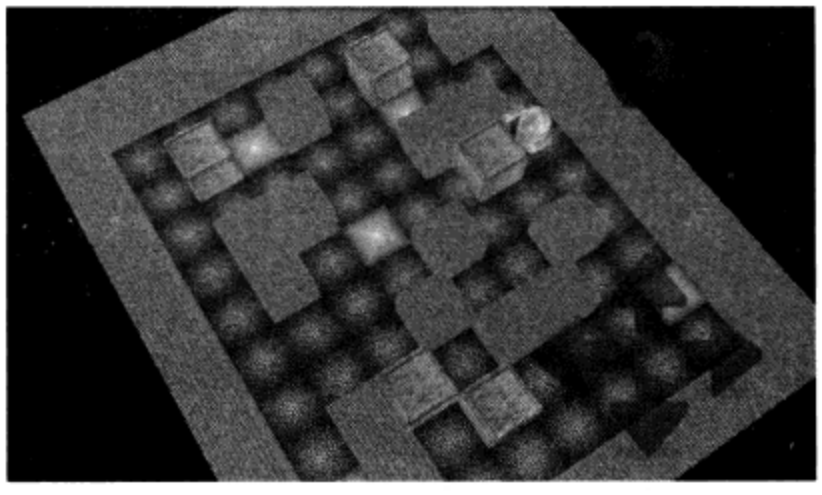


▲图 18-17　第一视角机器人推箱子前进动画 1



▲图 18-18　第一视角机器人推箱子前进动画 2

（11）本游戏设定为两关，当玩家通关之后，游戏进入通关界面，如图 18-20 所示。



▲图 18-19　第一视角机器人推箱子动画 3

▲图 18-20　游戏通关界面

提示

以上效果图，大部分是在第一视角下的效果。机器人的后转身动画效果与机器人的左转身、右转身基本相同，由于篇幅所限，这里不再赘述。

18.2 游戏的策划及准备工作

上一节介绍了本游戏的背景及功能，本节将要介绍游戏的策划以及开发前的准备工作，这些工作虽然略显枯燥，但是在游戏的开发过程中却起到相当大的作用。

18.2.1 游戏的策划

本小节将对该游戏的策划进行简单介绍。在真实的游戏开发中，对游戏的策划还需要更加细致、全面的准备，在此只列出其中一部分。

- 游戏情节

本游戏属于休闲益智类游戏，情景设置非常简单，主要有机器人、木箱、目标、墙体和地板等。

- 运行的目标平台

本游戏的运行目标平台为 Android 2.1、Android 2.2 或者是更高的 Android 版本。

- 操作方式

本游戏通过触摸屏幕进行操作，玩家可触控屏幕的虚拟按钮完成机器人的前进、推箱子前进、左转弯、右转弯等动作，触控右上角的“视角切换”按钮来完成界面视角的切换。

- 呈现技术

本游戏界面采用的是 OpenGL ES 技术完成 3D 游戏的开发。游戏在第三视角的下可随意拖动屏幕变换视角，并可以在第一视角和第三视角之间切换。

- 音效设置

为了增强游戏的用户体验，本游戏添加了背景音乐、机器人推箱子的音效等。

18.2.2 Android 平台下游戏开发的准备工作

游戏开发前需要做一些必不可少的准备工作，包括游戏中用到的图片、声音素材等。本案例所用到的部分图片资源，如表 18-1 所列。

表 18-1　　图片清单

图片名	大小（KB）	像素（w×h）	用途	图片名	大小（KB）	像素（w×h）	用途
b_down.png	7.46	128×128	后转身按钮	sound2.png	67.1	500×350	关闭音乐按钮
b_left.png	9.02	128×128	左转身按钮	sound3.png	60.6	500×267	开启音效按钮
b_right.png	8.69	128×128	右转身按钮	sound4.png	60.7	500×267	关闭音效按钮
b_up.png	8.09	128×128	前进按钮	start_buton1.png	59.5	500×300	开始游戏按钮

续表

图片名	大小(KB)	像素（w×h）	用途	图片名	大小（KB）	像素（w×h）	用途
dukea.png	37.1	480×320	欢迎背景 1	start_buton1.png	54.1	500×300	开始游戏按钮
dukeb.png	19.7	480×320	欢迎背景 2	target.png	19.5	128×128	目标
floor.jpg	40.5	128×128	地板	view.png	26.2	128×128	切换视角按钮
gameover.jpg	138	480×320	通关界面	wall.png	30.6	128×128	墙
main_background.png	128	480×320	主菜单界面背景	wood_bin0.png	33.4	128×128	普通木箱
robot_head.jpg	72.7	512×512	机器人头纹理	wood_bin1.png	33.9	128×128	红色木箱
robot_texid.png	108	512×512	机器人身体纹理	sound1.png	61.9	500×350	开启音乐按钮
set.png	15	128×128	设置按钮				

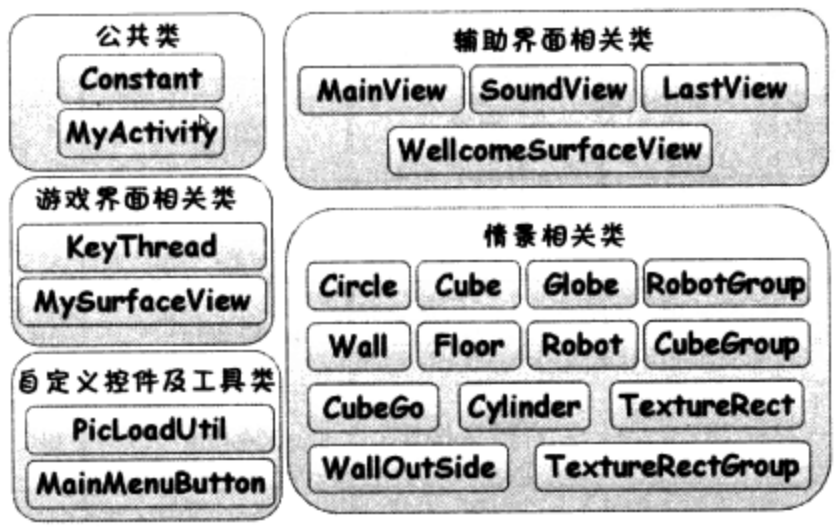
说明 表 18-1 列出了本案例中用到的部分位图资源详细信息，这些位图资源都存储在 res/drawable-nodpi 文件夹下。

18.3 游戏的架构

本节将对 3D 推箱子游戏的整体架构进行介绍，让读者了解该休闲益智游戏的整体框架，这样读者将更容易理解后面将要讲到的内容，达到事半功倍的效果。

18.3.1 游戏的框架简介

本小节先对该游戏的框架进行介绍，使读者对游戏的架构有个整体的把握，以方便对后面每个类的学习。游戏类框架如图 18-21 所示。



▲图 18-21 3D 版推箱子游戏框架图

说明	游戏中的类分为公共类、辅助界面相关类、游戏界面相关类、情景相关类和自定义控件及工具类，各个模块之相互关联、相互配合。

18.3.2　各个类的简要介绍

1. 公共类

- Activity 的实现类 MyActivity

该类继承并扩展了 Activity 类，是整个游戏程序的入口。

- 常量类 Constant

该类封装了游戏中用到的全部常量，便于游戏的维护和管理。

2. 辅助界面相关类

- 欢迎界面 WellcomeSurfaceView

该类实现游戏的欢迎界面，主要负责欢迎界面的绘制工作。

- 主菜单界面 MainView

该类为游戏主菜单界面类，主要负责主菜单中虚拟按钮的绘制及监听事件等工作。

- 音效控制界面 SoundView

该类为游戏中音效控制界面类，其主要负责游戏中背景音乐和音效的开关。

- 胜利界面 LastView

该类为玩家通关游戏后进入的界面，当玩家按下“返回”按钮后，会跳转到主菜单界面。

3. 游戏界面相关类

- 游戏主界面类 MySurfaceView

该类为游戏的主界面类，主要负责游戏中各个对象的创建、游戏中各个实体类的绘制等工作。

- 虚拟按钮触控监听线程类 KeyThread

该类为游戏界面虚拟按钮触控监听线程类，主要负责判断“前进”虚拟按钮是否被按下。

4. 情景相关类

- 圆类 Circle

该类为绘制 3D 圆面的封装类，是绘制英雄机器人头底面部分不可缺少的组件。

- 立方体类 Cube

该类为绘制 3D 立方体的封装类，游戏中的普通木箱和红色木箱就是该类的对象。

- 立方体运动类 CubeGo

该类为立方体木箱运动动画的封装类，当机器人遇可以推动的木箱时，执行木箱运动动画。

- 立方体组类 CubeGroup

该类为立方体木箱组的封装类，负责在游戏界面被创建时扫描地图数组，并且在相应位置绘制

木箱。

- 圆柱类 AirCraft

该类为圆柱体的封装类，机器人的天线、手臂和身体等就是该类的对象。

- 地板类 Floor

该类为游戏界面中地板的封装类，负责根据地图数组的大小绘制相应的地板。

- 半球类 Globe

该类为游戏界面中半球的封装类，负责绘制机器人头的上半部分、手臂的两端等。

- 机器人类 Robot

该类为游戏界面中机器人的封装类，负责利用前面介绍的圆类、半球类等拼接成一个机器人。

- 机器人组类 RobotGroup

该类为机器人组类的封装类，负责扫面地图数组，并且在相应位置绘制机器人。

- 矩形纹理类 TextureRect

该类为游戏界面中矩形纹理的封装类，负责绘制游戏中木箱每一面的矩形纹理和绘制红色目标。

- 矩形纹理组类 TextureRectGroup

该类为游戏界面中矩形纹理组的封装类，负责扫描地图数组，并且在相应位置绘制红色目标。

- 墙体类 Wall

该类为游戏界面中墙体的封装类，负责扫描地图数组，并且在相应位置绘制上部和内部墙体。

- 外围墙体类 WallOutSide

该类为游戏界面中外围墙体的封装类，负责根据地图数组绘制游戏界面中的外围墙体。

5. 自定义控件及工具类

- 主菜单按钮 MainMenuButton

该类为主菜单界面中按钮的封装类，主菜单的“开始”按钮就是该类的对象。

- 图片缩放的工具类 PicLoadUtil

该类为封装好的缩放图片的工具类。调用其相应的静态工具方法即可对图片进行不等比例的缩放。

18.4 公共类的实现

从本节开始将正式介绍游戏代码的开发。为了使代码的结构更加清晰，也为了使读者了解游戏的整个开发过程，本节将尽量按原开发顺序对各个类一一进行讲解。

18.4.1 主控制类——MyActivity 的代码框架

首先介绍的是 Activity 的实现。Activity 的作用是对各个界面进行管理、切换以及对欢迎界面、主界面中线程发送来的请求做出响应，其代码框架如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/ com/bn/pb/目录下的 MyActivity.java。

```
1    package com.bn.pb;                                          //引入的包名
2    import java.util.HashMap;                                   //相关包的引入
3    ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4    import android.view.WindowManager;                          //相关包的引入
5    enum WhichView {WELLCOM_VIEW,MAIN_VIEW,GAME_VIEW,SOUND_VIEW,LAST_VIEW}
6    public class MyActivity extends Activity {
7         WhichView curr;                                        //枚举类的引用
8         WellcomeSurfaceView wellcomView;                       //欢迎界面
9         MySurfaceView gameView;                                //进入游戏界面
10        MainView mainView;                                     //主菜单界面
11        SoundView soundView;                                   //背景音乐控制界面
12        LastView lastView;                                     //最后的界面
13        boolean soundFlag=true;                                //背景音乐播放标志位
14        boolean soundPoolFlag=true;                            //声音池的标志位
15        SoundPool soundPool;                                   //声音池
16        HashMap<Integer, Integer> soundPoolMap;                //声音池中声音ID与自定义声音ID的Map
17        Handler myHandler = new Handler(){};
18      @Override
19      public void onCreate(Bundle savedInstanceState) {    //重写 onCreate 方法
20        super.onCreate(savedInstanceState);                //实现父类方法的功能
21        setVolumeControlStream(AudioManager.STREAM_MUSIC); //游戏过程中允许调整多媒体音量
22        this.requestWindowFeature(Window.FEATURE_NO_TITLE);      //去掉标题
23          getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
24              WindowManager.LayoutParams.FLAG_FULLSCREEN);       //去掉标头
25          this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                                                                   //强制横屏
26          DisplayMetrics dm=new DisplayMetrics();                //获取分辨率
27          getWindowManager().getDefaultDisplay().getMetrics(dm);
28          if(dm.widthPixels>dm.heightPixels) {               //若宽度大于高度
29              Constant.SCREEN_WIDTH=dm.widthPixels;          //记录屏幕宽度
30              Constant.SCREEN_HEIGHT=dm.heightPixels;        //记录屏幕高度
31          }else{//若宽度小于等于高度
32              Constant.SCREEN_HEIGHT=dm.widthPixels;         //记录屏幕宽度
33              Constant.SCREEN_WIDTH=dm.heightPixels;         //记录屏幕高度
34          }
35          goToWellcomView();                                 //进入欢迎界面
36          initSound();                                       //初始化声音
37      }
38        @Override
39        public boolean onKeyDown(int keyCode,KeyEvent e){}  //重写 onKeyDown 方法
40      @Override
41        protected void onResume() {super.onResume();}       //重写 onResume 方法
42        @Override
43        protected void onPause() {super.onPause();}         //重写 onPause 方法
44        public void initSound(){}                           //加载声音资源的方法
45      public void playSound(int sound, int loop) {          //播放声音的方法
46      public void sendMessage(int what){}                   //向 Handler 发送信息的方法
47        public void goToWellcomView(){}                     //进入"百纳科技"界面
48        public void goToGameView(){}                        //进入"游戏界面"
49        public void goToMainView(){}                        //进入 "选择主界面"
50        public void goToSoundView(){}                       //进入背景音乐控制界面
51        public void goToLastView(){}                        //进入游戏结束界面
52   }
```

- 代码第 5 行是安全类型枚举，方便主菜单界面、游戏主界面、声音管理界面和通关界面的跳转。
- 代码第 7-17 行分别为枚举类的引用、声音池的引用、游戏主界面和各个辅助界面类的引用，

代码 17 定义了 Handler 的对象用于线程之间传递消息，方便页面跳转。

- 代码第 18-37 行分别表示的是游戏过程中允许调节多媒体音量、强制横屏、去掉标题、获取屏幕分辨率，跳转到欢迎界面和声音资源的加载等。
- 代码第 38-52 行分别表示的是自己定义的加载声音的方法、播放声音的方法、以及跳转到欢迎界面、游戏主界面、菜单主界面、背景音乐控制界面和通关界面的方法。

说明 以上代码只是 MyActiviy 类的代码框架，其中大部分方法和成员变量都是空实现，不过读者不用担心，这些方法在下一小节将一一实现。

18.4.2 MyActivity 类成员方法的实现

在前面小节中介绍了 MyActivity 类的代码框架，本小节将会对其中主要成员变量及方法的开发进行详细介绍，其步骤如下。

（1）onKeyDown 方法的实现。onKeyDown 方法主要负责监听在游戏中按下返回键转到相应界面。其代码如下。将下列代码插入到 MyActivity 类的第 39 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 MyActivity.java。

```
1    @Override
2    public boolean onKeyDown(int keyCode,KeyEvent e){
3         if(keyCode==4){                                  //按下返回键
4              switch(curr) {                              //判断是哪一个界面
5               case GAME_VIEW:                            //如果游戏进行中，返回到主界面
6                    gameView.overGame();                  //调用自定义的 overGame()方法
7               case SOUND_VIEW:                           //得分界面，跳转到主菜单界面
8               case LAST_VIEW:                            //最后的界面，跳转到主菜单界面
9                    goToMainView();                       //调用 goToMainView()方法
10                   break;
11              case WELLCOM_VIEW:                         //欢迎界面，退出游戏
12              case MAIN_VIEW:                            //主控制界面，退出游戏
13                   System.exit(0);                       //退出游戏
14                   break;
15                  }
16             return true;                                //返回 true
17        }
18        return false;                                    //返回 false
19   }
```

提示 onKeyDown 方法是系统自定义的方法，在这里重写了此方法，用于各个界面的跳转，curr 是安全类型枚举的对象，用于标记当前是哪一个界面，方便界面跳转。

（2）接收各个界面传递来消息的类的实现，将下列代码插入到 MyActivity 类的第 17 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 MyActivity.java。

```
1    Handler myHandler = new Handler(){          //处理各个 SurfaceView 发送的消息
```

```
2       public void handleMessage(Message msg) {  //接收消息
3          switch(msg.what){
4          case 0:                                //传递的消息为 0
5               Constant.COUNT=0;                 //游戏关数归 0
6               goToMainView();                   //进入“主菜单界面”
7               break;
8          case 1:                                //传递的消息为 1
9               goToGameView();                   //进入"游戏界面"
10              break;
11         case 2:                                //传递的消息为 2
12              goToSoundView();                  //进入背景音乐控制界面
13              break;
14         case 3:                                //传递的消息为 3
15              Constant.COUNT=0;                 //游戏关数清零
16              goToLastView();                   //进入最后的通关界面
17              break;
18         }}};
```

提示　myHandler 通过接收各个界面中 sendMessage 方法传递来的消息进行页面跳转，传递的消息为 0、1、2、3，则游戏界面分别跳转到主菜单界面、游戏界面、声音控制界面和通关界面。

（3）goToMainView()、goToWellcomeView()以及 goToSoundView()等方法的实现，goToMainView()方法的代码如下。将下列代码插入到 MyActivity 类的第 66 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 MyActivity.java。

```
1    public void goToMainView(){                          //进入“主菜单界面”
2         if(mainView==null){                             //如果 mainView 为 null
3              mainView=new MainView(this);               //创建对象
4    }
5    setContentView(mainView);                            //跳转到主菜单界面
6    mainView.requestFocus();                             //获取焦点
7    mainView.setFocusableInTouchMode(true);              //设为可触控
8    curr=WhichView.MAIN_VIEW;                            //安全类型枚举的引用赋值
9    }
```

提示　goToMainView()等方法用于各个界面的跳转，写成一个方法便于管理。其他方法和此方法非常类似，由于本书篇幅有限，这里不再赘述，读者可自行查阅随书光盘中源代码。

18.4.3　常量类 Constant 的实现

Constant 常量类的作用是将代码中用到的常量全部集中起来，封装成一个类。这样不仅方便游戏的开发，而且便于开发完成后的调试等工作，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Constant.java。

```
1    package com.bn.pb;                                    //包声明
2    public class Constant {
3        public static final int[][][] MAP={{//1 表示墙，2 表示地，3 表示目标，4 表示箱子，5 表示人
4                {1,1,1,1,1,1,1,1,1,1,1,1},{1,2,2,2,2,2,2,3,2,2,2,1},{1,2,2,4,2,4,
                 2,2,2,1,2,1},
5                {1,1,1,2,2,1,2,2,2,1,2,1},{1,2,3,1,2,5,2,1,2,2,2,1},{1,4,2,1,2,1,
                 2,2,3,1,2,1},
6                {1,3,2,2,2,1,2,4,2,2,2,1},{1,2,2,4,2,1,2,2,1,1,2,1},{1,2,2,1,1,1,
                 2,2,3,1,2,1},
7                {1,1,1,1,1,1,1,1,1,1,1,1}                    //第一关地图数组
8            },{
9                {1,1,1,1,1,1,1,1,1,1,1,1},{1,2,1,2,2,2,2,2,2,3,2,1},{1,2,2,2,1,4,
                 2,1,2,2,2,1},
10               {1,2,4,3,1,2,2,2,1,2,2,1},{1,2,2,2,2,2,1,5,1,2,2,1},{1,2,1,2,2,3,
                 2,1,2,4,2,1},
11               {1,2,3,2,1,2,2,2,4,2,2,1},{1,2,4,2,1,1,2,2,1,1,2,1},{1,2,2,2,2,2,
                 2,2,2,3,2,1},
12               {1,1,1,1,1,1,1,1,1,1,1,1}                    //第二关地图数组
13           }};
14       public static int COUNT=0;                           //当前为第几关
15       public static int SCREEN_WIDTH;                      //屏幕的宽
16       public static int SCREEN_HEIGHT;                     //屏幕的高
17       public static final float WALL_HEIGHT=1.0f;          //墙的高度
18       public static final float UNIT_SIZE=1.0f;            //地面每个格子的大小
19       public static final float CUBE_SIZE=0.5f;            //立方体木箱的半边长
20       public static final float FLOOR_Y=0f;                //地面的 Y 坐标
21       public static final float TOUCH_SCALE_FACTOR=180.0f/480f;    //角度缩放
22       public static final float DISTANCE=14f;              //摄像机与目标点的距离
23       public static final float ANTENNA_RADIUS=0.1F/5;     //天线半径
24       public static final float ANTENNA_LENGTH=1.0f/5;     //天线长度
25       public static final float HEAD_RADIUS=1.5F/5;        //头半径
26       public static final float ARM_RADIUS=0.4F/5;         //胳膊半径
27       public static final float ARM_LENGTH=1.8F/5;         //胳膊的长度
28       public static final float BODY_RADIUS=1.5F/5;        //身体半径
29       public static final float BODY_LENGTH=2.5F/5;        //身体长度
30       public static final float LEG_RADIUS=0.5F/5;         //腿半径
31       public static final float LEG_LENGTH=1.4f/5;         //腿长度
32       public static final float HEAD_Y=3.0f/5;             //头 Y 坐标
33       public static final float BODY_Y=1.5f/5;             //身体 Y 坐标
34       public static final float ARM_Y=2.5f/5;              //胳膊 Y 坐标
35       public static final float LEG_Y=-0.5F/5;             //腿 Y 坐标
36       public static final float upYMax=0.666f;             //上虚拟按钮的有效触控范围
37       public static final float upYMin=0.519f;             //y 的最小值
38       public static final float upXMax=0.871f;             //x 的最大值
39       public static final float upXMin=0.773f;             //x 的最小值
40       public static final float downYMax=0.95f;            //下虚拟按钮的有效触控范围
41       public static final float downYMin=0.806f;           //y 的最小值
42       public static final float downXMax=0.871f;           //x 的最大值
```

```
43        public static final float downXMin=0.773f;          //x 的最小值
44        public static final float leftYMax=0.806f;          //左虚拟按钮的有效触控范围
45        public static final float leftYMin=0.666f;          //y 的最小值
46        public static final float leftXMax=0.772f;          //x 的最大值
47        public static final float leftXMin=0.666f;          //x 的最小值
48        public static final float rightYMax=0.806f;         //右虚拟按钮的有效触控范围
49        public static final float rightYMin=0.666f;         //y 的最小值
50        public static final float rightXMax=0.977f;         //x 的最大值
51        public static final float rightXMin=0.872f;         //x 的最小值
52        public static final float viewXMax=0.877f;          //视角变换虚拟按钮
53        public static final float viewXMin=0.771f;          //y 的最小值
54        public static final float viewYMax=0.194f;          //x 的最大值
55        public static final float viewYMin=0.041f;          //x 的最小值
56    }
```

- 代码第 3-22 行分别为第一关和第二关的游戏地图数组，其中 1、2、3、4、5 分别代表墙、地面、目标矩形、箱子和人，14-22 行分别表示当前为第几关，箱子的高度、墙体的高度等。
- 代码第 23-35 行为机器人的信息，分别为机器人天线的半径及长度、头的半径、胳膊的半径及长度、腿的半径及长度和头、身体、腿的 *Y* 坐标等。
- 代码第 36-56 行为“前进”，“左转”，“右转”，“后转”和“视角变换”按钮的触控范围，以按钮占屏幕比例，来确定按钮的位置和触控范围，便于管理和修改。

18.5 辅助界面相关类的实现

上一节介绍了公共类的实现，本节开始介绍辅助界面相关类的实现。辅助界面相关类指，除游戏界面之外其他辅助界面，包括欢迎动画界面、主菜单界面、音效设置界面和游戏通关界面。

18.5.1 欢迎动画界面 WelcomeView 类的实现

本游戏欢迎动画界面的实现比较简单，主要是靠单独开启的线程来不断地调节画笔的透明度，并不断地用该画笔绘制图片来实现的。其代码如下

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 WelcomView.java。

```
1     package com.bn.fkdp;                                        //声名包
2     import android.graphics.Bitmap;                             //引入相关类
3     ……//此处省略了部分类的引入代码，读者可自行查看随书光盘的源代码
4     import android.view.SurfaceView;                            //引入相关类
5     public class WellcomeSurfaceView extends SurfaceView
6     implements SurfaceHolder.Callback  {                        //实现生命周期回调接口
7          GameActivity activity;
8          Paint paint;                                           //画笔
9          int currentAlpha=0;                                    //当前的不透明值
10         int screenWidth=Constant.SCREEN_WIDTH;                 //屏幕宽度
11         int screenHeight=Constant.SCREEN_HEIGHT;               //屏幕高度
12         int sleepSpan=50;                                      //动画的时延 ms
13         Bitmap[] logos=new Bitmap[2];                          //Logo 图片数组
14         Bitmap currentLogo;                                    //当前 Logo 图片引用
15         int currentX;                                          //当前位置
```

```
16          int currentY;
17          public WellcomeSurfaceView(GameActivity activity) { //构造器
18               super(activity);
19               this.activity = activity;
20               this.getHolder().addCallback(this);          //设置生命周期回调接口的实现者
21               paint = new Paint();                         //创建画笔
22               paint.setAntiAlias(true);                    //打开抗锯齿
23               logos[0]=BitmapFactory.decodeResource(activity.getResources(), R.drawable.
                 dukea);
24               logos[1]=BitmapFactory.decodeResource(activity.getResources(), R.drawable.
                 dukeb);
25          }
26          public void onDraw(Canvas canvas){                //重写 onDraw 方法
27               paint.setColor(Color.BLACK);                 //设置画笔颜色
28               paint.setAlpha(255);                         //设置画笔透明度
29               canvas.drawRect(0, 0, screenWidth, screenHeight, paint);
                                                              //绘制黑填充矩形清背景
30               if(currentLogo==null)return;
31               paint.setAlpha(currentAlpha);                //设置画笔透明度
32               canvas.drawBitmap(currentLogo, currentX, currentY, paint);//进行平面贴图
33          }
34          public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {}
35          public void surfaceCreated(SurfaceHolder holder) {   //创建时被调用
36               new Thread(){                                   //创建线程对象
37                    public void run(){                         //实现 run 方法
38                         for(Bitmap bm:logos){                 //循环两张欢迎图片
39                              currentLogo=bm;                  //获得当前显示图片
40                              currentX=screenWidth/2-bm.getWidth()/2  //计算图片位置
41                              currentY=screenHeight/2-bm.getHeight()/2;
42                              for(int i=255;i>-10;i=i-10) {
                                                       //动态更改图片的透明度值并不断重绘
43                                   currentAlpha=i;   //获得当前透明度
44                                   if(currentAlpha<0){        //保证透明度大于 0
45                                        currentAlpha=0;
46                                   }
47                                   SurfaceHolder  myholder=WellcomeSurfaceView.this.
                                     getHolder();
48                                   Canvas canvas = myholder.lockCanvas(); //获取画布
49                                   try{
50                                        synchronized(myholder){       //锁定 myholder
51                                             onDraw(canvas);          //绘制
52                                        }}
53                                   catch(Exception e){
54                                        e.printStackTrace();}//打印异常
55                                   finally{
56                                        if(canvas != null){  //如果画布引用不为空
57                                             myholder.unlockCanvasAndPost(canvas);
                                                              //释放画布
58                                        }}
59                                   try{
60                                        if(i==255){          //若是新图片，多等待一会
61                                             Thread.sleep(1000);}
62                                        Thread.sleep(sleepSpan);}//使线程睡眠一段时间
63                                   catch(Exception e){
64                                        e.printStackTrace(); //打印异常
65                                   }}}
66                         activity.sendMessage(0);             //切换到主菜单界面
67                    }
68               }.start();                                     //启动线程
69          }
70          public void surfaceDestroyed(SurfaceHolder arg0) {} //销毁时被调用
71     }
```

- 代码第 1-16 行为包的声明、相关类的引入、欢迎界面动画自适应屏幕用到的屏幕分辨率 screenWidth 和 screenHeight 的赋值以及动画的延迟时间 sleepSpan 的赋值等。
- 代码第 26-33 行为 onDraw 方法的实现。onDraw 方法主要负责用成员变量 paint 来绘制图片。通过不断地改变两张位图的透明度，从而形成图片由亮变暗的效果。
- 代码第 36-71 行创建并启动了一个线程，负责动态改变成员变量 paint 和 currentLogo 的值，其中第 66 行，当动画播放完后向 Activity 发送了一个切换到主菜单界面的消息。

18.5.2　主菜单界面 MainView 的实现

上一小节中讲到动画播放完毕后，Activity 会将界面转到主菜单界面，本小节就将介绍主菜单界面的开发与实现，主菜单界面是游戏的主控制界面，采用的是 2D 贴图技术，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 MainView.java。

```
1    package com.bn.pb;                                          //包声明
2    import android.graphics.Bitmap;                             //相关类的引入
3    ……//此处省略了部分类引入的代码，读者可自行查看随书光盘的源代码
4    import android.view.SurfaceView;                            //相关类的引入
5    public class MainView extends SurfaceView                   //继承 SurfaceView
6    implements SurfaceHolder.Callback{                          //实现 SurfaceHolder 接口
7         MyActivity activity;                                   // MyActivity 类的引用
8         ……//此处省略了部分成员变量定义的代码，读者可自行查看随书光盘的源代码
9         Bitmap set_pic;                                        //设置图片
10        public MainView(MyActivity activity) {                 //构造器
11             super(activity);                                  //实现父类的方法
12             this.activity=activity;
13             getHolder().addCallback(this);                    //注册回调接口
14        }
15        @Override
16        public void onDraw(Canvas canvas){                     //重写 onDraw 方法
17             super.onDraw(canvas);
18             canvas.clipRect(new Rect(0,0,Constant.SCREEN_WIDTH,Constant.SCREEN_
               HEIGHT));
19             canvas.drawColor(Color.WHITE);                    //界面设置为白色
20             canvas.drawBitmap(mainbackground, 0, 0, paint);//绘制主界面的背景图片
21             startbutton.drawSelf(canvas, paint);              //绘制开始按钮
22             set.drawSelf(canvas, paint);                      //绘制设置按钮
23        }
24        @Override
25        public boolean onTouchEvent(MotionEvent event){        //重写 onTouchEvent 方法
26             pointx=(float) event.getX();                      //获取触控点
27             pointy=(float) event.getY();                      //获取触控点
28             switch(event.getAction()){
29             case MotionEvent.ACTION_DOWN:                     //按下
30                  if(startbutton.isPointInRect(pointx, pointy)){//跳转到游戏界面
31                       startbutton.switchOff();                //变换图片
32                       activity.sendMessage(1);                //发送消息，进入游戏界面
33                  }else if(set.isPointInRect(pointx, pointy)){
34                       activity.sendMessage(2);                //跳转到背景音乐控制页面
35                  }
36                  break;
37             case MotionEvent.ACTION_UP:                       //抬起
38                  startbutton.switchOn();                      //变换图片
39                  break;
40             }
41             return true;                                      //返回 true
```

```
42          }
43          @Override
44          public void surfaceCreated(SurfaceHolder holder) {   //重写surfaceCreated方法
45               ……//此处省略了加载资源和启动线程的代码，读者可自行查看随书光盘的源代码
46          }
47          @Override
48          public void surfaceDestroyed(SurfaceHolder holder) {keythread.setKeyFlag
            (false);}                                            //标志位设为 false
49          public void initBitmap(){                            //初始化图片资源
50               ……//此处省略了加载图片的代码，读者可自行查看随书光盘的源代码
51          }
52          private class KeyThread extends Thread{              //刷帧线程
53               ……//此处省略了绘制线程类的实现，读者可自行查看随书光盘的源代码
54          }
55          public void repaint{
56               ……//此处省略了 repaint 类的实现，读者可自行查看随书光盘的源代码}
57          @Override
58          public void surfaceChanged(SurfaceHolder holder, int format, int width,int
            height) {
59     }}
```

- 代码第 16-22 行分别表示的是在相应的位置绘制“开始游戏”按钮和“设置”按钮，当触控到“开始按钮”后，跳转到游戏界面，触控到“设置”按钮跳转到声音控制界面。
- 代码第 24-42 行表示的是当按下“开始游戏”按钮的触控范围时，“开始游戏”按钮图片变换成另一张图片，游戏开始，触控到“设置”按钮，则游戏直接进入声音控制界面。
- 代码第 43-59 行为重写 surfaceCreated 和 surfaceDestroyed 方法，并且自定义了刷帧线程和一个 repaint()方法，repaint()方法被刷帧线程调用，不断地绘制主菜单界面。

说明

声音控制界面和游戏通关界面非常类似，由于本书篇幅所限，这里不再赘述，请读者自行查看随书光盘中的源代码。

18.6 游戏界面相关类

上一节中介绍了辅助界面相关类的实现，本节将介绍游戏中最重要的游戏界面相关类的实现。游戏界面相关类是游戏的主体类，也是实现最为复杂的类，下面将对其进行详细介绍。首先介绍的是游戏主界面类 MySurfaceView 的设计与实现。

18.6.1 游戏界面 MySurfaceView 类的设计与实现

MySurfaceView 类是游戏的界面类，游戏中所有用到的组件，最终都会在 MySurfaceView 类中绘制，下面将详细介绍该类的设计及开发，其步骤如下。

（1）MySurfaceView 类继承自 GLSurfaceView 类，采用 OpenGL ES 1.0 绘制游戏的主界面，下面首先介绍 MySurfaceView 类的代码框架，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1    package com.bn.pb;                                    //声明包名
2    import static com.bn.pb.Constant.*;                   //静态导入
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
```

```
4    import android.view.MotionEvent;                          //导入相关类
5    class MySurfaceView extends GLSurfaceView{                //创建继承 SurfaceView 的类
6         ......//该处省略了本类成员变量的代码，读者可以自行查阅随书光盘的源代码
7         public MySurfaceView(Context context){               //构造器
8           super(context);                                    //调用父类
9           this.father=(MyActivity)context;                   //设置摄像机的位置
10         cx=(float)(Math.cos(Math.toRadians(xAngle))*Math.sin(Math.toRadians(yAngle))
           *DISTANCE);
11         cz=(float)(Math.cos(Math.toRadians(xAngle))*Math.cos(Math.toRadians(yAngle))
           *DISTANCE);
12          cy=(float)(Math.sin(Math.toRadians(xAngle))*DISTANCE); //摄像机 y 坐标
13          mRenderer = new SceneRenderer();                   //创建场景渲染器
14          setRenderer(mRenderer);                            //设置渲染器
15          setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);//设置渲染模式为主动渲染
16          rows=Constant.MAP[Constant.COUNT].length;                //得到行数
17          cols=Constant.MAP[Constant.COUNT][0].length;             //得到列数
18          objectMap=new int[rows][cols];                           //创建二维数组
19          for(int i=0;i<rows;i++){                                 //for 循环
20                for(int j=0;j<cols;j++){                           //嵌套的 for 循环
21                     objectMap[i][j]=Constant.MAP[Constant.COUNT][i][j];//为二维数组赋值
22        } } }
23       ......//该处省略了重写的 onTouchEvent 方法的代码，将在后面给出
24        ......//该处省略了键盘监听事件的代码，将在后面给出
25        ......//该处省略了渲染内部类的代码，将在后面给出
26        public void visualAngle(GL10 gl, float ratio){  //第一视角和第三视角的视角大小
27             if(viewFlag){                                  //第一视角
28             gl.glFrustumf(-ratio*0.3f, ratio*0.3f, -1, 1, 0.45f, 100);//设置某一投影
29        }else{                                              //第三视角计算产生透视投影矩阵
30             gl.glFrustumf(-ratio, ratio, -1, 1, 3, 1000);   //设置另一投影
31       } }
32        public void cameraPosition(GL10 gl){                     //视角变化后
33             if(viewFlag){                                       //第一视角
34             GLU.gluLookAt(  gl,                                 //画笔
35                  (float)(robotgroup.positionx+6*Math.sin(Math.toRadians(robotgroup.
                    yAngle+180))),
36                  8*CUBE_SIZE+0.3f,                              //3/2 倍的木箱高度
37                  (float)(robotgroup.positionz+6*Math.cos(Math.toRadians(robotgroup.
                    yAngle+180))),
38                 (float)(robotgroup.positionx+0.5f*Math.sin(Math.toRadians(robotgroup.
                    yAngle+180))),
39                  2*CUBE_SIZE+0.3f,                              //目标点的 y 坐标
40                  (float)(robotgroup.positionz+0.5f*Math.cos(Math.toRadians(robotgroup.
                    yAngle+180))),
41                  0,                                             //UP 向量的 x 坐标
42                  1,0                                            //UP 向量的 y 与 z 坐标
43             );
44          }else{                                                //第三视角
45             GLU.gluLookAt
46             (gl, cx,cy,cz, 0,0, 0,0,1, 0);                     //第三视角，永远看向原点
47       } }
48        ......//该处省略了判断机器人按某一方向运行的代码，将在后面给出
49        ......//该处省略了绘制虚拟按钮的代码，将在后面给出
50        ......//该处省略了加载本游戏用到的所有纹理，将在后面给出
51        ......//该处省略了初始化本游戏用到的全部组件对象的代码，将在后面给出
52        public void initSound(){                                 //初始化声音
53             if(mMediaPlayer==null){                             //背景音乐
54                  mMediaPlayer=MediaPlayer.create(father, R.raw.sound1);
                                                                   //创建背景音乐播放器
55                  mMediaPlayer.setLooping(true);                 //循环播放
56                  if(father.soundFlag){                          //soundFlag 为 true 时
57                       mMediaPlayer.start();                     //开启音乐
58        } } }
```

```
59          ......//该处省略了初始化纹理的代码，将在后面给出
60          ......//该处省略了当前关卡的游戏结束，关闭线程和声音的代码，将在后面给出
61          ......//该处省略了游戏结束，判断是进入下一关游戏，还是进入最后的通关界面的代码，将在后面给出
62      }
```

- 代码第 7-22 行 MySurfaceView 类构造器中实现了摄像机的位置赋值、创建了场景渲染器、设定渲染模式为主动渲染并且复制了当前关的二维数组。
- 代码第 26-47 行为自定义的两个方法，本游戏可以变换视角，第一视角时，摄像机总是在机器人的后上方看向机器人，第三视角时，则摄像机永远在地图中心上方看向原点。
- 代码第 52-58 行为加载背景音乐的方法，创建 MediaPlayer 的对象、设置为循环播放、当标志位为 true 时，开启背景音乐，并且背景音乐只在 MySurfaceView 界面才会播放。

（2）onTouchEvent 方法用于第三视角时，拖动屏幕来旋转摄像机的位置，以及触控虚拟按钮时机器人执行旋转或者前进动作，将下列代码插入到 MySurfaceView 类代码第 23 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1     public boolean onTouchEvent(MotionEvent e){                    //重写 onTouchEvent 方法
2         float y = e.getY();                                        //触控点的 x 坐标
3         float x = e.getX();                                        //触控点的 y 坐标
4         switch (e.getAction()){                                    //判断动作类型
5         case MotionEvent.ACTION_DOWN:                              //按下动作
6            float yRatio=y/screenHeight;                            //得到 y 偏移量
7             float xRatio=x/screenWidth;                            //得到 x 偏移量
8             if(flag){                                              //flag 为 true
9                     tempRow=robotgroup.m;                          //得到当前关机器人所在的行数
10                    tempCol=robotgroup.n;                          //得到当前关机器人所在的列数
11                    flag=false;                                    //设置 flag 为 false
12            }
13        if( yRatio>viewYMin&&yRatio<viewYMax&&xRatio>viewXMin&&xRatio<viewXMax ){
14                viewFlag=!viewFlag;                                //设置 viewFlag 的值
15        }else if( (yRatio>upYMin&&yRatio<upYMax&&                  //上下左右的虚拟按钮
16            xRatio>upXMin&&xRatio<upXMax )&&
17            robotgroup.robot.flag==true){                          //后转身按钮
18            keyState=1;                                            //前进
19        }else if( ( yRatio>downYMin&&yRatio<downYMax&&
20                    xRatio>downXMin&&xRatio<downXMax) &&
21                    robotgroup.robot.flag==true){                  //按向下虚拟按钮
22                    ......//该处省略了后转身的代码，读者可以查阅随书光盘中的源代码
23        }else if( (yRatio>leftYMin&&yRatio<leftYMax&& //左转身按钮
24                      xRatio>leftXMin&&xRatio<leftXMax) &&
25                      robotgroup.robot.flag==true){                //按下向左虚拟按钮
26                    ......//该处省略了左转身的代码，读者可以查阅随书光盘中的源代码
27        }else if( ( yRatio>rightYMin&&yRatio<rightYMax&&   //右转身按钮
28                        xRatio>rightXMin&&xRatio<rightXMax) &&
29                        robotgroup.robot.flag==true){              //按下向右虚拟按钮
30                    ......//该处省略了右转身的代码，读者可以查阅随书光盘中的源代码
31        }
32        break;                                                     //退出
33    case MotionEvent.ACTION_UP:                                    //抬起动作
34      keyState=0;                                                  //键盘状态
35      break;                                                       //退出
36    case MotionEvent.ACTION_MOVE:
37        float dy = y - mPreviousY;                                 //计算触控笔 Y 位移
38        float dx = x - mPreviousX;                                 //计算触控笔 X 位移
39        yAngle += dx * TOUCH_SCALE_FACTOR;                         //仰角改变
40        xAngle += dy * TOUCH_SCALE_FACTOR;                         //方位角改变
41        if(xAngle<0){                                              //限制，不可以看到反面
```

```
42                  xAngle=0;                                   //xAngle 为 0
43          }
44          if(xAngle>=90){                                     //限制
45                  xAngle=90;                                  //xAngle 为 90
46          }
47          if(xAngle<=-90){                                    //限制
48                  xAngle=-90;                                 //xAngle 为-90
49          }
50              requestRender();                                //重绘画面
51          break;                                              //退出
52      }
53      mPreviousY = y;                                         //记录触控笔位置
54      mPreviousX = x;                                         //记录触控笔位置
55      cx=(float)(Math.cos(Math.toRadians(xAngle))*Math.sin(Math.toRadians(yAngle))
        *DISTANCE);
56      cz=(float)(Math.cos(Math.toRadians(xAngle))*Math.cos(Math.toRadians(yAngle))
        *DISTANCE);
57      cy=(float)(Math.sin(Math.toRadians(xAngle))*DISTANCE);//摄像机 y 坐标
58      return true;                                            //返回 true
59  }
```

● 代码第 1-32 行当按下“前进”按钮时，keyState 的值设为 1，按下“左转”、“右转”和“后转”按钮时，机器人分别向左、向右、向后旋转，按下切换视角按钮时，变化视角。

● 代码第 33-59 行为在第三视角的情况下，拖动屏幕，变化摄像机位置，从而产生地图视角变换的效果，并且限制摄像机旋转角度，使摄像机不能看到地板的反面。

说明　onKeyDown 方法和 onTouchEvent 方法功能及代码基本类似，由于本书篇幅所限，这里不再赘述，请读者自行查看随书光盘中的源代码。

（3）前面介绍了 onTouchEvent 方法，下面将介绍绘制该游戏界面的渲染内部类 SceneRenderer，该类中 onDrawFrame 方法是系统一直在调用的方法，将下列代码插入到 MySurfaceView 类第 25 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1   private class SceneRenderer implements GLSurfaceView.Renderer{
2       float ratio=0;                                          //设置 ratio 为 0
3       public void onDrawFrame(GL10 gl){                       //绘制方法
4           visualAngle(gl,ratio);                              //调用 visualAngle 方法
5           gl.glShadeModel(GL10.GL_SMOOTH);                    //采用平滑着色
6           gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);//清除缓存
7           gl.glMatrixMode(GL10.GL_MODELVIEW);                 //设置当前矩阵为模式矩阵
8           gl.glLoadIdentity();                                //设置当前矩阵为单位矩阵
9           cameraPosition(gl);                                 //调用cameraPosition方法
10          gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);       //顶点数组
11          gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);//顶点坐标
12          gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);       //允许使用法向量数组
13          gl.glEnable(GL10.GL_TEXTURE_2D);                    //启用纹理
14          gl.glPushMatrix();                                  //绘制地板
15          gl.glTranslatef(0, 0, 0);                           //平移
16          floor.drawSelf(gl, floorId);                        //绘制
17          gl.glPopMatrix();                                   //恢复矩阵
18          gl.glPushMatrix();                                  //绘制墙面
19          gl.glTranslatef((-Constant.MAP[Constant.COUNT][0].length/2)*UNIT_SIZE,0,、
20                          -Constant.MAP[Constant.COUNT].length/2*UNIT_SIZE);
21          wall.drawSelf(gl, wallId);                          //绘制
22          walloutside.drawSelf(gl, wallId);                   //绘制墙体的最外层
23          gl.glPopMatrix();                                   //恢复矩阵
```

```
24          gl.glPushMatrix();                                  //绘制木箱
25          gl.glTranslatef(0, 0, 0);                           //平移
26          cubegroup.drawSelf(gl);                             //绘制
27          gl.glPopMatrix();                                   //恢复矩阵
28          gl.glPushMatrix();                                  //绘制矩形纹理（即地面和目标）
29          gl.glTranslatef(0, 0, 0);                           //平移
30          texturerectgroup.drawSelf(gl);                      //调用 texturerectgroup 方法
31          gl.glPopMatrix();                                   //恢复矩阵
32          gl.glPushMatrix();                                  //绘制机器人
33          gl.glTranslatef(0, 0, 0);                           //平移
34          robotgroup.drawSelf(gl);                            //绘制
35          gl.glPopMatrix();                                   //恢复矩阵
36          gl.glPushMatrix();                                  //绘制木箱运动动画
37          gl.glTranslatef(0, 0, 0);                           //平移
38          cubego.drawSelf(gl);                                //绘制
39          gl.glPopMatrix();                                   //恢复矩阵
40          draw_button(gl);                                    //绘制虚拟按钮
41          gl.glDisableClientState(GL10.GL_NORMAL_ARRAY);//禁止顶点法向量
42          gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);//禁止顶点数组
43          gl.glDisable(GL10.GL_TEXTURE_2D);                   //禁止纹理
44          gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);       //禁止顶点坐标
45          judgeGoToLastViewOrGoToNext();                                //调用方法
46       }
47       public void onSurfaceChanged(GL10 gl, int width, int height){ //重写的方法
48             screenWidth=width;                                //得到屏幕宽度
49             screenHeight=height;                              //得到屏幕高度
50             gl.glViewport(0, 0, width, height);               //设置视窗大小及位置
51             gl.glMatrixMode(GL10.GL_PROJECTION);              //设置当前矩阵为投影矩阵
52             gl.glLoadIdentity();                              //设置当前矩阵为单位矩阵
53             ratio = (float) width / height;                   //计算透视投影的比例
54       }
55       public void onSurfaceCreated(GL10 gl, EGLConfig config){   //重写的方法
56             gl.glDisable(GL10.GL_DITHER);                          //关闭抗抖动
57             gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.GL_FASTEST);
                                                          //设置 Hint 模式
58             gl.glEnable(GL10.GL_CULL_FACE);            //设置为打开背面剪裁
59             gl.glShadeModel(GL10.GL_SMOOTH);           //设置着色模型为平滑着色
60             gl.glEnable(GL10.GL_BLEND);                //开启混合
61             gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
                                                          //设置源混合
62             gl.glClearColor(0,0,0,0);                  //设置屏幕背景色黑色 RGBA
63             gl.glEnable(GL10.GL_DEPTH_TEST);           //启用深度测试
64             loadTexture(gl);                           //调用方法，加载本游戏用到的所有纹理
65             initObject();                              //本游戏用到的全部组件对象的初始化
66             initSound();                               //初始化，并播放声音
67             kt=new KeyThread(father.gameView);         //创建并启动键盘监控线程
68             kt.flag=true;                              //设置 flag 为 true
69             kt.start();                                //开启线程
70    } }
```

- 代码第 5-39 行为设置当前矩阵为单位矩阵、启用纹理、采用平滑着色后绘制机器人、普通木箱、红色木箱、目标矩形、地板和墙面等游戏组件。
- 代码第 40 行为恢复当前矩阵为单位矩阵后绘制虚拟按钮，这样是为了防止旋转摄像机的位置时，虚拟按钮在屏幕上的位置变化。
- 代码第 41-46 行为绘制完游戏中用到的组件和虚按钮后，禁用顶点纹理等，并且调用 judgeGoToLastViewOrGoToNext 方法判断是否需要进入下一关游戏。

（4）介绍完游戏界面的渲染内部类 SceneRenderer 后，下面将介绍机器人向 z 轴正方向运动时，

当机器人遇到普通箱子或者墙面后执行不同动作的方法 zRightDirection，将下列代码插入到 MySurfaceView 类中代码第 48 行，zRightDirection 方法的代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1    public void zRightDirection(){                          //如果机器人向 z 轴正方向运动
2         tempRow+=1;                                        //行数加 1(将要运动到的行)
3         switch(objectMap[tempRow][tempCol])                //判断前方是什么，然后产生不同的结果
4         {
5              case 1:                                       //遇到墙
6              case 6:                                       //遇到木箱（红色木箱）
7                   tempRow=tempRow-1;                       //既然不能走，那么提前加的行数减回来
8                   break;
9              case 2:                                       //遇到路
10             case 3:                                       //遇到目标
11             case 5:                                       //遇到机器人
12                  robotgroup.robot.rotate(0);              //开启机器人走路动画
13                  break;                                   //退出
14             case 4:                                       //遇到箱子
15                  if(objectMap[tempRow+1][tempCol]==1||
16                            objectMap[tempRow+1][tempCol]==4
17                            ||objectMap[tempRow+1][tempCol]==6) //箱子将要被推到的地方
18                  {
19                       tempRow=tempRow-1;                  //推不动箱子
20                  }else if(objectMap[tempRow+1][tempCol]==3){//箱子将要推到的地方是目标点
21                       robotgroup.robot.pushbox(0);        //机器人推箱子动画
22                       if(father.soundPoolFlag){
23                            father.playSound(1, 0);        //推箱子的声音
24                       }
25                       objectMap[tempRow][tempCol]=2;      //以前有箱子的地方画路
26                       cubego.flag=true;                   //设置 flag 为 true
27                       cubego.m=tempRow;                   //当前木箱所在的位置绘制木箱
28                       cubego.n=tempCol;
29                       cubego.cubeGo(0);                   //播放箱子运动动画
30                  }else{                                   //箱子遇到的是路，机器人
31                       robotgroup.robot.pushbox(0);        //机器人推箱子动画
32                       if(father.soundPoolFlag){           //soundPoolFlag 为 true
33                            father.playSound(1, 0);        //推箱子的声音
34                       }
35                       objectMap[tempRow][tempCol]=2;      //以前有箱子的地方画路
36                       cubego.flag=true;                   //设置 flag 为 true
37                       cubego.m=tempRow;                   //当前木箱所在的位置绘制木箱
38                       cubego.n=tempCol;                   //设置 n
39                       cubego.cubeGo(0);                   //播放箱子运动动画
40                  }
41             break;                                        //退出
42   } }
43   public void znegativeDirection(){                       //创建 znegativeDirection 方法
44        ......//该处省略了机器人向 Z 轴负方向运动的代码
45   }
46   public void xRightDirection(){                          //创建 xRightDirection 方法
47        ......//该处省略了机器人向 x 轴正方向运动的代码
48   }
49   public void xnegativeDirection(){                       //创建 xnegativeDirection 方法
50        ......//该处省略了机器人向 x 轴负方向运动的代码
51   }
```

- 代码第 5-13 行为当机器人遇到墙或者是已经推到目标点的木箱时，机器人不执行走路动画，将提前加 1 的行数减回来，当机器人遇到目标矩形或者路，执行走路动画。

- 代码第 14-42 行为当机器人遇到木箱时，当箱子要被推到的地方是墙、普通木箱或者是已经推好的木箱时，机器人原地不动，如果箱子要被推到的地方是路或者是目标矩形，则执行机器人推箱子动画和箱子运动动画，并且在原箱子所在的地方画路。

说明 znegativeDirection 方法、xRightDirection 方法、xnegativeDirection 方法与 zRightDirection 方法的功能及代码基本类似，由于本书篇幅所限，这里不再赘述，请读者自行查看随书光盘中的源代码。

（5）介绍完 zRightDirection 方法后，下面将介绍绘制虚拟按钮的 draw_button 方法，将下列代码插入到 MySurfaceView 类代码第 49 行，draw_button 方法的代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1    public void draw_button(GL10 gl){                          //绘制虚拟按钮
2          gl.glMatrixMode(GL10.GL_MODELVIEW);                  //设计矩阵模式
3          gl.glLoadIdentity();                                 //设置当前矩阵为单位矩阵
4          gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);        //启用定点数组
5          gl.glEnable(GL10.GL_TEXTURE_2D);                     //启用纹理
6          gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
7          gl.glPushMatrix();                                   //保存矩阵
8          gl.glTranslatef(1.3f, -0.3f, -4);                    //平移
9          buttonUp.drawSelf(gl);                               //绘制
10         gl.glPopMatrix();                                    //恢复矩阵
11         gl.glPushMatrix();                                   //保存矩阵
12         gl.glTranslatef(1.3f, -1.0f, -4);                    //平移
13         buttonDown.drawSelf(gl);                             //绘制
14         gl.glPopMatrix();                                    //恢复矩阵
15         gl.glPushMatrix();                                   //保存矩阵
16         gl.glTranslatef(0.9f, -0.65f, -4);                   //平移
17         buttonLeft.drawSelf(gl);                             //绘制
18         gl.glPopMatrix();                                    //恢复矩阵
19         gl.glPushMatrix();                                   //保存矩阵
20         gl.glTranslatef(1.7f, -0.65f, -4);                   //平移
21         buttonRight.drawSelf(gl);                            //绘制
22         gl.glPopMatrix();                                    //恢复矩阵
23         gl.glPushMatrix();                                   //保存矩阵
24         gl.glTranslatef(1.3f, 1.0f, -4);                     //平移
25         view.drawSelf(gl);                                   //绘制
26         gl.glPopMatrix();                                    //恢复矩阵
27         gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);       //关闭顶点
28         gl.glDisable(GL10.GL_TEXTURE_2D);                    //关闭纹理
29         gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
30   }
```

- 代码第 1-6 行设置当前矩阵为模式矩阵、设置当前矩阵为单位矩阵可以防止拖动手机屏幕时，虚拟按钮在手机屏幕上的位置随着摄像机位置变化而变化。
- 代码第 7-30 行为在手机屏幕的相应位置绘制“前进”、“左转”、“右转”、“ 后转”按钮和屏幕右上角的切换视角按钮，绘制完毕后，禁用纹理、关闭顶点法向量。

（6）介绍完绘制虚拟按钮的 draw_button 方法后，下面将介绍加载该游戏中用到的所有纹理的 loadTexture 方法，将下列代码插入到 MySurfaceView 类代码第 50 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1    public void loadTexture(GL10 gl){                                  //加载本游戏用到的所有纹理
2         floorId=initTexture(gl,R.drawable.floor);                     //地面纹理 ID
3         wallId=initTexture(gl,R.drawable.wall);                       //墙纹理 ID
4         woodbinId0=initTexture(gl,R.drawable.wood_bin0);              //木箱纹理 ID
5         woodbinId1=initTexture(gl,R.drawable.wood_bin1);              //目标纹理 ID
6         targetId=initTexture(gl,R.drawable.target);                   //目标纹理 ID
7         robot_texId=initTexture(gl,R.drawable.robot_texid);           //机器人纹理 ID
8         robot_head=initTexture(gl,R.drawable.robot_head);
9         button_up=initTexture(gl,R.drawable.b_up);                    //上键的纹理 ID
10        button_down=initTexture(gl,R.drawable.b_down);                //下键的纹理 ID
11        button_left=initTexture(gl,R.drawable.b_left);                //左键的纹理 ID
12        button_right=initTexture(gl,R.drawable.b_right);              //右键的纹理 ID
13        viewId=initTexture(gl,R.drawable.view);                       //视角纹理
14   }
```

提示　loadTexture 方法调用 initTexture 方法加载了该游戏中用到的地板纹理、目标矩形纹理、机器人头和身体纹理、墙面纹理以及 5 个虚拟按钮的矩形纹理。

（7）前面介绍了游戏中加载纹理的 loadTexture 方法，下面将介绍为该游戏中用到的各个组件引用创建对象的 initObject 方法，将下列代码插入到 MySurfaceView 类代码第 51 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1    public void initObject(){                                          //本游戏用到的全部组件对象的初始化
2         floor=new Floor((int)(-Constant.MAP[Constant.COUNT][0].length/2),
3                         (int)(-Constant.MAP[Constant.COUNT].length/2),
4                         1,0,Constant.MAP[Constant.COUNT][0].length,
5                         Constant.MAP[Constant.COUNT].length);//地板类的对象
6         wall=new Wall();                                                //墙类的对象
7         walloutside=new WallOutSide();                                  //墙体的最外层类的对象
8         cubegroup=new CubeGroup(father.gameView,woodbinId0,woodbinId1);//木箱组类的对象
9         texturerectgroup=new TextureRectGroup(father.gameView,targetId);
                                                                          //（目标组）矩形纹理组类的对象
10        robotgroup=new RobotGroup(father.gameView,robot_texId,robot_head);
11        cubego=new CubeGo(father.gameView,Constant.CUBE_SIZE,woodbinId0);
12        buttonUp=new TextureRect(0.25f,0.25f,button_up);                //向上的虚拟按钮类声明
13        buttonDown=new TextureRect(0.25f,0.25f,button_down);            //向下的虚拟按钮类声明
14        buttonLeft=new TextureRect(0.25f,0.25f,button_left);            //向左的虚拟按钮类声明
15        buttonRight=new TextureRect(0.25f,0.25f,button_right);          //向右的虚拟按钮类声明
16        view=new TextureRect(0.25f,0.25f,viewId);                       //视角纹理
17   }
```

提示　initObject 方法是将游戏中用到的各个组件的对象放在一起组成的一个方法，便于查找和管理。

（8）在前面介绍的加载纹理的 loadTexture 方法中调用了 initTexture 方法，加载了该游戏用到的全部纹理，下面介绍初始化纹理的 initTexture 方法，将下列代码插入到 MySurfaceView 类代码第 59 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1    public int initTexture(GL10 gl,int drawableId){                    //初始化纹理
2         int[] textures = new int[1];                                  //生成纹理 ID
3         gl.glGenTextures(1, textures, 0);                             //纹理
4         int currTextureId=textures[0];
```

```
5          gl.glBindTexture(GL10.GL_TEXTURE_2D, currTextureId);     //绑定
6          gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,
7                                  GL10.GL_LINEAR_MIPMAP_NEAREST);//设置
8          gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER,
9                                  GL10.GL_LINEAR_MIPMAP_LINEAR); //生成Mipmap纹理
10         ((GL11)gl).glTexParameterf(GL10.GL_TEXTURE_2D,
11                                  GL11.GL_GENERATE_MIPMAP,GL10.GL_TRUE);//纹理拉伸方式
12       gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,GL10.GL_REPEAT);
13       gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,GL10.GL_REPEAT);
14       InputStream is = this.getResources().openRawResource(drawableId);//通过输入流加载
15       Bitmap bitmapTmp;                                          //创建Bitmap的引用
16       try{
17         bitmapTmp = BitmapFactory.decodeStream(is);              //加载图片
18       }finally{                                                  //最后执行
19           try{
20               is.close();                                        //关闭输入流
21           }catch(IOException e){                                 //捕获异常
22               e.printStackTrace();                               //打印堆栈信息
23       } }
24       GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmapTmp, 0);   //贴纹理
25       bitmapTmp.recycle();                                       //加载后释放图片
26       return currTextureId;                                      //返回currTextureId
27   }
```

提示 initTexture 方法是通过打开输入流，加载并生成当前纹理的 ID，没有弄清楚的读者可以复制此方法，直接使用，在今后的使用中慢慢理解此方法。

（9）由于在 3D 游戏的渲染类中，系统并没有像开发 2D 游戏一样，存在一个 onSurfaceDestroyed 方法，所以，在这里自定义了一个游戏结束的 overGame 方法，当游戏主界面被撤销时，调用此方法关闭游戏中开启的线程和游戏的背景音乐等，将下列代码插入到 MySurfaceView 类代码第 60 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1    public void overGame(){                          //当前关卡的游戏结束，关闭线程和声音等
2          boolean retry = true;                      //设置rety为true
3          kt.flag=false;                             //关闭所有线程
4        while (retry){                               //while循环
5            try{
6                    kt.join();                       //线程等待
7                retry = false;                       //设置rety为false
8            } catch (InterruptedException e){        //捕获异常
9                e.printStackTrace();                 //打印堆栈信息
10       } }
11         if(mMediaPlayer.isPlaying()){              //判断isPlaying方法的返回值
12               mMediaPlayer.stop();                 //关闭媒体播放器
13   } }
```

提示

overGame 为游戏主界面被撤销时调用的方法，此方法的作用是用来关闭游戏进行过程中开启的线程和正在播放的游戏背景音乐。

（10）游戏在手机上运行后，还有一个如何过关的问题，下面介绍当完成当前关卡后，判断是进入下一关游戏还是进入通关界面的方法，将下列代码插入到 MySurfaceView 类代第 61 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MySurfaceView.java。

```
1    public void judgeGoToLastViewOrGoToNext(){              //游戏结束调用的方法
2         int row=0;                                         //行数 row 为 0
3         int col=0;                                         //列数 col 为 0
4         int targetCount=0;                                 //targetCount 为 0
5         row=objectMap.length;                              //初始化行数
6         col=objectMap[0].length;                           //初始化列数
7         for(int i=0;i<row;i++){                            //for 循环
8              for(int j=0;j<col;j++){                       //嵌套的 for 循环
9                   if(objectMap[i][j]==3){                  //目标数量
10                       targetCount++;                       //targetCount 的值加 1
11        } } }
12        if(Constant.COUNT==MAP.length-1&&targetCount==0){    //游戏结束
13             overGame();                                     //调用 overGame 方法
14             father.sendMessage(3);                          //进入最后的界面
15        }else if(Constant.COUNT<MAP.length-1&&targetCount==0){//目标的数量为 0，游戏结束
16             overGame();                                     //调用 overGame 方法
17             Constant.COUNT++;                               //COUNT 值加 1
18             father.sendMessage(1);                          //进入下一关游戏界面
19             try{
20                  Thread.sleep(100);                         //使主线程休眠一会
21             }catch(Exception e){                            //捕获异常
22                  e.printStackTrace();                       //打印堆栈信息
23   } } }
```

- 代码第 2-18 行是不断扫描 MySurfaceView 类中复制的当前关卡的二维数组，当目标矩形的数量为零时，表明当前关卡游戏结束。当前关卡结束后，如果当前关数等于此游戏关卡的总数，那么游戏结束，进入通关界面，否则游戏进入下一关。
- 代码第 19-23 行为使主线程停止 100 毫秒，为游戏进入下一关做准备。

18.6.2　键盘监听线程 KeyThread

前面介绍的 MySurfaceView 类中，用到了 KeyThread 线程，该线程的作用是不断地循环判断 keyState 的值，如果 keyState 为 1，则表明前进按钮一直被按下，执行机器人运动动画，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 KeyThread.java。

```
1    package com.bn.pb;                                  //声明包名
2    public class KeyThread extends Thread{              //监听键盘状态的线程
```

```
3          MySurfaceView mv;                                  //声明 MySurfaceView 的引用
4          boolean flag=true;                                 //设置标志位 flag 为 true
5          public KeyThread(MySurfaceView mv){                //构造器
6               this.mv=mv;                                   //初始化 MySurfaceView 的引用
7          }
8          public void run(){                                 //继承 Thread 类需要重写的 run()方法
9               while(flag){
10                   if(mv.keyState==1&&mv.robotgroup.robot.flag==true){//有 UP 键按下
11                        if(mv.robotgroup.yAngle==0){     //z 轴正方向运动
12                             mv.zRightDirection();  //通过 mv 调用 zRightDirection 方法
13                        }else if(mv.robotgroup.yAngle==180){ //z 轴负方向
14                             mv.znegativeDirection();   //通过 mv 调用
                                                          //znegativeDirection 方法
15                        }else if(mv.robotgroup.yAngle==270){ //x 轴负方向
16                             mv.xnegativeDirection();   //通过 mv 调用
                                                          //xnegativeDirection 方法
17                        }else if(mv.robotgroup.yAngle==90){  //x 轴正方向
18                             mv.xRightDirection();  //通过 mv 调用 xRightDirection 方法
19                   } }
20                   try{
21                        Thread.sleep(100);               //休眠 100ms
22                   } catch (InterruptedException e){     //捕获异常
23                        e.printStackTrace();             //打印堆栈信息
24   } } } }
```

- 代码第 3-4 行为本类的成员变量，主要有 MySurfaceView 类的引用以及 flag 循环标志位。
- 代码第 5-7 行为本类的构造器，在该构造器中主要是初始化 MySurfaceView 类的对象。
- 代码第 8-24 行为继承 Thread 类需要重写的方法，在该方法中，根据 flag 的值开启 while 循环。首先判断是否按下 UP 键，如果 UP 键被按下，则根据 yAngle 的值调用不同的方法。

18.7 自定义控件及工具类

介绍完游戏界面相关类后，本节将介绍该游戏中用到的图片缩放的工具类 PicLoadUtil 和主菜单类中用到的虚拟按钮类 MainMenuButton，首先介绍 PicLoadUtil 类。

18.7.1 缩放图片的方法 PicLoadUtil

PicLoadUtil 工具类用于图片的缩放，主要作用是使该游戏可以适应各种分辨率的手机，从而使该游戏的通用性更强，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 PicLoadUtil.java。

```
package com.bn.pb;                                            //声明包名
import android.graphics.Bitmap;                               //导入相关类
import android.graphics.Matrix;                               //导入相关类
public class PicLoadUtil{
   public static Bitmap scaleToFit(Bitmap bm,float targetWidth,float targetHeight){
                                                              //缩放图片的方法
      float width = bm.getWidth();                            //图片宽度
      float height = bm.getHeight();                          //图片高度
      Matrix m1 = new Matrix();                               //创建 Matrix 对象
      m1.postScale(targetWidth/width, targetHeight/height);   //这里指的是目标区域
      Bitmap bmResult = Bitmap.createBitmap(bm, 0, 0, (int)width, (int)height, m1, true);
                                                              //声明位图
```

```
        return bmResult;                                    //返回 Bitmap 对象
    } }
```

提示　PicLoadUtil 工具类中的 scaleToFit 方法通过传递来的位图资源的引用，以及 targetWidth 和 targetHeight 的值，利用矩阵不等比例拉伸图片，从而使该位图资源适应不同分辨率的手机。

18.7.2　主菜单上的按钮类 MainMenuButton

MainMenuButton 类也是一个封装好的类，用于 2D 游戏开发中虚拟按钮的开发，在该游戏中，该类用于绘制主菜单界面中的虚拟按钮，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com.bn.pb 目录下的 MainMenuButton.java。

```
1    package com.bn.pb;                                     //声明包名
2    import android.graphics.Bitmap;                        //导入相关类
3    import android.graphics.Canvas;                        //导入相关类
4    import android.graphics.Paint;                         //导入相关类
5    public class MainMenuButton{
6         MyActivity activity;                              //创建 MyActivity 的引用
7         int x;                                            //图片的左上点 x 坐标
8         int y;                                            //图片的左上点 y 坐标
9         int width;                                        //虚拟按钮的宽
10        int height;                                       //虚拟按钮的高
11        Bitmap onBitmap;                                  //按下图片
12        Bitmap offBitmap;                                 //抬起图片
13        boolean isOn=true;                                //按下状态为 false
14        public MainMenuButton(MyActivity activity,Bitmap onBitmap,Bitmap offBitmap,
          int x,int y){
15             this.activity=activity;                      //初始化 MyActivity 的对象
16             this.onBitmap=onBitmap;                      //初始化按下的图片
17             this.offBitmap=offBitmap;                    //初始化抬起的图片
18             this.x=x;                                    //初始化图片左上角的 x 坐标
19             this.y=y;                                    //初始化图片左上角的 y 坐标
20             this.width=offBitmap.getWidth();             //初始化虚拟按钮的宽
21             this.height=offBitmap.getHeight();           //初始化虚拟按钮的高
22        }
23        public void drawSelf(Canvas canvas,Paint paint){     //绘制虚拟按钮
24             if(isOn){                                       //判断 isOn 是否为 true
25                  canvas.drawBitmap(onBitmap, x, y, paint); //绘制图片
26             }else{
27                  canvas.drawBitmap(offBitmap, x, y, paint);//绘制图片
28        } }
29        public void setswitch(){                             //创建改变 isOn 值的方法
30             isOn=!isOn;                                     //设置 isOn 的值
31        }
32        public boolean isOnflag(){                           //返回 isOn 的方法
33             return isOn;                                    //返回 isOn 的值
34        }
35        public void switchOn(){                              //设置 isOn 值的方法
36             isOn=true;                                      //返回 isOn 为 true
37        }
38        public void switchOff(){                             //设置 isOn 值的方法
39             isOn=false;                                     //返回 isOn 为 false
40        }
41        public boolean isPointInRect(float pointx,float pointy){
                                                              //一个点是否在矩形内（包括边界）
42             if(                                            //如果在矩形内，返回 true
```

```
43                          pointx>=x&&pointx<=x+width&&
44                          pointy>=y&&pointy<=y+height
45                  ){
46                      return true;                          //如果在，返回 true
47                  }
48              return false;                                 //否则返回 false
49    } }
```

- 第 1-22 行为包的声明、相关类的引入、相关成员变量的声明及赋值，按下和抬起手机屏幕显示的位图资源引用的赋值以及虚拟按钮在手机屏幕位置的赋值。
- 第 23-49 行为绘制虚拟按钮的方法、改变显示图片标志位的 setswitch、isOnflag、switchOn 和 switchOff 方法以及判断一个点是否在矩形内的 isPointInRect 方法。

18.8 情景相关类的实现

在前面游戏架构中介绍了很多的情景相关类，并且用到了很多读者并不熟悉的情景中的实体类，从本节开始将对这些类一一进行讲解。首先介绍的是由玩家控制的机器人类。

18.8.1 机器人类的代码框架

机器人是由半球体和圆柱等图形拼接组成的，要求机器人的左右手臂和左右腿可以自然摆动，由于机器人最终要加入游戏界面中，所以，还要机器人可以等比例放大或缩小。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Robot.java。

```
1    package com.bn.pb;                                        //相关包的声明
2    import javax.microedition.khronos.opengles.GL10;          //相关类的引入
3    import static com.bn.pb.Constant.*;                       //相关类的引入
4    public class Robot {
5         MySurfaceView father;                                //游戏主界面类的引用
6       boolean flag=true;                                     //运动标志位
7         Globe head;                                          //头
8         Globe body_bottom;                                   //身体下部
9         Globe arm_terminal;                                  //胳膊终端
10        Globe leg_terminal;                                  //腿终端
11        Circle head_bottom;                                  //头下部
12        Circle body_top;                                     //身体上部
13        Cylinder body;                                       //身体
14        Cylinder arm;                                        //胳膊
15        Cylinder leg;                                        //腿
16        Cylinder antenna;                                    //天线
17        public static float left_arm_degree=0;               //左胳膊甩动角度
18        public static float right_arm_degree=0;              //右胳膊甩动角度
19        public static float left_leg_degree=0;               //左腿甩动角度
20        public static float right_leg_degree=0;              //右腿甩动角度
21        public static float count=5.0001f;                   //角度增长值
22        int android_texId;                                   //纹理 id
23        int android_head;                                    //机器人头纹理 id
24        int i=0;                                             //控制 rotate()方法只执行一次
25        public Robot(MySurfaceView father,int android_texId,int android_head){
26             this.father=father;                             //为游戏主界面类的引用赋值
27             this.android_texId=android_texId;               //身体纹理图片的 id
28             this.android_head=android_head;                 //头纹理图片的 id
29             head=new Globe(HEAD_RADIUS,15,0);               //头部
```

```
30            head_bottom=new Circle(12,HEAD_RADIUS,new float[]{10,10},HEAD_RADIUS);
                                                          //头下部
31            body=new Cylinder(BODY_RADIUS,BODY_LENGTH,20,0.1F);      //身体
32            body_top=new Circle(12,BODY_RADIUS,new float[]{10,10},BODY_RADIUS);
                                                                      //身体上部
33            body_bottom=new Globe(BODY_RADIUS*1.16f,15,30);          //身体下部
34            arm=new Cylinder(ARM_RADIUS,ARM_LENGTH,20,0.1f);         //胳膊
35            arm_terminal=new Globe(ARM_RADIUS,15,0);                 //胳膊终端
36            leg=new Cylinder(LEG_RADIUS,LEG_LENGTH,20,0.1f);         //腿
37            leg_terminal=new Globe(LEG_RADIUS,15,0);                 //腿终端
38            antenna=new Cylinder(ANTENNA_RADIUS,ANTENNA_LENGTH,20,0.1f);//天线
39        }
40        public void drawSelf(GL10 gl){}                       //绘制机器人的方法
41        public  void rotate(final int robotAngle){}           //机器人走路的动画效果
42        public  void pushbox(final int robotAngle){}          //推箱子的动画效果
43    }
```

- 代码第 1-24 行分别为包的声明、相关类的引入、组成机器人组件类的引用和相关变量的声明。
- 代码第 25-39 行为该类的构造器，其中为机器人的头、胳膊、腿等组件类的引用赋值，并且为游戏中主界面类的引用赋值、头和身体的纹理 id 赋值。

说明　代码框架中的部分方法为空实现，这些方法的实现将在下一小节中详细给出。

18.8.2　机器人类成员方法的实现

上一小节中介绍了 Robot 类的代码框架，本小节将对其中空实现的成员方法进行详细的介绍。下面按代码的开发顺序进行介绍。

（1）drawSelf 方法的实现，该方法分别绘制了机器人的头、胳膊、腿、身体等组件，其代码如下。将下列代码插入到 Robot 类的第 40 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Robot.java。

```
1     public void drawSelf(GL10 gl){
2          gl.glPushMatrix();                                  //绘制头
3          gl.glTranslatef(0, HEAD_Y, 0);                      //移动
4          head.drawSelf(gl, android_head);                    //绘制头
5          gl.glPushMatrix();                                  //入栈
6          gl.glRotatef(90, 1, 0, 0);                          //旋转
7          head_bottom.drawSelf(gl, android_texId);            //绘制头下部
8          gl.glPopMatrix();                                   //出栈
9          gl.glPopMatrix();                                   //出栈
10         gl.glPushMatrix();                                  //绘制身体
11         gl.glTranslatef(0, BODY_Y, 0);                      //移动
12         body.drawSelf(gl, android_texId);                   //绘制身体
13         gl.glPushMatrix();                                  //入栈
14         gl.glTranslatef(0, BODY_LENGTH/2, 0);               //移动
15         gl.glRotatef(-90, 1, 0, 0);                         //旋转
16         body_top.drawSelf(gl,android_texId);                //绘制身体上部
17         gl.glPopMatrix();                                   //出栈
18         gl.glPushMatrix();                                  //入栈
19         gl.glTranslatef(0, -BODY_LENGTH/8f, 0);             //移动
20         gl.glRotatef(180, 1, 0, 0);                         //旋转
21         body_bottom.drawSelf(gl, android_texId);            //绘制身体下部
22         gl.glPopMatrix();                                   //出栈
```

```
23          gl.glPopMatrix();                                          //出栈
24          ……//此处省略了部分机器人组件绘制的代码，读者可自行查看随书光盘的源代码
25          gl.glPushMatrix();                                         //绘制右腿
26          gl.glTranslatef((float)(BODY_RADIUS/2),                    //移动
27               (float) (LEG_Y-LEG_LENGTH/2*Math.cos(Math.toRadians (right_leg_degree))),
28                    (float)  (LEG_LENGTH/2*Math.sin(Math.toRadians(right_leg_degree))));
                                                                     //移动
29          gl.glRotatef(-right_leg_degree,1, 0, 0);                   //旋转
30          leg.drawSelf(gl, android_texId);                           //绘制
31          gl.glPushMatrix();                                         //入栈
32          gl.glTranslatef(0, LEG_LENGTH/2,0 );                       //移动
33          leg_terminal.drawSelf(gl, android_texId);                  //绘制
34          gl.glPopMatrix();                                          //出栈
35          gl.glPushMatrix();                                         //入栈
36          gl.glTranslatef(0, -LEG_LENGTH/2,0 );                      //移动
37          gl.glRotatef(180, 1, 0, 0);                                //旋转
38          leg_terminal.drawSelf(gl, android_texId);                  //绘制
39          gl.glPopMatrix();                                          //出栈
40          gl.glPopMatrix();                                          //出栈
41          gl.glPushMatrix();                                         //左天线
42          gl.glTranslatef(-(float)(2*BODY_RADIUS/3),
43               (float)(HEAD_Y+ANTENNA_LENGTH+ANTENNA_LENGTH/3), 0);//移动
44          gl.glRotatef(30, 0, 0, 1);                                 //旋转
45          antenna.drawSelf(gl, android_texId);                       //绘制
46          gl.glPopMatrix();
47          gl.glPushMatrix();                                         //右天线
48          gl.glTranslatef((float)(2*BODY_RADIUS/3),
49               (float)(HEAD_Y+ANTENNA_LENGTH+ANTENNA_LENGTH/3), 0);//移动
50          gl.glRotatef(-30, 0, 0, 1);                                //旋转
51          antenna.drawSelf(gl, android_texId);                       //绘制
52          gl.glPopMatrix();                                          //出栈
53     }
```

- 代码第 2-23 行分别为用半球绘制头的上半部分、用圆绘制头的下半部分、用圆绘制身体的上部、用圆柱绘制身体的中间部分和用半球绘制身体的下部。
- 代码第 24-53 行为绘制机器人的左右手臂，左右腿和左右两根天线。它们的组成部件是完全相同的，当然绘制也基本相同，都是中间绘制圆柱，两端绘制半球

（2）rotate 方法的实现，该方法实现了机器人的走路动画。将下列代码插入到 Robot 类的第 41 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Robot.java。

```
1     public  void rotate(final int robotAngle){
2          i=0;                                                  //记录机器人用多少步走一个格子
3          right_arm_degree=0;                                   //右手臂抬起的角度
4          left_arm_degree=0;                                    //左手臂抬起的角度
5          right_leg_degree=0;                                   //右腿抬起的角度
6          left_leg_degree=0;                                    //左腿臂抬起的角度
7          new Thread(){                                         //开启一个线程
8               public void run(){                               //启动线程
9                    while(i<100){                               //用 100 步走出 1.0 的距离
10                        flag=false;                            //动画运行时，按键不起作用
11                        switch(robotAngle){                    //判断机器人的朝向
12                        case 0:                                //z 轴正方向
13                             father.robotgroup.offsetz+=0.01;     //z 的大小增加
14                             break;
15                        case 180:                                   //z 轴负方向
16                             father.robotgroup.offsetz-=0.01;     //z 减小
```

```
17                              break;
18                          case 270:                                   //x 轴负方向
19                              father.robotgroup.offsetx-=0.01;        //x 减小
20                              break;
21                          case 90:                                    //x 轴正方向
22                              father.robotgroup.offsetx+=0.01;        //x 值增大
23                              break;
24                          }
25                          if(left_arm_degree>60){             //如果左手臂的抬起角度大于 60°
26                              count=-5.0001f;                 //变换 count 的值
27                          }
28                          else if(left_arm_degree<-60){       //如果右手臂的抬起角度小于 60°
29                              count=5.0001f;                  //变换 count 的值
30                          }
31                          right_arm_degree-=count;            //右手臂旋转角度变化
32                          left_arm_degree+=count;             //左手臂旋转角度变化
33                          right_leg_degree+=count;            //右腿旋转角度变化
34                          left_leg_degree-=count;             //左腿旋转角度变化
35                          if(i==99){
36                              right_arm_degree=0;             //右手臂旋转角清零
37                              left_arm_degree=0;              //左手臂旋转角清零
38                              right_leg_degree=0;             //右腿旋转角清零
39                              left_leg_degree=0;              //左腿旋转角清零
40                          }
41                          try{Thread.sleep(10);}              //线程休息 10 毫秒
42                          catch(InterruptedException e){e.printStackTrace();}
43                          i++;
44                      }
45                      flag=true;                              //动画结束时，按键才会起作用
46              }}.start();                                     //启动线程
47      }
```

- 代码第 2-6 行为机器人用多少步走一个格子的距离的帧数、以及机器人的左手臂、机器人的右手臂、机器人的左右腿的旋转角度归零。
- 代码第 7-24 行为开启一个线程，当机器人相对于坐标轴向 x 轴的正负方向、z 轴的正负方向运动时，x 与 z 值自增自减的变化。
- 代码第 25-46 行为当左手臂旋转角度最大值的绝对值大于 60° 后，变换旋转角度的增加或减少值 count，从而完成一个自然的机器人走路动画。

说明　机器人推箱子运动的动画与机器人走路动画基本相同，由于本书篇幅所限，这里不再赘述，请读者自行查看随书光盘中的源代码。

18.8.3　机器人组类 RobotGroup 类的实现

RobotGroup 类是扫描常量类中的地图数组，如果某个格子有机器人，则在此格子绘制机器人的类，并且此类中还有 3 个机器人转变方向的方法，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 RobotGroup.java。

```
1    package com.bn.pb;                                          //包的声明
2    import static com.bn.pb.Constant.*;                         //相关类的引入
3    import javax.microedition.khronos.opengles.GL10;            //相关类的引入
4    public class RobotGroup {                                   //表示立方体木箱组的类
```

```
5          MySurfaceView father;                                  //游戏主界面类的引用
6          Robot robot;                                           //立方体木箱
7          float yAngle=0;                                        //立方体木箱沿 Y 轴旋转的角度
8          float offsetx=0;                                       //x 与 z 的增量
9          float offsetz=0;
10         int i=0;                                               //控制动画的帧数
11         float positionx=0;                                     //机器人的位置
12         float positionz=0;                                     //机器人的位置
13         boolean flag=true;                                     //是否旋转的标记
14         int m=0;                                               //机器人的行和列
15         int n=0;
16         public RobotGroup(MySurfaceView father,int texId,int texId_head){
17              this.father=father;                               //为游戏主界面类的引用赋值
18              robot=new Robot(father,texId,texId_head);         //为机器人类的引用创建对象
19         }
20         public void drawSelf(GL10 gl){                         //扫描地图的每个格子
21              for(int i=0;i<father.rows;i++){
22                   for(int j=0;j<father.cols;j++){
23                        if(father.objectMap[i][j]==5){          //若此格有机器人则绘制
24                             m=i;  n=j;                         //得到机器人所在的行与列
25                             positionx=offsetx+(j+0.5f)*UNIT_SIZE-(int)//机器人位置
26                                  (Constant.MAP[Constant.COUNT][0].length/2)*UNIT_
                                    SIZE;
27                             positionz=offsetz+(i+0.5f)*UNIT_SIZE-(int) //机器人位置
28                                  (Constant.MAP[Constant.COUNT].length/2)*UNIT_SIZE;
29                        gl.glPushMatrix();                      //保护现场
30                        gl.glTranslatef(positionx, FLOOR_Y+CUBE_SIZE, positionz);
31                        gl.glRotatef(yAngle, 0, 1, 0);    //将机器人绕 Y 轴旋转
32                        robot.drawSelf(gl);               //绘制机器人
33                        gl.glPopMatrix();                 //恢复现场
34         }}}}
35         public void backRotate(final int robotAngle){}      //后转弯动画
36         public void leftRotate(final int robotAngle){}      //左转弯动画
37         public void rightRotate(final int robotAngle){}          //右转弯动画
38    }
```

- 代码第 1-15 行为包的声明、相关类的引入、机器人类的引用、游戏主界面类的引用以及机器人所在地图的行与列数、机器人的移动位置等相关成员变量的声明和赋值。
- 代码第 16-38 行分别为 RobotGroup 类的构造器、RobotGroup 类扫描，并且在相应位置绘制机器人的方法和机器人向左、向右和向后旋转的方法。

18.8.4 机器人组类成员方法的实现

上一小节中介绍了 RobotGroup 类的代码框架，本小节将对其中空实现的成员方法进行详细的介绍。下面按代码的开发顺序进行介绍。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 RobotGroup.java。

```
1    public void backRotate(final int robotAngle){        //机器人向后旋转的方法
```

```
2          i=0;                                          //机器人旋转 180° 所有的帧数
3          new Thread(){                                 //开启一个线程
4               public void run(){                       //重写 run 方法
5                    while(i<180){                       //用 100 步走出 1.0 的距离
6                         robot.flag=false;              //标志位设为 false
7                         try{Thread.sleep(10);}         //线程休息 10 毫秒
8                         catch(InterruptedException e){ //捕获异常
9                              e.printStackTrace();      //打印异常
10                        }
11                        yAngle+=1;                     //每次加 1° ,分 180 次转 180°
12                        i++;                           //i 的值增加
13                   }
14                   if(robotAngle==0){                  //机器人转变方向
15                        yAngle=180;
16                   }else if(robotAngle==90){           //机器人转变方向
17                        yAngle=270;
18                   }else if(robotAngle==180){          //机器人转变方向
19                        yAngle=0;
20                   }else if(robotAngle==270){          //机器人转变方向
21                        yAngle=90;
22                   }
23                   robot.flag=true;}                   //标志位设为 true
24         }.start();
25    }
```

- 代码第 2-13 行为开启一个线程，使机器人每隔 10 毫秒旋转 1°，用 1800 毫秒旋转 180°，从而实现了机器人向后转动的动画效果。
- 代码第 14-25 行为当调用机器人旋转的方法时，会传给此方法一个当前机器人相对与坐标系的角度，根据当前的角度，确认最后机器人相对坐标系的角度。

说明　leftRotate、rightRotate 方法与机器人向后旋转的方法 backRotate 基本相同，由于本书篇幅所限，这里不再赘述，请读者自行查看随书光盘中的源代码。

18.8.5　圆面 Circle 类的实现

Circle 类是构成机器人的一个重要的组件，机器人头的下部、身体的上部均是圆面构成的，下面对圆面的开发代码进行详细的介绍。Circle 类的代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Circle.java。

```
1    package com.bn.pb;                                          //包声明
2    import java.nio.ByteBuffer;                                 //相关类的引入
3    import java.nio.ByteOrder;                                  //相关类的引入
4    import java.nio.FloatBuffer;                                //相关类的引入
5    import javax.microedition.khronos.opengles.GL10;            //相关类的引入
6    public class Circle{                                        //用 triangle_fan 方式绘制圆面
7         private FloatBuffer   mVertexBuffer;                   //顶点坐标数据缓冲
8       private FloatBuffer   mTextureBuffer;                    //顶点纹理数据缓冲
9       private int vCount;                                      //顶点数
```

```
10      public Circle(
11              float angleSpan,                          //切分角度
12              float radius,                             //圆半径
13              float[] centerTexCoor,                    //纹理中心点
14              float textureRadius){                     //纹理半径
15        vCount=1+(int)(360/angleSpan)+1;//顶点的个数 //顶点纹理坐标数据的初始化——begin
16        float[] vertices=new float[vCount*3];         //初始化顶点数组
17        float[] textures=new float[vCount*2];         //初始化纹理坐标数组
18        vertices[0]=0;                                //存放中心点坐标
19        vertices[1]=0;
20        vertices[2]=0;
21        textures[0]=centerTexCoor[0];                 //存放中心点纹理
22        textures[1]=centerTexCoor[1];
23        int vcount=3;                                 //当前顶点坐标索引
24        int tcount=2;                                 //当前纹理坐标索引
25        for(float angle=0;angle<=360;angle=angle+angleSpan){
26              double angleRadian=Math.toRadians(angle);
27              vertices[vcount++]=radius*(float)Math.cos(angleRadian);     //顶点坐标
28              vertices[vcount++]=radius*(float)Math.sin(angleRadian);     //顶点坐标
29              vertices[vcount++]=0;                                       //顶点坐标
30              textures[tcount++]=centerTexCoor[0]+textureRadius*(float)Math.cos(angle-
                Radian);                                                    //纹理坐标
31              textures[tcount++]=centerTexCoor[1]+textureRadius*(float)Math.sin
                (angleRadian);
32              }
33          //vertices.length*4 是因为一个整数 4 个字节
34          ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
35          vbb.order(ByteOrder.nativeOrder());           //设置字节顺序
36          mVertexBuffer = vbb.asFloatBuffer();          //转换为 int 型缓冲
37          mVertexBuffer.put(vertices);                  //向缓冲区中放入顶点坐标数据
38          mVertexBuffer.position(0);                    //设置缓冲区起始位置
39          ByteBuffer tbb = ByteBuffer.allocateDirect(textures.length*4);
                                                          //创建顶点纹理数据缓冲
40          tbb.order(ByteOrder.nativeOrder());           //设置字节顺序
41          mTextureBuffer= tbb.asFloatBuffer();          //转换为 Float 型缓冲
42          mTextureBuffer.put(textures);                 //向缓冲区中放入顶点着色数据
43          mTextureBuffer.position(0);                   //设置缓冲区起始位置
44      }
45      public void drawSelf(GL10 gl,int textureId){ //绘制圆面
46          gl.glVertexPointer(                           //为画笔指定顶点坐标数据
47                  3,                                    //每个顶点的坐标数量为 3(xyz)
48                  GL10.GL_FLOAT,                        //顶点坐标值的类型为 GL_FIXED
49                  0,                                    //连续顶点坐标数据之间的间隔/跨度
50                  mVertexBuffer                         //顶点坐标数据
51          );
52          gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);
                                                          //为画笔指定纹理 ST 坐标缓冲
53          gl.glBindTexture(GL10.GL_TEXTURE_2D, textureId);   //绑定当前纹理
54          gl.glDrawArrays(                                   //绘制图形
55                  GL10.GL_TRIANGLE_FAN,                      //以 TRIANGLE_FAN 方式填充
56                  0,
57                  vCount);                                   //顶点数量
58    }}
```

- 代码第 1-44 行为分别为包声明、相关类的引入、相关成员变量的声明及赋值。第 10-44 行分别表示的是顶点坐标数据的初始化和顶点纹理数据的初始化。
- 代码第 45-58 行为绘制圆面方法的实现，表示的是为绘制圆面指定顶点数据、绑定纹理，并且用 TRIANGLE_FAN 填充方式绘制圆面。

说明　圆柱类 Cylinder、半球类 Globe 与圆面类 Circle 的开发及代码基本相同，由于本书篇幅所限，这里不再赘述，请读者自行查看随书光盘中的源代码。

18.8.6　矩形纹理类 TextureRect 的实现

在前面的小节中已经介绍过圆面类，矩形纹理类 TextureRect 相比于圆面类更加简单，但却是构成木箱、目标矩形以及虚拟按钮的重要组件，下面对其进行详细介绍，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 TextureRect.java。

```
1    package com.bn.pb;
2    import java.nio.ByteBuffer;                              //相关类的引入
3    import java.nio.ByteOrder;                               //相关类的引入
4    import java.nio.FloatBuffer;                             //相关类的引入
5    import javax.microedition.khronos.opengles.GL10;         //相关类的引入
6    import static com.bn.pb.Constant.*;                      //相关类的引入
7    public class TextureRect {                               //表示立方体某个面的纹理矩形
8          private FloatBuffer  mVertexBuffer;                //顶点坐标数据缓冲
9       private FloatBuffer mTextureBuffer;                   //顶点纹理数据缓冲
10      int vCount=0;                                         //顶点数目
11      int texId;                                            //纹理 id
12      public TextureRect(float width,float height,int texId){
13        this.texId=texId;                                   //纹理 id
14          vCount=6;                                         //顶点坐标数据的初始化
15        float vertices[]=new float[]{//
16            width*UNIT_SIZE,height*UNIT_SIZE,0,             //纹理矩形的 6 个顶点
17            -width*UNIT_SIZE,height*UNIT_SIZE,0,            //纹理矩形的 6 个顶点
18            -width*UNIT_SIZE,-height*UNIT_SIZE,0,           //纹理矩形的 6 个顶点
19            -width*UNIT_SIZE,-height*UNIT_SIZE,0,           //纹理矩形的 6 个顶点
20            width*UNIT_SIZE,-height*UNIT_SIZE,0,            //纹理矩形的 6 个顶点
21            width*UNIT_SIZE,height*UNIT_SIZE,0,             //纹理矩形的 6 个顶点
22        };
23        //vertices.length*4 是因为一个整数 4 个字节
24        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                              //创建顶点坐标数据缓冲
25        vbb.order(ByteOrder.nativeOrder());                 //设置字节顺序
26        mVertexBuffer = vbb.asFloatBuffer();                //转换为 Float 型缓冲
27        mVertexBuffer.put(vertices);                        //向缓冲区中放入顶点坐标数据
28        mVertexBuffer.position(0);                          //设置缓冲区起始位置
29        float[] texST={                                     //纹理坐标数据初始化
30            1,0,0,0,0,1,                                    //纹理矩形的 6 个点坐标
31            0,1,1,1,1,0                                     //纹理矩形的 6 个点坐标
32        };
33        ByteBuffer tbb = ByteBuffer.allocateDirect(texST.length*4);
34        tbb.order(ByteOrder.nativeOrder());                 //设置字节顺序
35        mTextureBuffer = tbb.asFloatBuffer();               //转换为 int 型缓冲
```

```
36          mTextureBuffer.put(texST);                          //向缓冲区中放入顶点着色数据
37          mTextureBuffer.position(0);                         //设置缓冲区起始位置
38      }
39      public void drawSelf(GL10 gl){
40          gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);  //启用顶点坐标数组
41          gl.glVertexPointer (                                //为画笔指定顶点坐标数据
42                  3,                                          //每个顶点的坐标数量为 3(xyz)
43                  GL10.GL_FLOAT,                              //顶点坐标值的类型为 GL_FIXED
44                  0,                                          //连续顶点坐标数据之间的间隔
45                  mVertexBuffer                               //顶点坐标数据
46          );
47          gl.glEnable(GL10.GL_TEXTURE_2D);                    //开启纹理
48          gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);//允许使用纹理 ST 坐标缓冲
49          gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);
                                                                //为画笔指定纹理 ST 坐标缓冲
50          gl.glBindTexture(GL10.GL_TEXTURE_2D, texId);   //绑定当前纹理
51          gl.glDrawArrays(                                    //绘制图形
52                  GL10.GL_TRIANGLES,                          //以三角形方式填充
53                  0,                                          //开始点编号
54                  vCount                                      //顶点的数量
55  );}}
```

- 代码第 1-11 行分别为包声明、相关类的引入、相关成员变量的声明及赋值。第 12-38 行分别表示的是顶点坐标数据的初始化，创建顶点坐标数据缓冲以及顶点纹理数据的初始化。
- 代码第 39-55 行为绘制纹理矩形方法的实现，表示的是启用顶点坐标数据，并为画笔指定顶点坐标数据、开启纹理并绑定当前纹理，以三角行方式填充纹理矩形。

18.8.7 纹理矩形组类 TextureRectGroup 的实现

TextureRectGroup 类的主要作用是赋值当前关卡的二维数组，并进行扫描，如果当前数组的值为 3，便在当前格子画红色目标纹理，下面将详细介绍其开发代码，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 TextureRectGroup.java。

```
1   package com.bn.pb;
2   import static com.bn.pb.Constant.*;                     //相关类的引入
3   import javax.microedition.khronos.opengles.GL10;        //相关类的引入
4   public class TextureRectGroup {                         //表示立方体木箱组的类
5       MySurfaceView father;                               //游戏主界面类的引用
6       TextureRect texturerect1;                           //目标纹理矩形
7       float yAngle=0;                                     //旋转角度
8       int objectMap[][];                                  //用于复制当前关的地图的二维数组
9       int rows=0;                                         //当前二维数组的行数
10      int cols=0;                                         //当前二维数组的列数
11      public TextureRectGroup(MySurfaceView father,int texId1){    //构造器
12          this.father=father;                             //主界面类的引用赋值
13          texturerect1=new TextureRect(CUBE_SIZE,CUBE_SIZE,texId1);
```

```
14              rows=Constant.MAP[Constant.COUNT].length;       //复制当前关的二维数组
15              cols=Constant.MAP[Constant.COUNT][0].length;    //复制当前关的二维数组
16              objectMap=new int[rows][cols];                  //为数组开辟空间
17              for(int i=0;i<rows;i++){                        //复制当前关的二维数组
18                   for(int j=0;j<cols;j++){                   //复制当前关的二维数组
19                        objectMap[i][j]=Constant.MAP[Constant.COUNT][i][j];
20         }}}
21         public void drawSelf(GL10 gl){                   //绘制目标矩形纹理
22              for(int i=0;i<rows;i++){                    //扫描地图，若此格有目标则绘制
23                   for(int j=0;j<cols;j++){
24                        if(objectMap[i][j]==3){           //绘制目标
25                             gl.glPushMatrix();           //保护现场
26                           gl.glTranslatef((j+0.5f)*UNIT_SIZE-(int) //移动到此格对应的位置
27                        (Constant.MAP[Constant.COUNT][0].length/2)*UNIT_SIZE,
28                                  FLOOR_Y+0.1f,
29                                  (i+0.5f)*UNIT_SIZE-(int)
30                        (Constant.MAP[Constant.COUNT].length/2)*UNIT_SIZE );
31                           gl.glRotatef(yAngle, 0, 1, 0);        //绕 Y 轴旋转
32                           gl.glRotatef(-90, 1, 0, 0);
33                           texturerect1.drawSelf(gl);            //绘制目标矩形
34                           gl.glPopMatrix();                     //恢复现场
35    }}}}}
```

- 代码第 1-10 行分别为包声明、相关类的引入、相关成员变量的声明及赋值。第 11-20 行构造器中根据当前的关数，获取二维数组的行与列，扫描地图并复制二维数组。
- 代码第 21-35 行为绘制所有目标矩形的方法，该方法的实现非常简单，只要遍历构造器中复制的二维数组，如果当前的数组值为 3，则在此格子处绘制目标矩形。

18.8.8　墙类 Wall 的实现

Wall 类的主要作用是根据当前关卡的地图二维数组，判断当前地图中每一个格子是否为可通过的格子，如果可以通过，则考虑此格子的上下左右是否需要围墙。首先介绍 Wall 类的代码框架，在游戏界面中绘制墙时，不仅仅是要画出墙来，更重要的是要节省资源，墙类的代码框架如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Wall.java。

```
1     package com.bn.pb;
2     import java.nio.ByteBuffer;                                  //相关类的引入
3     import java.nio.ByteOrder;                                   //相关类的引入
4     import java.nio.FloatBuffer;                                 //相关类的引入
5     import java.util.ArrayList;                                  //相关类的引入
6     import javax.microedition.khronos.opengles.GL10;             //相关类的引入
7     import static com.bn.pb.Constant.*;                          //相关类的引入
8     public class Wall {                                          //表示墙的类
9          private FloatBuffer   mVertexBuffer;                    //顶点坐标数据缓冲
10       private FloatBuffer   mTextureBuffer;                     //顶点纹理数据缓冲
11       private FloatBuffer   mNormalBuffer;                      //顶点法向量数据缓冲
12       int vCount;                                               //顶点数量
13       public Wall(){}                                           //构造器
14       public void drawSelf(GL10 gl,int texId){                  //绘制墙的方法
15          gl.glVertexPointer(                                    //为画笔指定顶点坐标数据
```

```
16                  3,                                          //每个顶点的坐标数量为 3(xyz)
17                  GL10.GL_FLOAT,                              //顶点坐标值的类型为 GL_FIXED
18                  0,                                          //连续顶点坐标数据之间的间隔
19                  mVertexBuffer                               //顶点坐标数据
20          );
21          gl.glNormalPointer(GL10.GL_FLOAT, 0, mNormalBuffer);//为画笔指定顶点法向量数据
22          gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);
                                                                //为画笔指定纹理 ST 坐标缓冲
23          gl.glBindTexture(GL10.GL_TEXTURE_2D, texId);        //绑定当前纹理
24          gl.glDrawArrays(                                    //绘制图形
25                  GL10.GL_TRIANGLES,                          //以三角形方式填充
26                  0,
27                  vCount
28   );}}
```

- 代码第 1-7 行分别为包声明、相关类的引入，第 8-12 行为墙类的顶点坐标数据缓冲、顶点纹理数据缓冲、顶点法向量数据缓冲以及顶点的数量。
- 代码第 13 行为墙类的构造器，第 14-28 行是执行绘制墙的方法，此方法中绘制了墙的顶部的墙以及最内侧的墙，绘制墙时，均是以三角形的方式填充。

说明 代码框架中的部分方法为空实现，这些方法的实现将在下面详细给出。

18.8.9 墙 Wall 类成员方法的实现

前面介绍了墙类 Wall 的代码框架，下面介绍 Wall 类的构造器，由于墙类的构造器代码较长，这里将通过几个部分进行详细介绍。

（1）如果地图的某一个格子可以通过，则需要判断这个格子的上一行、下一行、左边一列和右边一列是否需要墙，下面介绍判断上一行与下一行是否需要墙，将下列代码插入到 Wall 类的第 13 行代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Wall.java。

```
1     int rows=MAP[COUNT].length;                           //得到行数
2     int cols=MAP[COUNT][0].length;                        //得到列数
3     ArrayList<Float> alVertex=new ArrayList<Float>();     //顶点坐标数组
4     ArrayList<Float> alNormal=new ArrayList<Float>();     //纹理坐标数组
5     ArrayList<Float> alTexture=new ArrayList<Float>();    //顶点法向量
6     for(int i=0;i<rows;i++){
7        for(int j=0;j<cols;j++){                           //对地图中的每一块进行处理
8           if(MAP[COUNT][i][j]!=1){                        //若是可通过区域则要考虑围墙问题
9              if(i==0||MAP[COUNT][i-1][j]==1){             //若上面需要围墙
10                 float x1=j*UNIT_SIZE;                    //此矩形的 4 个顶点
11                 float y1=FLOOR_Y;                        //此矩形的 4 个顶点
12                 float z1=i*UNIT_SIZE;                    //此矩形的 4 个顶点
13                 float x2=j*UNIT_SIZE;                    //此矩形的 4 个顶点
14                 float y2=FLOOR_Y+WALL_HEIGHT;            //此矩形的 4 个顶点
15                 float z2=i*UNIT_SIZE;                    //此矩形的 4 个顶点
16                 float x3=(j+1)*UNIT_SIZE;                //此矩形的 4 个顶点
17                 float y3=FLOOR_Y+WALL_HEIGHT;            //此矩形的 4 个顶点
18                 float z3=i*UNIT_SIZE;                    //此矩形的 4 个顶点
19                 float x4=(j+1)*UNIT_SIZE;                //此矩形的 4 个顶点
20                 float y4=FLOOR_Y;                        //此矩形的 4 个顶点
21                 float z4=i*UNIT_SIZE;                    //此矩形的 4 个顶点
22                 alVertex.add(x1);alVertex.add(y1);alVertex.add(z1);
                                                            //第一个三角形的 3 个点放入数组
```

```
23                  alVertex.add(x3);alVertex.add(y3);alVertex.add(z3);
                                                        //第一个三角形的 3 个点放入数组
24                  alVertex.add(x2);alVertex.add(y2);alVertex.add(z2);
                                                        //第一个三角形的 3 个点放入数组
25                  alVertex.add(x1);alVertex.add(y1);alVertex.add(z1);
                                                        //第二个三角形的 3 个点放入数组
26                  alVertex.add(x4);alVertex.add(y4);alVertex.add(z4);
                                                        //第二个三角形的 3 个点放入数组
27                  alVertex.add(x3);alVertex.add(y3);alVertex.add(z3);
                                                        //第二个三角形的 3 个点放入数组
28                  alTexture.add(0f);alTexture.add(1f);  //第一个三角形的纹理
29                  alTexture.add(1f);alTexture.add(0f);  //第一个三角形的纹理
30                  alTexture.add(0f);alTexture.add(0f);  //第一个三角形的纹理
31                  alTexture.add(0f);alTexture.add(1f);  //第二个三角形的纹理
32                  alTexture.add(1f);alTexture.add(1f);  //第二个三角形的纹理
33                  alTexture.add(1f);alTexture.add(0f);  //第二个三角形的纹理
34                  alNormal.add(0f);alNormal.add(0f);alNormal.add(1f);
                                                        //第一个三角形的顶点法向量
35                  alNormal.add(0f);alNormal.add(0f);alNormal.add(1f);
                                                        //第一个三角形的顶点法向量
36                  alNormal.add(0f);alNormal.add(0f);alNormal.add(1f);
                                                        //第一个三角形的顶点法向量
37                  alNormal.add(0f);alNormal.add(0f);alNormal.add(1f);
                                                        //第二个三角形的顶点法向量
38                  alNormal.add(0f);alNormal.add(0f);alNormal.add(1f);
                                                        //第二个三角形的顶点法向量
39                  alNormal.add(0f);alNormal.add(0f);alNormal.add(1f);
                                                        //第二个三角形的顶点法向量
40              }
41              if(i==rows-1||MAP[COUNT][i+1][j]==1){     //若是此块下面需要围墙
42                  float x1=j*UNIT_SIZE;                 //此矩形的 4 个顶点
43                  float y1=FLOOR_Y;                     //此矩形的 4 个顶点
44                  float z1=(i+1)*UNIT_SIZE;             //此矩形的 4 个顶点
45                  float x2=j*UNIT_SIZE;                 //此矩形的 4 个顶点
46                  float y2=FLOOR_Y+WALL_HEIGHT;         //此矩形的 4 个顶点
47                  float z2=(i+1)*UNIT_SIZE;             //此矩形的 4 个顶点
48                  float x3=(j+1)*UNIT_SIZE;             //此矩形的 4 个顶点
49                  float y3=FLOOR_Y+WALL_HEIGHT;         //此矩形的 4 个顶点
50                  float z3=(i+1)*UNIT_SIZE;             //此矩形的 4 个顶点
51                  float x4=(j+1)*UNIT_SIZE;             //此矩形的 4 个顶点
52                  float y4=FLOOR_Y;                     //此矩形的 4 个顶点
53                  float z4=(i+1)*UNIT_SIZE;             //此矩形的 4 个顶点
54                  alVertex.add(x2);alVertex.add(y2);alVertex.add(z2);
                                                        //第一个三角形的 3 个点放入数组
55                  alVertex.add(x3);alVertex.add(y3);alVertex.add(z3);
                                                        //第一个三角形的 3 个点放入数组
56                  alVertex.add(x1);alVertex.add(y1);alVertex.add(z1);
                                                        //第一个三角形的 3 个点放入数组
57                  alVertex.add(x3);alVertex.add(y3);alVertex.add(z3);
                                                        //第二个三角形的 3 个点放入数组
58                  alVertex.add(x4);alVertex.add(y4);alVertex.add(z4);
                                                        //第二个三角形的 3 个点放入数组
59                  alVertex.add(x1);alVertex.add(y1);alVertex.add(z1);
                                                        //第二个三角形的 3 个点放入数组
60                  alTexture.add(0f);alTexture.add(0f);  //第一个三角形的纹理
61                  alTexture.add(1f);alTexture.add(0f);  //第一个三角形的纹理
62                  alTexture.add(0f);alTexture.add(1f);  //第一个三角形的纹理
63                  alTexture.add(1f);alTexture.add(0f);  //第二个三角形的纹理
64                  alTexture.add(1f);alTexture.add(1f);  //第二个三角形的纹理
65                  alTexture.add(0f);alTexture.add(1f);  //第二个三角形的纹理
66                  alNormal.add(0f);alNormal.add(0f);alNormal.add(-1f);
                                                        //第一个三角形的顶点法向量
```

```
67                  alNormal.add(0f);alNormal.add(0f);alNormal.add(-1f);
                                                        //第一个三角形的顶点法向量
68                  alNormal.add(0f);alNormal.add(0f);alNormal.add(-1f);
                                                        //第一个三角形的顶点法向量
69                  alNormal.add(0f);alNormal.add(0f);alNormal.add(-1f);
                                                        //第二个三角形的顶点法向量
70                  alNormal.add(0f);alNormal.add(0f);alNormal.add(-1f);
                                                        //第二个三角形的顶点法向量
71                  alNormal.add(0f);alNormal.add(0f);alNormal.add(-1f);
                                                        //第二个三角形的顶点法向量
72                  }
```

- 代码第 1-40 行为根据当前的关数，得到当前关的二维数组的行数和列数，由此扫描地图二维数组，如果地图的某一个格子可以通过，那么判断这个格子的上一行是否需要绘制墙面。
- 代码第 41-72 行为扫描地图二维数组，如果地图的某一个格子可以通过，那么判断这个格子的下一行是否需要绘制墙面，如果不可以通过，则绘制墙面。

（2）介绍完如何判断当前格子上面一行与下面一行是否需要绘制墙面后，下面将介绍，判断左边一列与右边一列是否需要绘制墙面，将下列代码插入到 Wall 类的第 13 行代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Wall.java。

```
1     if(j==0||MAP[COUNT][i][j-1]==1){                          //若是左面需要围墙
2          float x1=j*UNIT_SIZE;                                //此矩形的 4 个顶点
3          float y1=FLOOR_Y;                                    //此矩形的 4 个顶点
4          float z1=(i+1)*UNIT_SIZE;                            //此矩形的 4 个顶点
5          float x2=j*UNIT_SIZE;                                //此矩形的 4 个顶点
6          float y2=FLOOR_Y+WALL_HEIGHT;                        //此矩形的 4 个顶点
7          float z2=(i+1)*UNIT_SIZE;                            //此矩形的 4 个顶点
8          float x3=j*UNIT_SIZE;                                //此矩形的 4 个顶点
9          float y3=FLOOR_Y+WALL_HEIGHT;                        //此矩形的 4 个顶点
10         float z3=i*UNIT_SIZE;                                //此矩形的 4 个顶点
11         float x4=j*UNIT_SIZE;                                //此矩形的 4 个顶点
12         float y4=FLOOR_Y;                                    //此矩形的 4 个顶点
13         float z4=i*UNIT_SIZE;                                //此矩形的 4 个顶点
14         alVertex.add(x1);alVertex.add(y1);alVertex.add(z1);//第一个三角形的 3 个点放入数组
15         alVertex.add(x3);alVertex.add(y3);alVertex.add(z3);//第一个三角形的 3 个点放入数组
16         alVertex.add(x2);alVertex.add(y2);alVertex.add(z2);//第一个三角形的 3 个点放入数组
17         alVertex.add(x1);alVertex.add(y1);alVertex.add(z1);//第二个三角形的 3 个点放入数组
18         alVertex.add(x4);alVertex.add(y4);alVertex.add(z4);//第二个三角形的 3 个点放入数组
19         alVertex.add(x3);alVertex.add(y3);alVertex.add(z3);//第二个三角形的 3 个点放入数组
20         alTexture.add(0f);alTexture.add(1f);                 //第一个三角形的纹理
21         alTexture.add(1f);alTexture.add(0f);                 //第一个三角形的纹理
22         alTexture.add(0f);alTexture.add(0f);                 //第一个三角形的纹理
23         alTexture.add(0f);alTexture.add(1f);                 //第二个三角形的纹理
24         alTexture.add(1f);alTexture.add(1f);                   //第二个三角形的纹理
25         alTexture.add(1f);alTexture.add(0f);                   //第二个三角形的纹理
26         alNormal.add(1f);alNormal.add(0f);alNormal.add(0f); //第一个三角形的顶点法向量
27         alNormal.add(1f);alNormal.add(0f);alNormal.add(0f); //第一个三角形的顶点法向量
28         alNormal.add(1f);alNormal.add(0f);alNormal.add(0f); //第一个三角形的顶点法向量
29         alNormal.add(1f);alNormal.add(0f);alNormal.add(0f); //第二个三角形的顶点法向量
30         alNormal.add(1f);alNormal.add(0f);alNormal.add(0f); //第二个三角形的顶点法向量
31         alNormal.add(1f);alNormal.add(0f);alNormal.add(0f); //第二个三角形的顶点法向量
32    }
33    if(j==cols-1||MAP[COUNT][i][j+1]==1){
34         float x1=(j+1)*UNIT_SIZE;                            //此矩形的 4 个顶点
35         float y1=FLOOR_Y;                                    //此矩形的 4 个顶点
36         float z1=(i+1)*UNIT_SIZE;                            //此矩形的 4 个顶点
37         float x2=(j+1)*UNIT_SIZE;                            //此矩形的 4 个顶点
```

```
38          float y2=FLOOR_Y+WALL_HEIGHT;                          //此矩形的 4 个顶点
39          float z2=(i+1)*UNIT_SIZE;                              //此矩形的 4 个顶点
40          float x3=(j+1)*UNIT_SIZE;                              //此矩形的 4 个顶点
41          float y3=FLOOR_Y+WALL_HEIGHT;                          //此矩形的 4 个顶点
42          float z3=i*UNIT_SIZE;                                  //此矩形的 4 个顶点
43          float x4=(j+1)*UNIT_SIZE;                              //此矩形的 4 个顶点
44          float y4=FLOOR_Y;                                      //此矩形的 4 个顶点
45          float z4=i*UNIT_SIZE;                                  //此矩形的 4 个顶点
46          alVertex.add(x2);alVertex.add(y2);alVertex.add(z2);//第一个三角形的 3 个点放入数组
47          alVertex.add(x3);alVertex.add(y3);alVertex.add(z3);//第一个三角形的 3 个点放入数组
48          alVertex.add(x1);alVertex.add(y1);alVertex.add(z1);//第一个三角形的 3 个点放入数组
49          alVertex.add(x3);alVertex.add(y3);alVertex.add(z3);//第二个三角形的 3 个点放入数组
50          alVertex.add(x4);alVertex.add(y4);alVertex.add(z4);//第二个三角形的 3 个点放入数组
51          alVertex.add(x1);alVertex.add(y1);alVertex.add(z1);//第二个三角形的 3 个点放入数组
52          alTexture.add(0f);alTexture.add(0f);                   //第一个三角形的纹理
53          alTexture.add(1f);alTexture.add(0f);                   //第一个三角形的纹理
54          alTexture.add(0f);alTexture.add(1f);                   //第一个三角形的纹理
55          alTexture.add(1f);alTexture.add(0f);                   //第二个三角形的纹理
56          alTexture.add(1f);alTexture.add(1f);                   //第二个三角形的纹理
57          alTexture.add(0f);alTexture.add(1f);                   //第二个三角形的纹理
58          alNormal.add(-1f);alNormal.add(0f);alNormal.add(0f);//第一个三角形的顶点法向量
59          alNormal.add(-1f);alNormal.add(0f);alNormal.add(0f);//第一个三角形的顶点法向量
60          alNormal.add(-1f);alNormal.add(0f);alNormal.add(0f);//第一个三角形的顶点法向量
61          alNormal.add(-1f);alNormal.add(0f);alNormal.add(0f);//第二个三角形的顶点法向量
62          alNormal.add(-1f);alNormal.add(0f);alNormal.add(0f);//第二个三角形的顶点法向量
63          alNormal.add(-1f);alNormal.add(0f);alNormal.add(0f);//第二个三角形的顶点法向量
64      }}
```

- 代码第 1-32 行为扫描地图二维数组，如果地图的某一个格子可以通过，那么判断这个格子的左边一列是否需要绘制墙面，如果不可以通过，则绘制墙面。
- 代码第 33-64 行为扫描地图二维数组，如果地图的某一个格子可以通过，那么判断这个格子的右边一列是否需要绘制墙面，如果不可以通过，则绘制墙面。

（3）介绍完如何判断当前格子左边一列与右边一列是否需要绘制墙面后，下面将介绍判断某个格子的顶端是否需要绘制墙面，将下列代码插入到 Wall 类的第 13 行代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox /src/com/bn/pb 目录下的 Wall.java。

```
1       else if(MAP[COUNT][i][j]==1){                          //如果此处为墙
2           float xx1=j*UNIT_SIZE;                             //此矩形的 4 个顶点
3           float y=FLOOR_Y+WALL_HEIGHT;                       //此矩形的 4 个顶点
4           float zz1=i*UNIT_SIZE;                             //此矩形的 4 个顶点
5           float xx2=j*UNIT_SIZE;                             //此矩形的 4 个顶点
6           float zz2=(i+1)*UNIT_SIZE;                         //此矩形的 4 个顶点
7           float xx3=(j+1)* UNIT_SIZE;                        //此矩形的 4 个顶点
8           float zz3=(i+1)*UNIT_SIZE;                         //此矩形的 4 个顶点
9           float xx4=(j+1)*UNIT_SIZE;                         //此矩形的 4 个顶点
10          float zz4=i*UNIT_SIZE;                             //此矩形的 4 个顶点
11          alVertex.add(xx1);alVertex.add(y);alVertex.add(zz1);
                                                               //第一个三角形的 3 个点放入数组
12          alVertex.add(xx2);alVertex.add(y);alVertex.add(zz2);
                                                               //第一个三角形的 3 个点放入数组
13          alVertex.add(xx3);alVertex.add(y);alVertex.add(zz3);
                                                               //第一个三角形的 3 个点放入数组
14          alVertex.add(xx3);alVertex.add(y);alVertex.add(zz3);
                                                               //第二个三角形的 3 个点放入数组
15          alVertex.add(xx4);alVertex.add(y);alVertex.add(zz4);
                                                               //第二个三角形的 3 个点放入数组
16          alVertex.add(xx1);alVertex.add(y);alVertex.add(zz1);
```

```
                                                        //第二个三角形的 3 个点放入数组
17          alTexture.add(0f);alTexture.add(0f);        //第一个三角形的纹理
18          alTexture.add(0f);alTexture.add(1f);        //第一个三角形的纹理
19          alTexture.add(1f);alTexture.add(1f);        //第一个三角形的纹理
20          alTexture.add(1f);alTexture.add(1f);        //第二个三角形的纹理
21          alTexture.add(1f);alTexture.add(0f);        //第二个三角形的纹理
22          alTexture.add(0f);alTexture.add(0f);        //第二个三角形的纹理
23          alNormal.add(0f);alNormal.add(1f);alNormal.add(0f); //第一个三角形的顶点法向量
24          alNormal.add(0f);alNormal.add(1f);alNormal.add(0f); //第一个三角形的顶点法向量
25          alNormal.add(0f);alNormal.add(1f);alNormal.add(0f); //第一个三角形的顶点法向量
26          alNormal.add(0f);alNormal.add(1f);alNormal.add(0f); //第二个三角形的顶点法向量
27          alNormal.add(0f);alNormal.add(1f);alNormal.add(0f); //第二个三角形的顶点法向量
28          alNormal.add(0f);alNormal.add(1f);alNormal.add(0f); //第二个三角形的顶点法向量
29      }}}
30          vCount=alVertex.size()/3;                   //顶点数
31            float vertices[]=new float[alVertex.size()];
32            for(int i=0;i<alVertex.size();i++){       //扫描 ArrayList
33                vertices[i]=alVertex.get(i);          //扫描 ArrayList
34            }
35            ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
                                                        //创建顶点坐标数据缓冲
36            vbb.order(ByteOrder.nativeOrder());       //设置字节顺序
37            mVertexBuffer = vbb.asFloatBuffer();      //转换为 int 型缓冲
38            mVertexBuffer.put(vertices);              //向缓冲区中放入顶点坐标数据
39            mVertexBuffer.position(0);                //设置缓冲区起始位置
40            float normals[]=new float[vCount*3];      //顶点法向量数据初始化
41            for(int i=0;i<vCount*3;i++) {             //顶点法向量数据初始化
42                normals[i]=alNormal.get(i);           //顶点法向量数据初始化
43            }
44            ByteBuffer nbb = ByteBuffer.allocateDirect(normals.length*4);
45            nbb.order(ByteOrder.nativeOrder());       //设置字节顺序
46            mNormalBuffer = nbb.asFloatBuffer();      //转换为 int 型缓冲
47            mNormalBuffer.put(normals);               //向缓冲区中放入顶点着色数据
48            mNormalBuffer.position(0);                //设置缓冲区起始位置
49            float textures[]=new float[alTexture.size()]; //顶点纹理数据的初始化
50            for(int i=0;i<alTexture.size();i++) {     //顶点纹理数据的初始化
51                textures[i]=alTexture.get(i);         //顶点纹理数据的初始化
52            }
53            ByteBuffer tbb = ByteBuffer.allocateDirect(textures.length*4);
                                                        //创建顶点纹理数据缓冲
54            tbb.order(ByteOrder.nativeOrder());       //设置字节顺序
55            mTextureBuffer= tbb.asFloatBuffer();      //转换为 Float 型缓冲
56            mTextureBuffer.put(textures);             //向缓冲区中放入顶点着色数据
57            mTextureBuffer.position(0);               //设置缓冲区起始位置
58        }
```

- 代码第 1-29 行为判断地图二维数组的某一个值是否为 1，若为 1 则在该处顶端绘制墙面。
- 代码第 30-39 行为顶点法向量数据的初始化，第 40-48 行为顶点发向量数据的初始化，代码第 49-58 行为顶点纹理坐标数据的初始化。

说明　墙体的外围 WallOutSide 类与 Wall 类的开发及代码基本相同，由于本书篇幅所限，这里不再赘述，请读者自行查看随书光盘中的源代码。

18.8.10　箱子 Cube 类的实现

Cube 类是推箱子游戏中的箱子类，该类实现起来相对简单，主要是成员变量的赋值和箱子各个面的纹理贴图，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com/bn/pb 目录下的 Cube.java。

```
1    package com.bn.pb;                                          //声明包
2    import javax.microedition.khronos.opengles.GL10;            //引入相关类
3    import static com.bn.pb.Constant.*;                         //引入相关类
4    public class Cube {
5         TextureRect texture1;                                  //纹理矩形的前面
6         TextureRect texture2;                                  //纹理矩形的后面
7         TextureRect texture3;                                  //纹理矩形的上面
8         TextureRect texture4;                                  //纹理矩形的下面
9         TextureRect texture5;                                  //纹理矩形的左面
10        TextureRect texture6;                                  //纹理矩形的右面
11        float height;                                          //设定高
12        float width;                                           //设定宽
13        float length;                                          //设定长
14        float angleZ=0;                                        //绕 Z 轴旋转角度
15        float angleX=0;                                        //绕 X 轴旋转角度
16        float angleY=0;                                        //绕 Y 轴旋转角度
17        public Cube(float height,float length,float width,
18             int textureID1,int textureID2,int textureID3,int textureID4,int
               textureID5,int textureID6){
19             this.height=height;                                  //给高赋值
20             this.length=length;                                  //给长赋值
21             this.width=width;                                    //给宽赋值
22             texture1=new TextureRect(length,height,textureID1);//创建纹理矩形的前面
23             texture2=new TextureRect(length,height,textureID2);//创建纹理矩形的后面
24             texture3=new TextureRect(length,width,textureID3); //创建纹理矩形的上面
25             texture4=new TextureRect(length,width,textureID4); //创建纹理矩形的下面
26             texture5=new TextureRect(height,width,textureID5); //创建纹理矩形的左面
27             texture6=new TextureRect(height,width,textureID6); //创建纹理矩形的右面
28        }
29        public void drawSelf(GL10 gl){
30             gl.glPushMatrix();                                   //保护现场
31             gl.glRotatef(angleY, 0, 1, 0);                       //坐标系绕 Y 轴旋转角度
32             gl.glRotatef(angleZ, 0, 0, 1);                       //坐标系绕 Z 轴旋转角度
33             gl.glPushMatrix();                                   //绘制长方体前面
34             gl.glTranslatef(0, 0, UNIT_SIZE*width);              //移动坐标系
35             texture1.drawSelf(gl);                               //绘制纹理矩形的前面
36             gl.glPopMatrix();                                    //恢复现场
37             gl.glPushMatrix();                                   //绘制长方体后面
38             gl.glTranslatef(0, 0, -UNIT_SIZE*width);             //移动坐标系
39             gl.glRotatef(180, 0, 1, 0);                          //绕 Y 轴旋转 180°
40             texture2.drawSelf(gl);                               //绘制纹理矩形的后面
41             gl.glPopMatrix();                                    //恢复现场
42             gl.glPushMatrix();                                   //绘制纹理矩形的上面
43             gl.glTranslatef(0,UNIT_SIZE*height,0);               //移动坐标系
44             gl.glRotatef(-90, 1, 0, 0);                          //绕 X 轴旋转-90°
45             texture3.drawSelf(gl);                               //移动坐标系
46             gl.glPopMatrix();                                    //恢复现场
47             gl.glPushMatrix();                                   //绘制纹理矩形的左面
48             gl.glTranslatef(UNIT_SIZE*length,0,0);               //移动坐标系
49             gl.glRotatef(-90, 1, 0, 0);                          //绕 X 轴旋转-90°
50             gl.glRotatef(90, 0, 1, 0);                           //绕 Y 轴旋转 90°
51             texture5.drawSelf(gl);                               //绘制纹理矩形的左面
52             gl.glPopMatrix();                                    //恢复现场
53             gl.glPushMatrix();                                   //绘制纹理矩形的右面
54             gl.glTranslatef(-UNIT_SIZE*length,0,0);              //移动坐标系
55             gl.glRotatef(90, 1, 0, 0);                           //绕 X 轴旋转 90°
56             gl.glRotatef(-90, 0, 1, 0);                          //绕 Y 轴旋转-90°
57             texture6.drawSelf(gl);                               //绘制纹理矩形的右面
58             gl.glPopMatrix();                                    //恢复现场
59             gl.glPopMatrix();                                    //恢复现场
```

```
60    }}
```

- 代码第 5-16 行为箱子成员变量的定义，包括了箱子所有的属性，其中 texture1~ texture6 为箱体的 6 个面的纹理矩形，angleX~ angleZ 为箱体绕坐标系各轴的旋转角度。
- 代码第 17-28 行完成箱子的初始化，为箱子的长宽高赋值，并创建了箱子 6 个面的纹理矩形。
- 代码第 30-59 行实现了箱子 6 个面的纹理贴图，先保护现场，移动坐标系到合适的位置，然后旋转坐标系，绘制纹理矩形，最后恢复现场。

18.8.11 箱子组 CubeGroup 类的实现

CubeGroup 类是推箱子游戏中的箱子组的类，该类实现起来比较简单，主要是箱子对象的创建并根据地图在相应位置绘制普通箱子和红色箱子，其代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com/bn/pb 目录下的 CubeGroup.java。

```
1     package com.bn.pb;                                          //声明包
2     import static com.bn.pb.Constant.*;                         //引入相关包
3     import javax.microedition.khronos.opengles.GL10;            //引入相关包
4     public class CubeGroup {
5          MySurfaceView father;                                  //声明 MySurfaceView 的引用
6          Cube cube;                                             //声明箱子的引用
7          Cube cubered;                                          //声明红箱子的引用
8          float yAngle=0;/                                       /箱子沿 Y 轴旋转的角度
9          int redbincount=0;                                     //红箱子的数量
10         int bincount=0;                                        //普通箱子的数量
11         int targetcount=0;                                     //目标点的数量
12         public CubeGroup(MySurfaceView father,int texId,int texId2){
13              this.father=father;                               //得到 MySurfaceView 的引用
14              cube=new Cube(CUBE_SIZE,CUBE_SIZE,CUBE_SIZE,texId,texId,texId,texId,
                texId,texId);
15              cubered=new Cube(CUBE_SIZE,CUBE_SIZE,             //创建箱子对象
16                   CUBE_SIZE,texId2,texId2,texId2,texId2,texId2,texId2);
17         }
18         public void drawSelf(GL10 gl){
19              for(int i=0;i<father.rows;i++){                   //扫描地图的每个格子
20                   for(int j=0;j<father.cols;j++){
21                        if(father.objectMap[i][j]==4){          //如果是普通箱子
22                             gl.glPushMatrix();                 //保护现场
23                          gl.glTranslatef((j+0.5f)*UNIT_SIZE-(int)(father.objectMap
                            [0].length/2)*UNIT_SIZE,
24                                  FLOOR_Y+CUBE_SIZE,            //移动坐标系
25                                  (i+0.5f)*UNIT_SIZE-(int)(father.objectMap.length/
                                    2)*UNIT_SIZE );
26                        gl.glRotatef(yAngle, 0, 1, 0);                //将箱子绕 Y 轴旋转
27                        cube.drawSelf(gl);                            //绘制箱子
28                        gl.glPopMatrix();                             //恢复现场
29                        }else if(father.objectMap[i][j]==6){          //绘制红颜色的箱子
30                             gl.glPushMatrix();                       //保护现场
31                          gl.glTranslatef((j+0.5f)*UNIT_SIZE-(int)(father.object-
                            Map[0].length/2)*UNIT_SIZE,
32                                  FLOOR_Y+CUBE_SIZE,                  //移动坐标系
33                                  (i+0.5f)*UNIT_SIZE-(int)(father.objectMap.length/
                                    2)*UNIT_SIZE );
34                          gl.glRotatef(yAngle, 0, 1, 0);              //将箱子绕 Y 轴旋转
35                          cubered.drawSelf(gl);                       //绘制箱子
```

```
36                          gl.glPopMatrix();                    //恢复现场
37 }}}}}
```

- 代码第 5-16 行声明了相关类的引用，并在第 12-16 行的构造器中，创建了两个箱子对象，普通箱子和普通箱子被推到目标点上的红色箱子。
- 代码第 19-36 行遍历地图数组，判断是否绘制箱子，绘制过程，首先保护现场，其次移动坐标系到相应位置，并旋转坐标系，绘制箱子，最后恢复现场。

18.8.12　箱子移动 CubeGo 类的代码框架

CubeGo 类是当机器人推箱子时，箱子移动动画的实现，箱子可以上下左右直线移动，效果连续，且每次移动距离为单位长度。实现代码如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com/bn/pb 目录下的 CubeGo.java。

```
1     package com.bn.pb;                                    //声明包
2     import javax.microedition.khronos.opengles.GL10; //引入相关包
3     import static com.bn.pb.Constant.*;                   //引入相关包
4     public class CubeGo {
5          MySurfaceView father;                            //声明 MySurfaceView 的引用
6          TextureRect texture1;                            //纹理矩形的前面
7          TextureRect texture2;                            //纹理矩形的后面
8          TextureRect texture3;                            //纹理矩形的上面
9          TextureRect texture4;                            //纹理矩形的下面
10         TextureRect texture5;                            //纹理矩形的左面
11         TextureRect texture6;                            //纹理矩形的右面
12         float height;                                    //设置箱子高
13         float angleZ=0;                                  //绕 Z 轴旋转角度
14         float angleX=0;                                  //绕 X 轴旋转角度
15         float angleY=0;                                  //绕 Y 轴旋转角度
16         boolean flag=false;                              //是否绘制木箱的标志位
17         float offsetx=0;                                 //坐标位置增量
18         float offsetz=0;                                 //坐标位置增量
19         float positionx;                                 //坐标位置
20         float positionz;                                 //坐标位置
21         int m=0;                                         //行数
22         int n=0;                                         //列数
23         int i=0;                                         //控制箱子动画的变量
24    public CubeGo(MySurfaceView father,float height,int textureID1){  //构造器
25         this.father=father;                              //得到 MySurfaceView 的引用
26         this.height=height;                              //设置箱子的高
27         texture1=new TextureRect(length,height,textureID1);      //创建纹理矩形的前面
28         texture2=new TextureRect(length,height,textureID2);      //创建纹理矩形的后面
29         texture3=new TextureRect(length,width,textureID3);       //创建纹理矩形的上面
30         texture4=new TextureRect(length,width,textureID4);       //创建纹理矩形的下面
31         texture5=new TextureRect(height,width,textureID5);       //创建纹理矩形的左面
32         texture6=new TextureRect(height,width,textureID6);       //创建纹理矩形的右面
33    }
34    public void drawSelf(GL10 gl){                                //箱子绘制的方法
35    ……//此处省略了方法体中的代码，将在下一小节中给出
36    }
37    public  void cubeGo(final int robotAngle){                    //箱子运动动画
38    ……//此处省略了方法体中的代码，将在下一小节中给出
39    }}
```

- 代码第 5-23 行为该类成员变量的定义，包括了箱子在移动过程中所需的属性，其中 i 的含义为控制箱子动画的变量，offsetx 和 offsetz 为箱子坐标位置的增量。

- 代码第 34-39 行为在箱子移动过程中所用到的两个成员方法。即箱子自身绘制的方法和箱子移动动画实现的方法。

说明 代码框架中的部分方法为空实现，部分方法的实现将在下一小节中详细介绍。

18.8.13 箱子移动 CubeGo 类的成员方法的实现

上一小节中介绍了 CubeGo 类的代码框架，本小节将介绍其中空实现的成员方法。下面按代码的开发顺序进行介绍。

drawSelf 方法的实现。该方法依次为箱子贴 6 个面的纹理矩形，其代码如下。将下列代码插入到 CubeGo 类的第 34 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com/bn/pb 目录下的 CubeGo.java。

```
1    public void drawSelf(GL10 gl){
2        if(flag){                                          //通过标志位判断是否绘制
3            gl.glPushMatrix();                             //保护现场
4            positionx=offsetx+(n+0.5f)*UNIT_SIZE           //计算箱子的 x 坐标
5                -(int)(Constant.MAP[Constant.COUNT][0].length/2)*UNIT_SIZE;
6            positionz=offsetz+(m+0.5f)*UNIT_SIZE           //计算箱子的 y 坐标
7                -(int)(Constant.MAP[Constant.COUNT].length/2)*UNIT_SIZE;
8            gl.glTranslatef(positionx, FLOOR_Y+CUBE_SIZE, positionz);//移动坐标系
9            gl.glRotatef(angleY, 0, 1, 0);                 //绕 Y 轴旋转
10           gl.glRotatef(angleZ, 0, 0, 1);                 //绕 Z 轴旋转
11           gl.glPushMatrix();                             //绘制箱子前面
12           gl.glTranslatef(0, 0, UNIT_SIZE*width);        //移动坐标系
13           texture1.drawSelf(gl);                         //绘制纹理矩形的前面
14           gl.glPopMatrix();                              //恢复现场
15           gl.glPushMatrix();                             //绘制箱子后面
16           gl.glTranslatef(0, 0, -UNIT_SIZE*width);       //移动坐标系
17           gl.glRotatef(180, 0, 1, 0);                    //绕 Y 轴旋转 180°
18           texture2.drawSelf(gl);                         //绘制纹理矩形的后面
19           gl.glPopMatrix();                              //恢复现场
20           gl.glPushMatrix();                             //绘制箱子的上面
21           gl.glTranslatef(0,UNIT_SIZE*height,0);         //移动坐标系
22           gl.glRotatef(-90, 1, 0, 0);                    //绕 X 轴旋转-90°
23           texture3.drawSelf(gl);                         //移动坐标系
24           gl.glPopMatrix();                              //恢复现场
25           gl.glPushMatrix();                             //绘制箱子的下面
26           gl.glTranslatef(0,-UNIT_SIZE*height,0);        //移动坐标系
27           gl.glRotatef(90, 1, 0, 0);                     //绕 X 轴旋转 90°
28           texture4.drawSelf(gl);                         //绘制纹理矩形的下面
29           gl.glPopMatrix();                              //恢复现场
30           gl.glPushMatrix();                             //绘制纹理矩形的左面
31           gl.glTranslatef(UNIT_SIZE*length,0,0);         //移动坐标系
32           gl.glRotatef(-90, 1, 0, 0);                    //绕 X 轴旋转-90°
33           gl.glRotatef(90, 0, 1, 0);                     //绕 Y 轴旋转 90°
34           texture5.drawSelf(gl);                         //绘制纹理矩形的左面
35           gl.glPopMatrix();                              //恢复现场
36           gl.glPushMatrix();                             //绘制纹理矩形的右面
37           gl.glTranslatef(-UNIT_SIZE*length,0,0);        //移动坐标系
38           gl.glRotatef(90, 1, 0, 0);                     //绕 X 轴旋转 90°
39           gl.glRotatef(-90, 0, 1, 0);                    //绕 Y 轴旋转-90°
40           texture6.drawSelf(gl);                         //绘制纹理矩形的右面
41           gl.glPopMatrix();                              //恢复现场
42           gl.glPopMatrix();                              //恢复现场
43   }}
```

- 代码第 4-10 行保护现场后，计算箱子在空间中的位置，并将坐标系移动到该位置，然后对坐标系进行沿 y 轴和 z 轴的旋转，再进行下面的绘制。
- 代码第 11-41 行实现了箱子 6 个面的纹理贴图，首先保护现场并移动坐标系到合适的位置，然后旋转坐标系，绘制纹理矩形，最后恢复现场，完成绘制。

18.8.14　地板类 Floor 的实现

前面介绍了箱子相关类的实现，本小节将详细介绍地板类的实现，该类通过创建顶点坐标数据缓冲，顶点纹理数据缓冲和顶点向量数据缓冲来创建纹理矩形，然后在 MySurfaceView 中绘制出地板。由于此类篇幅较大，所以先介绍其框架，然后再详细介绍各部分功能。基本框架如下。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com/bn/pb 目录下的 Floor.java。

```
1    package com.bn.pb;                                          //声明包
2    import java.nio.ByteBuffer;                                 //引入相关包
3    import java.nio.ByteOrder;                                  //引入相关包
4    import java.nio.FloatBuffer;                                //引入相关包
5    import javax.microedition.khronos.opengles.GL10;            //引入相关包
6    import static com.bn.pb.Constant.*;                         //引入相关包
7    public class Floor {
8         private FloatBuffer mVertexBuffer;                     //顶点坐标数据缓冲
9         private FloatBuffer mTextureBuffer;                    //顶点纹理数据缓冲
10        private FloatBuffer mNormalBuffer;                     //顶点向量数据缓冲
11        int vCount=0;                                          //顶点数量
12        float yAngle;                                          //y 轴旋转角度
13        int width;                                             //地板横向 width 个单位
14        int height;                                            //地板纵向 height 个单位
15        int xOffset;                                           //坐标位置增量
16        int zOffset;                                           //坐标位置增量
17    public Floor(int xOffset,int zOffset,float scale,float yAngle,int width,int height) {
18        ……//此处省略了方法体中的代码，将在本小节后面给出
19   }
20   public void drawSelf(GL10 gl,int texId) {                   //地板的绘制方法
21        gl.glPushMatrix();                                     //保护现场
22        gl.glTranslatef(xOffset*UNIT_SIZE, 0, 0);              //移动坐标系
23        gl.glTranslatef(0, 0, zOffset*UNIT_SIZE);              //移动坐标系
24        gl.glRotatef(yAngle, 0, 1, 0);                         //旋转坐标系
25        gl.glVertexPointer(3,GL10.GL_FLOAT,0,mVertexBuffer);//为画笔指定顶点坐标
26        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);
27        gl.glBindTexture(GL10.GL_TEXTURE_2D, texId);           //绑定当前纹理
28        gl.glNormalPointer(GL10.GL_FLOAT, 0, mNormalBuffer);//启用向量数组指针
29        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, vCount);         //绘制图形
30        gl.glPopMatrix();                                      //恢复现场
31   }}
```

- 第 6-10 行声明绘制地板所需的顶点坐标数据缓冲，顶点纹理数据缓冲和顶点向量数据缓冲(绘制地板的时候使用了光照，所以要使用顶点向量数据缓冲)。
- 第 21-30 行为绘制地板的 drawSelf 方法。该方法主要是将顶点坐标数据缓冲，顶点纹理数据缓冲和顶点向量数据缓冲加载进相应的方法，并以三角形的方式进行绘制。

以上对地板类 Floor 的框架进行了简单的介绍，使读者对地板类有了整体的把握，接下来将对其构造器进行详细介绍，将下列代码插入到 Floor 类的 17 行。

代码位置：见随书光盘中源代码/第 18 章/3DPushBox/src/com/bn/pb 目录下的 Floor.java。

```
1    public Floor(int xOffset,int zOffset,float scale,float yAngle,int width,int height) {
2         this.yAngle=yAngle;                                    //地板绕 y 轴旋转角度
3         this.xOffset=xOffset;                                  //地板绕 x 轴旋转角度
4         this.zOffset=zOffset;                                  //地板绕 z 轴旋转角度
5         this.width=width;                                      //地板的宽
6         this.height=height;                                    //地板的高
7         vCount=width*height*6;                                 //每个地板块 6 个顶点
8         float vertices[]=new float[vCount*3];                  //每个顶点有 3 个坐标
9         int k=0;                                               //定义变量 k
10        for(int i=0;i<width;i++)
11        for(int j=0;j<height;j++){                             //每块地板由两个三角形构成
12                  vertices[k++]=i*UNIT_SIZE*scale;vertices[k++]=0;
13                  vertices[k++]=j*UNIT_SIZE*scale;vertices[k++]=i*UNIT_SIZE*scale;
14                  vertices[k++]=0; vertices[k++]=(j+1)*UNIT_SIZE*scale;
15                  vertices[k++]=(i+1)*UNIT_SIZE*scale;vertices[k++]=0;
16                  vertices[k++]=(j+1)*UNIT_SIZE*scale;vertices[k++]=(i+1)*UNIT_
                    SIZE*scale;
17                  vertices[k++]=0;vertices[k++]=(j+1)*UNIT_SIZE*scale;
18                  vertices[k++]=(i+1)*UNIT_SIZE*scale;vertices[k++]=0;
19                  vertices[k++]=j*UNIT_SIZE*scale;vertices[k++]=i*UNIT_SIZE*scale;
20                  vertices[k++]=0;vertices[k++]=j*UNIT_SIZE*scale;
21        };
22        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);//字节缓冲的大小
23        vbb.order(ByteOrder.nativeOrder());                    //设置字节顺序
24        mVertexBuffer = vbb.asFloatBuffer();                   //转换为 int 型缓冲
25        mVertexBuffer.put(vertices);                           //向缓冲区中放入顶点坐标数据
26        mVertexBuffer.position(0);                             //设置缓冲区起始位置
27        float normals[]=new float[vCount*3];                   //顶点向量数组
28        for(int i=0;i<vCount;i++){                             //为数组赋值
29             normals[i*3]=0;normals[i*3+1]=1;normals[i*3+2]=0;
30        }
31        ByteBuffer nbb = ByteBuffer.allocateDirect(normals.length*4); //字节缓冲的大小
32        nbb.order(ByteOrder.nativeOrder());                    //设置字节顺序
33        mNormalBuffer = nbb.asFloatBuffer();                   //转换为 int 型缓冲
34        mNormalBuffer.put(normals);                            //向缓冲区中放入顶点着色数据
35        mNormalBuffer.position(0);                             //设置缓冲区起始位置
36        float[] texST=new float[vCount*2];                     //纹理数组
37        for(int i=0;i<vCount*2/12;i++){                        //为数组赋值
38             texST[i*12]=0;texST[i*12+1]=0;texST[i*12+2]=0;
39             texST[i*12+3]=1;texST[i*12+4]=1;texST[i*12+5]=1;
40             texST[i*12+6]=1;texST[i*12+7]=1;texST[i*12+8]=1;
41             texST[i*12+9]=0;texST[i*12+10]=0;texST[i*12+11]=0;
42        };
43        ByteBuffer tbb = ByteBuffer.allocateDirect(texST.length*4);   //字节缓冲的大小
44        tbb.order(ByteOrder.nativeOrder());                    //设置字节顺序
45        mTextureBuffer = tbb.asFloatBuffer();                  //转换为 int 型缓冲
46        mTextureBuffer.put(texST);                             //向缓冲区中放入顶点着色数据
47        mTextureBuffer.position(0);                            //设置缓冲区起始位置
48   }
```

- 代码第 8-26 行为绘制地板所需顶点坐标数据。OpenGL ES 用三角形的方法绘制物体，因此，绘制地板的时候是用两个三角形组成一个矩形，然后将顶点坐标数组转换成顶点坐标数据缓冲。
- 代码第 27-35 行为每个顶点设置向量坐标。绘制地板的时候用了光照，所以，要为每个顶点设置向量坐标。一个顶点的向量值由 3 个坐标确定，这里设定每个顶点的向量方向都是向上的。
- 代码第 36-47 行为每个顶点设置纹理坐标。这里使用的是 S-T 纹理坐标方式，因此，每一

个顶点对应一对 S-T 坐标。

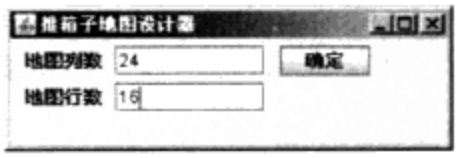
18.9 推箱子地图设计器的开发

推箱子这款游戏有多个关卡，因此使用地图设计器，可以大大方便游戏的开发。本节将对推箱子游戏的地图设计器进行简单介绍，这款地图设计器将会为后面的游戏开发提供地图信息数组。

提示　由于本设计器的开发相对简单，读者若有需要可以自行读光盘中所附的源代码，这里不再赘述。源代码位于光盘中“源代码/第 18 章/ 3DPushBoxMapSet/src/com/bn/frt”文件夹下。

地图设计的步骤如下。

（1）运行光盘中本章的地图设计器源代码，运行结果如图 18-22 所示。

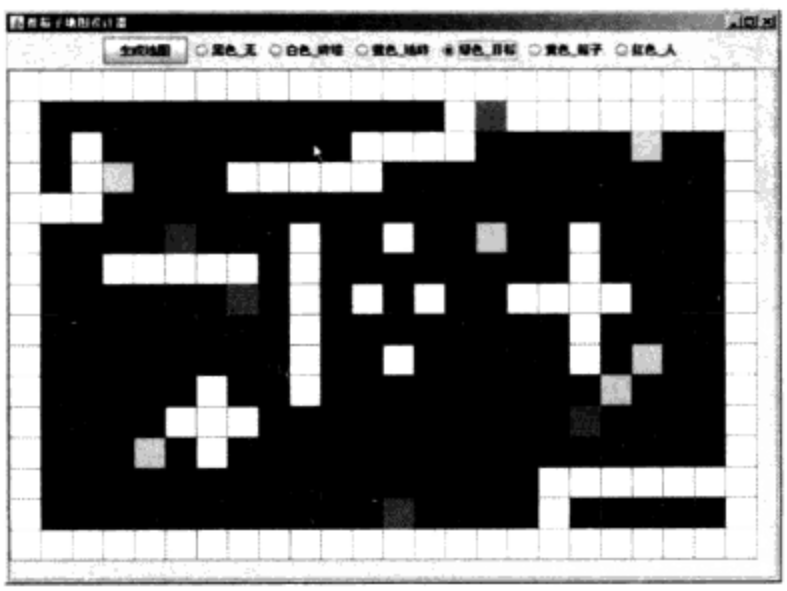


▲图 18-22　输入行列数的对话框

▲图 18-23　地图设计器初始界面

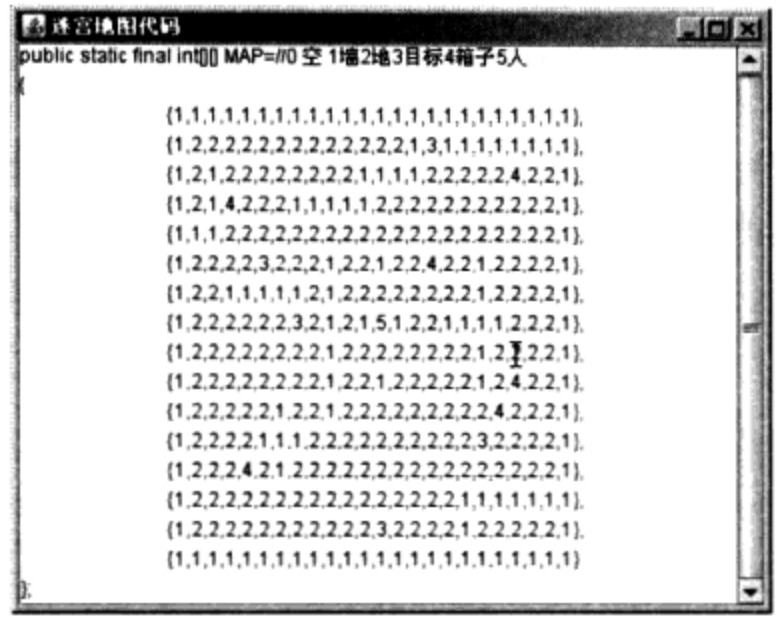
（2）在对话框中输入相应的行列数，单击确定，打开地图设计器，如图 18-23 所示。

（3）根据设计器上侧单选框提示，根据需求绘制地图，如图 18-24 所示。



▲图 18-24　地图设计器

（4）地图绘制完成后，单击生成地图，弹出地图数组，如图 18-25 所示。



▲图 18-25　用地图设计器自动成的代码

说明

读者可以按上述步骤自行运行光盘中该地图设计器，设计出自己想要的地图，将生成的地图代码复制到相应位置，然后运行游戏壁纸，以观察地图生成结果。

18.10 游戏的优化与改进

到此为止，本游戏的功能开发完毕，但是还有很多需要优化与改进的地方。下面提出可以改进的地方，有兴趣的读者可以对本游戏进行必要的提升。

- 游戏规则的完善

目前为止，游戏的开发虽然已经完成，但是，游戏规则并不完善，例如，本游戏将木箱推到目标点后，将不可以再次推动木箱，读者可以再开发一个推动红色木箱的动画，使本游戏的规则更加完善。

- 增加过关界面

当玩家通过某一关游戏后，游戏主界面将自动进入下一关游戏界面，读者可以根据本游戏中的主菜单界面或者是声音控制界面，在每两关之间设计一个过关界面，这样游戏将更加的美观。

- 地图的优化以及关数的增加

在本游戏中，只设计了两关，也就是说当玩家玩过两关之后，将进入游戏通关界面，读者可以利用地图设计器适当增加关数，并自行设置游戏的难易程度。

第 19 章　物理引擎游戏——盛怒的老鼠

《愤怒的小鸟》就如同前几年《植物大战僵尸》一样，一经发布便受到了广大游戏爱好者的好评。现在，该游戏不仅有 Android 版，还有 iPhone 版、PC 版、PSP 版，以及 PlayStation 版。

大多数游戏开发者在看到《愤怒的小鸟》时，肯定非常想知道该游戏的开发技术，为了充分满足读者的需求，笔者特意开发了一款《愤怒的小鸟》的山寨版本——《盛怒的老鼠》。看完本章后，读者会对《愤怒的小鸟》的开发技术有很好的了解。

19.1 游戏背景及功能概述

本节将从整体上对《盛怒的老鼠》游戏进行简单的介绍，首先介绍游戏的开发背景，然后对游戏的基本功能做详细的介绍，使读者了解本游戏的开发背景及基本玩法。

19.1.1　游戏开发背景

伴随着 Android 平台的逐步发展，基于该平台的游戏也备受人们的关注。Android 版《愤怒的小鸟》发布后，受到了广大游戏爱好者的喜爱，其也是当下 Android 平台下最受欢迎的游戏之一。而开发的《盛怒的老鼠》与《愤怒的小鸟》应用技术类似，也是休闲游戏的一种。

19.1.2　游戏功能概述

《盛怒的老鼠》游戏的规则是，游戏中每关只有一个鼠头，但是猫头的数量不等，用户必须使用这惟一的鼠头，消灭所有的猫头。如果猫头全部被消灭，则通过本关；否则，为通关失败。

（1）运行本游戏，首先进入的是游戏主界面，其运行效果如图 19-1 所示。

（2）在游戏的主界面点击“设置”按钮，弹出设置子选项。在设置子选项中可以打开或者关闭音效、查看游戏信息以及查看游戏最高分数，其运行效果如图 19-2 所示。

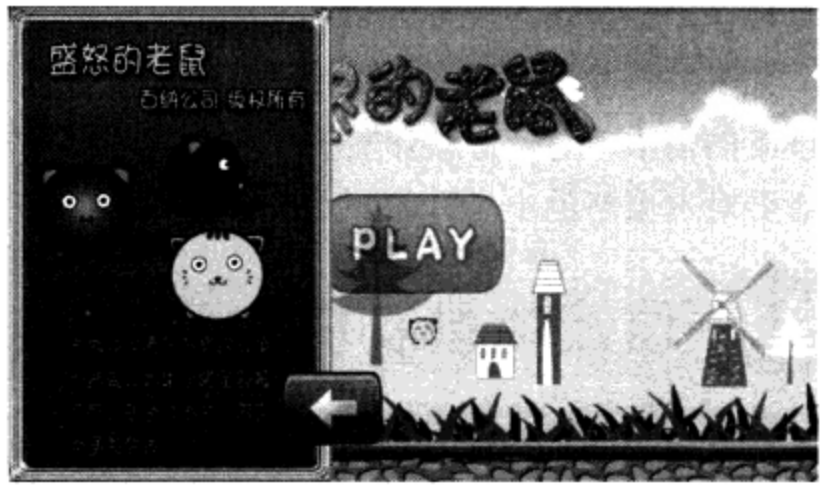
（3）在设置子选项中点击最上侧的“分数查看”按钮，可以查看每关的最高分，如图 19-3 所示。

（4）在设置子选项中点击中间的“信息查看”按钮，可以查看本游戏的相关信息，如图 19-4 所示。

▲图 19-1　游戏主界面

▲图 19-2　设置子选项

▲图 19-3　分数查看界面

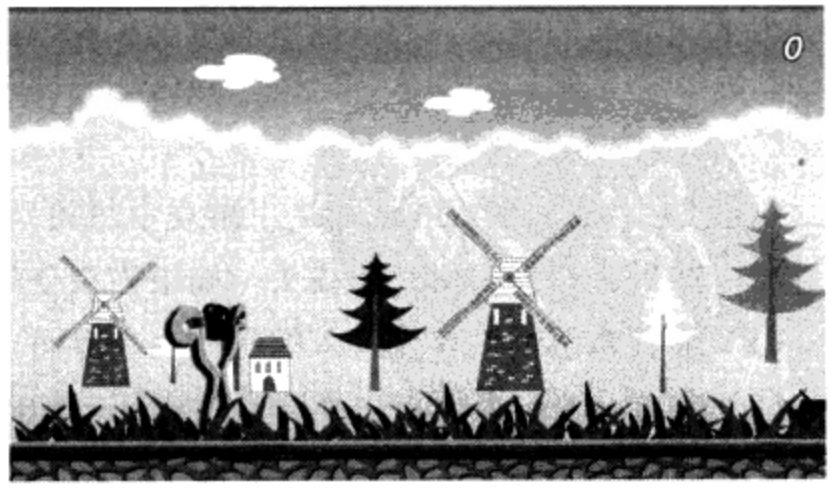


▲图 19-4　信息查看界面

（5）在设置子选项中点击下面的“声音”按钮，可以打开或者关闭声音，其运行效果如图 19-5 所示。

（6）在主界面点击“PLAY”按钮，可以进入游戏界面，其运行效果如图 19-6 所示。

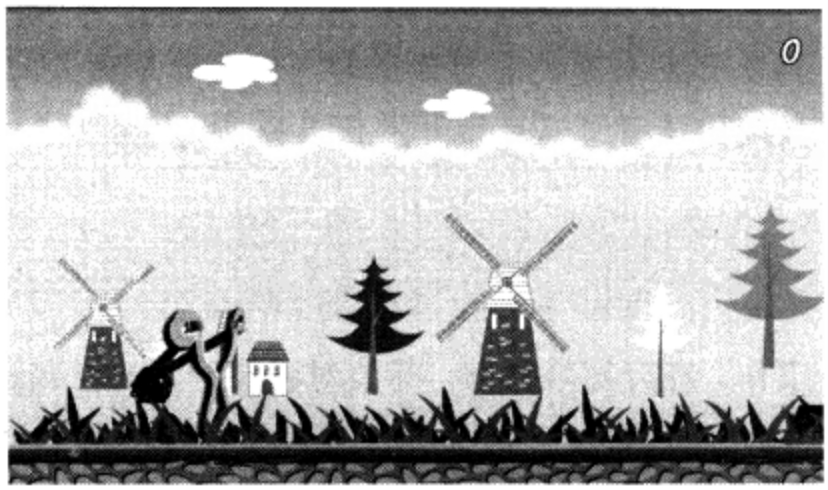
▲图 19-5　关闭或者打开声音



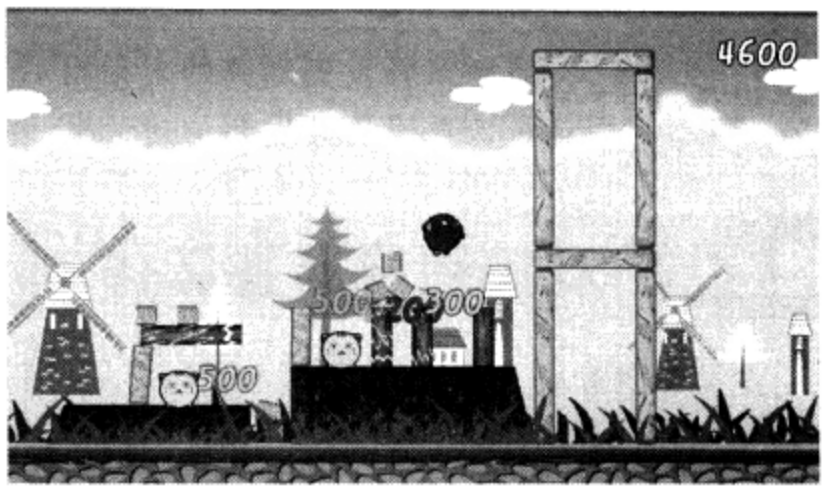
▲图 19-6　游戏界面

（7）按住鼠头并向后拖动，使得橡皮筋拉伸，其效果如图 19-7 所示。

（8）释放鼠头，本游戏的屏幕随着鼠头的移动而移动，鼠头碰撞到冰块、木条以及猫头后会显示对应的得分，第一关的运行效果如图 19-8 所示。



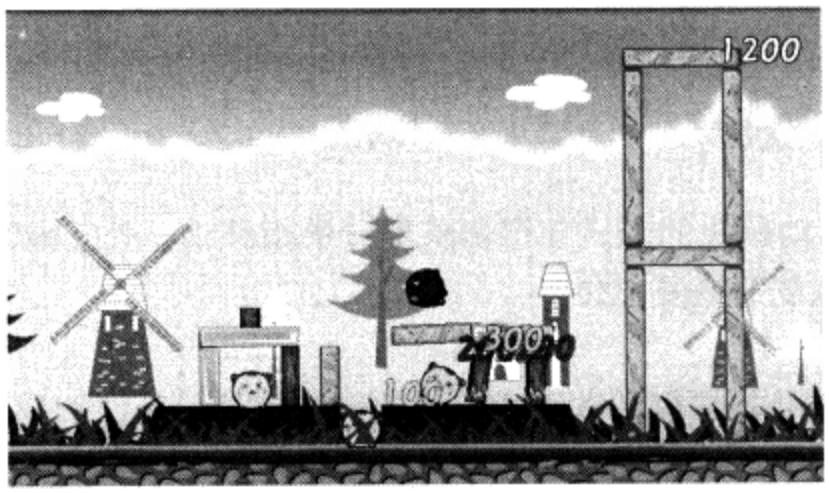
▲图 19-7　拖动鼠头向后



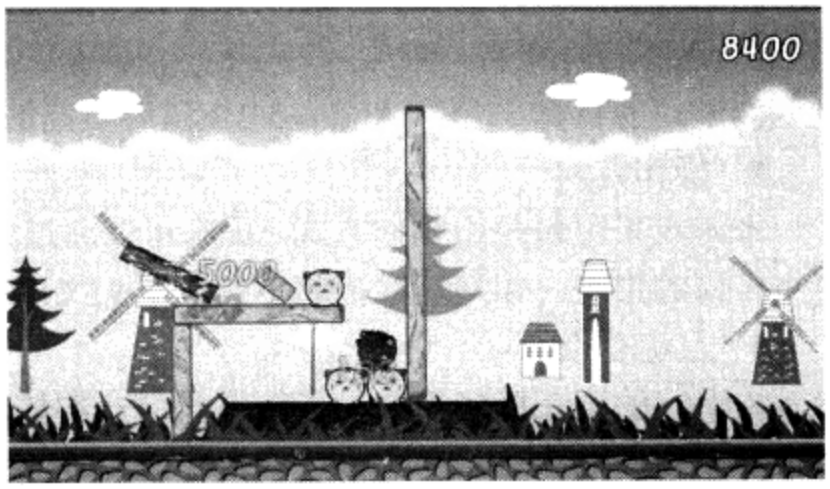
▲图 19-8　游戏第一关

（9）本游戏第二关有 3 个猫头，玩家必须用一个鼠头消灭 3 个猫头方可过关，其运行效果如图 19-9 所示。

（10）本游戏的第三关是由 4 个猫头组成，其运行效果如图 19-10 所示。



▲图 19-9　游戏第二关



▲图 19-10　游戏第三关

（11）如果鼠头将所有的猫头消灭，则为通过本关并弹出对应的对话框，其运行效果如图 19-11 所示。

（12）如果有猫头没有被消灭，则为通关失败，也弹出相应的对话框，其运行效果如图 19-12 所示。

▲图 19-11　通过本关

▲图 19-12　未通过本关

本游戏开发完通关的对话框后，再次开发未通关的对话框时，使用的是通关的对话框，并没有开发新的对话框，有兴趣的读者可以自行开发相关界面。

19.2 游戏策划及准备工作

本节主要介绍《盛怒的老鼠》的策划以及准备工作。在开发软件前，需要对将要开发的游戏进行策划、查找合适的图片以及音乐等。

19.2.1 游戏策划

由于本游戏只是一个案例，所以笔者对该步骤进行了适当的裁剪，但是如果开发真实项目，该步骤需要更细致、更全面以及更具体，本游戏的策划如下。

- 游戏类型以及呈现技术

本游戏属于单机的 2D 游戏，采用 2D 贴图技术，游戏画面美观，备受游戏爱好者的喜爱。

- 运行目标平台

本游戏的目标平台为 Android 2.1 或者更高的 Android 平台版本。

- 操作方式

本游戏采用触控的操作方式。玩家可以通过拖动鼠头的方式使得鼠头随着手指移动，橡皮筋也产生相应的形变，如果玩家抬起手指，鼠头飞出橡皮筋恢复原位。

- 目标受众

本游戏的玩法极其简单，任何手机用户群体均可非常容易掌握本游戏的玩法。

19.2.2 游戏开发的准备工作

在游戏开发前做好准备工作，可以使游戏开发顺利进行，起到事半功倍的效果。本游戏开发前的准备工作主要包括两部分。

- 由于本游戏为 2D 游戏，所以需要使用大量的图片。
- 为了增加游戏的可玩性，可以为本游戏添加适当的声音。

（1）首先介绍开发本游戏的主界面所用到的图片资源，图片以 png 类型为主，如表 19-1 所列。

表 19-1　　主界面图片清单

图 片 名	大小（KB）	像素（w×h）	用　途
about.png	5.3	320×480	游戏相关信息界面
close.png	4.98	100×75	关闭游戏相关信息按钮
fs_back.png	12.4	540×360	查看分数的对话框
play.png	7.01	180×100	开始按钮
set.png	3.77	80×80	设置按钮的背景
sndls.png	79.1	667×100	“盛怒的老鼠”文字

续表

图 片 名	大小（KB）	像素（w×h）	用　　途
set_back.png	4.23	70×290	设置按钮的半透明背景
set_ck.png	9.60	70×70	分数查看按钮
set_cl.png	1.82	80×80	设置按钮表面的齿轮
set_close.png	8.16	70×70	关闭音量按钮
set_sy.png	10.1	70×70	声音按钮
set_xx.png	13.7	70×70	游戏信息查看按钮

说明　res 文件夹存放所有非代码资源，其中 drawable 文件夹存放的是图片，而本游戏中与主界面相关的图片全部存放在 drawable-nodpi 文件夹中。

（2）接下来介绍开发本游戏的游戏界面时用到的图片资源，如表 19-2 所列。

表 19-2　　游戏界面图片清单

图 片 名	大小（KB）	像素（w×h）	用　　途
bg.png	69.4	1000×480	游戏背景
bottom_left_ntz.png	22.3	1000×80	左侧的灰泥台子
bottom_ntz.png	41.1	1000×80	灰泥台子
bottom_right_ntz.png	22.3	1000×80	右侧的灰泥台子
c_dd.png	26.5	1000×60	距离背景较近的草
c_xd.png	44.8	1000×96	距离背景较远的草
dg.png	31.6	122×100	游戏中用到的弹弓
dhk.png	30.0	540×360	输赢弹出的对话框
gk_1～3.png	0.8	300×100	关卡一至关卡三
ground.png	26.8	2000×40	地面
ice_hen.png	4.06	100×20	横着未破碎冰块
ice_hen_1～3.png	6.40	100×20	横着破碎的冰块一至冰块三
ice_shu.png	4.13	20×100	竖直未破碎冰块
ice_shu_1～3.png	6.40	20×100	竖直破碎的冰块一至冰块三
level_cleared.png	0.60	250×40	过关后对话框中的提示
level_failure.png	0.60	250×40	未过关对话框中的提示
ls_zy.png	4.78	40×40	鼠头
num_bs_0~9.png	3.27	20×40	白色数字 0~9
num_hs_0~9.png	3.30	20×40	黄色数字 0~9
num_zs_0~9.png	3.55	20×40	紫色数字 0~9

续表

图 片 名	大小（KB）	像素（w×h）	用　　途
stone_hen.png	5.14	100×20	横着的石条
stone_shu.png	5.26	20×100	竖直的石条
v_xyg.png	5.37	100×100	去下一关按钮
v_zjm.png	3.54	100×100	去主菜单按钮
wood_hen.png	5.22	100×20	横着未破碎的木条
wood_hen_1～2.png	6.09	100×20	横着破碎的木条一至木条二
wood_shu.png	5.18	100×20	竖直未破碎的木条
wood_shu_1～2.png	6.12	20×100	竖直破碎的木条一至木条二
xm.png	4.49	40×40	猫头

说明　表 19-2 为开发本游戏界面时用到的图片资源，而本游戏中与游戏界面相关的图片全部存放在 drawable-nodpi 文件夹中。

（3）本游戏中的拉动橡皮筋、鼠头飞出、碰撞冰块、碰撞木条以及碰撞石条均会发出声音，所有用到的声音列表如 19-3 所列。

表 19-3　　声音资源

声音文件名	大小（KB）	格　　式	用　　途
boli.mp3	5.00	mp3	冰块破碎的声音
cat.mp3	2.65	mp3	猫头被打死的声音
ls_fx.mp3	4.71	mp3	鼠头飞出的声音
pijin.mp3	56.2	mp3	皮筋拉伸的声音
stond.mp3	1.29	mp3	碰撞石条的声音
wood.mp3	3.50	mp3	碰撞木条的声音

提示　res 文件夹存放所有非代码资源，而游戏中的声音资源全部存放在 res/raw 文件夹中。

（4）本游戏所用到的物理引擎是开源的 JBox2D，而《愤怒的小鸟》正是使用的该物理引擎。因为本游戏使用的物理引擎是 JBox2D，所以需要在项目中添加相应的支持 jar 包“jbox2d-2.0.1-library-only.jar”。没有 jar 包的读者可以到 JBox2D 的官方网站“www.jbox2d.org”下载并解压，通过 Eclipse 加载即可。

19.3 游戏的框架

在介绍本游戏的具体代码之前，首先介绍的是本游戏的整体框架，让读者在整体上了解本游戏，

这非常有利于读者学习下面具体的代码。

19.3.1　游戏主要用到的技术

本游戏为 2D 游戏，主控制类继承自 Android 平台上的 Activity。所有的界面均是继承自 SurfaceView，并实现了 SurfaceHolder.Callback 接口。由于界面需要更新，所以开启一个线程，每隔一段固定的时间调用自定义的 repaint 方法重新绘制界面。

本游戏需要存储游戏的分数，若使用 Android 平台自带的 SQLite 数据库，则显得小题大做。所以在存储这些分数时，使用的是 Android 平台提供的轻量级的数据存储机制——Preferences。

19.3.2　游戏各个类的介绍

为了让读者可以更好地理解本游戏中各个类的作用，下面将本游戏中的所有类分成 5 类进行介绍，所有类的代码，将在后面给予详细的介绍。

1. 共有类

- 主控制类 MyBox2dActivity

此类继承自 Activity 类，是整个游戏的控制类，在游戏开始时调用。负责控制主界面、游戏界面的跳转以及游戏中用到的数据的初始化。

- 常量类 Constant

此类是游戏中的常量类，存储游戏中贴图的位置信息，通过创建 PicLoadUtil 的对象加载图片，并调用 PicLoadUtil 类中的 scaleToFitXYRatio 方法缩放图片。

2. 刚体相关类

- 多边形类 MyPolygonImg

此类是 BodyCat、BodyIce 以及 BodyWood 的父类，该类主要是创建自定义的绘制方法 drawSelf 以及创建碰撞后执行对应动作的方法 doAction。

- 猫头类 BodyCat

此类继承自 MyPolygonImg，并重写了 doAction 以及 drawSelf 方法。drawSelf 方法根据不同的域值，对不同状态下的猫头进行绘制。

- 冰块类 BodyIce

此类也是继承自 MyPolygonImg，并重写了 doAction 以及 drawSelf 方法，drawSelf 方法也是根据域值的值，绘制不同状态下的冰块。

- 木条类 BodyWood

此类是木条类，也是继承自 MyPolygonImg，该类重写了 doAction 方法，在绘制木条时调用的是其父类 MyPolygonImg 中的 drawSelf 方法。

- 刚体类型声明类 BodyType

此类主要是声明在游戏场景中存在的刚体的类型，主要类型有 LS_ZY（睁眼鼠头）、DM（地面）、ST（石条）、MT（木条）、XM（猫头）、BK（冰块）以及 HNTZ（灰泥台子）。

- 查询类 BodySearchUtil

此类主要是判断鼠头是否被创建，并判断记分标志位 START_SCORE 是否为 true，如果为 true，则根据碰撞物体的不同执行不同的动作。

3. 工具类

- 加载及缩放图片的工具类 PicLoadUtil

此类的主要作用是通过 BitmapFactory.decodeResource 方法加载图片，通过 Matrix（矩阵）实现图片的缩放，使得图片符合目标机型的屏幕分辨率。

- 生成刚体工具类 Box2DUtil

此类中主要是创建 createPolygonImg 方法，在该方法中创建多边形描述对象，并设置多边形的具体属性，最后在世界中创建刚体对象，并根据多边形的不同类型返回 MyPolygonImg 类的对象。

- 声音工具类 SoundUtil

此类主要是创建 initSounds 方法对声音缓冲池进行初始化，同时创建 playSound 方法，该方法主要是在播放声音时调用。

4. 主界面相关类

- 主界面 MainMenuView

此类表示的是本游戏的主界面，继承自 SurfaceView，并实现了 SurfaceHolder.Callback 接口。该类中的 onDraw 方法为绘制界面中的各个部件，重写 onTouchEvent 方法为该界面中的各个部件添加监听。

- 刷帧线程 MainMenuDrawThread

此类的主要作用是定时调用 repaint 方法为主界面 MainMenuView 刷帧，并定时更改草地及背景的位置，使得草地、背景以一定的速度向左移动。

- 鼠头和猫头类 Taj

此类根据鼠头或者猫头初速度的不同，调用 Matrix 类中的 setScale 方法更改图片的大小，并调用 drawBitmap 方法绘制图形。

- 鼠头和猫头控制类 TJThread

此类继承自 Thread，主要作用是随机生成鼠头或者猫头的位置以及速度，创建 Taj 对象，并将创建的 Taj 对象添加到 List<Taj>集合中。

5. 游戏界面相关类

- 游戏界面类 GameView

此类是本游戏中的场景，继承自 SurfaceView，并实现了 SurfaceHolder.Callback 接口。该类主要是调用 loadGameData 方法加载游戏数据，调用 JBox2D 中的 initContactListener 方法加载刚体的碰撞监听器，调用 onDraw 方法绘制游戏的具体场景，并调用 onTouchEvent 为游戏添加触控监听。

- 刷帧线程 DrawThread

此类继承自 Thread，主要是每隔固定的时间调用 repaint 方法刷帧，使得游戏时时更新，根据

flag 的值判断是否绘制过关对话框，并判断是否过关。创建 doAddtask 方法，将鼠头刚体添加进 ArrayList<MyPolygonImg>集合中。

- 皮筋类 Pijin

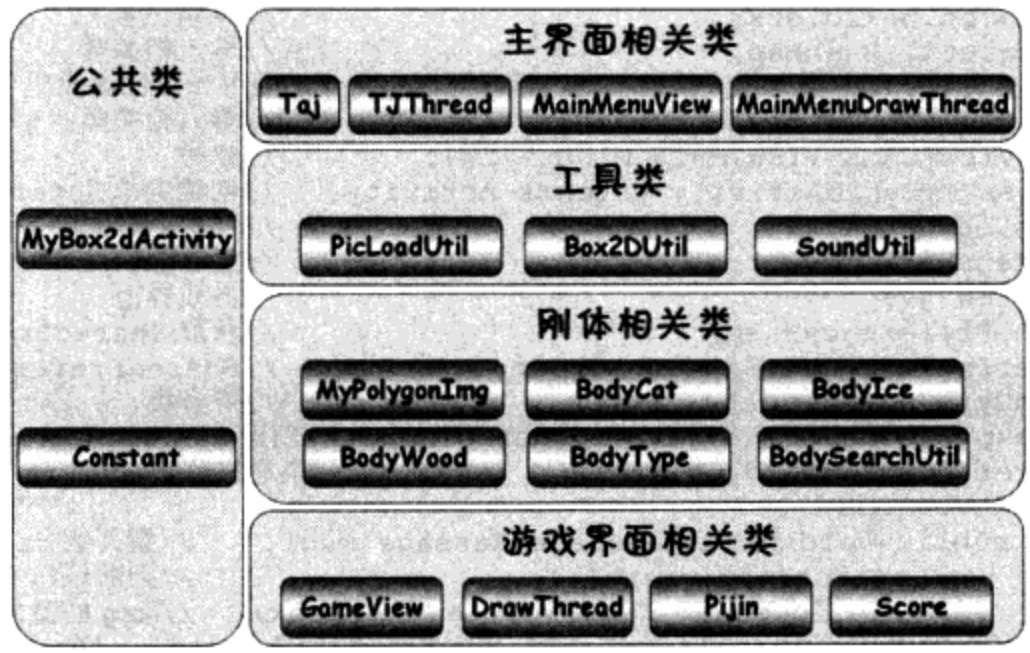
此类是弹弓上的皮筋类，该类的主要作用是根据手指拖动鼠头的位置，动态地改变皮筋的长度，创建自定义的 drawSelf 方法绘制橡皮筋。

- 记录分数类 Score

此类是游戏的得分类，根据鼠头碰撞各个刚体所得的分数在 GameView 中的 onDraw 方法绘制得分。

19.3.3　游戏的基本框架

上一小节已经对游戏中的各个类进行了介绍，接下来将用一幅图来介绍各个类之间的关系，相信读者在看完本游戏的框架图后会有进一步的理解。游戏的框架如图 19-13 所示。



▲图 19-13　游戏框架

说明　图 19-13 为游戏的基本框架图，分为公共类、主界面相关类、工具类、刚体相关类以及游戏界面相关类 5 部分。其中刚体相关类是本游戏中的实体类，主界面相关类是本游戏的主界面，游戏界面相关类指的是游戏界面。

（1）运行本游戏，首先调用主控制类 MyBox2dActivity 中的 onCreate 方法，在该方法中设置屏幕为全屏并且横屏，然后获得屏幕尺寸以及初始化游戏中用到的图片数据，最后跳转到主界面 MainMenuView。

（2）主界面中飞出的鼠头或者猫头使用 Taj 和 TJThread 控制的，在主界面点击“设置”按钮弹出游戏的设置子选项，点击“PLAY”按钮可以跳转到游戏界面 GameView。

（3）游戏界面的橡皮筋用 Pijin 类控制，该界面的各个实体通过刚体相关类中自定义的 drawSelf 方法绘制，并且通过线程类 DrawThread 定时调用 repaint 方法更新界面。

19.4 公共类

前一节介绍的是本游戏的框架，从本节开始将会对本游戏中的各个类进行详细的介绍。首先介绍本游戏的公共类，主要包括主控制类 MyBox2dActivity 和常量类 Constant。

19.4.1 主控制类——MyBox2dActivity

本小节介绍本游戏的主控制类 MyBox2dActivity，该类继承自 Android 平台的 Activity，在游戏开始运行时调用。该类中主要是设置了游戏为横屏全屏显示，同时设置了键盘监听事件。

（1）首先介绍主控制类 MyBox2dActivity 的框架，了解该框架有助于读者对该类整体结构的理解，该类的框架代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MyBox2dActivity.java。

```
1    package com.bn.box2d.szxn;                          //声明包名
2    import java.util.HashMap;                           //导入相关类
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.szxn.Constant.*;         //导入相关类
5    enum WhichView{GAME_VIEW,MAIN_MENU_VIEW};           //枚举
6    public class MyBox2dActivity extends Activity{      //创建继承 Activity 的类
7         WhichView curr;                                //当前的界面
8         MainMenuView mmv;                              //主界面
9         GameView gv;                                   //游戏界面
10        SharedPreferences sp;                          //获取 SharedPreferences
11        SharedPreferences.Editor editor;               //SharedPreferences 中写回数据
12        SoundUtil soundutil;                           //声音类
13        HashMap<Integer, Integer> soundPoolMap; //创建 HashMap<Integer, Integer>引用
14        Handler hd=new Handler(){                  //消息接收者
15             @Override
16             public void handleMessage(Message msg){    //重写的 handleMessage 方法
17                  switch(msg.what){                     //判断 msg 的值
18                       case 0: gotoGameView();break;    //msg 的值为 0，则跳转到游戏界面
19                       case 1: gotoMainMenuView(); break;   //msg 的值为 1，则跳转到
                                                              //主菜单界面
20        } } };
21        public void onCreate(Bundle savedInstanceState){     //继承 Activity 需重写的
                                                               //onCreate 方法
22         super.onCreate(savedInstanceState);                 //调用父类
23         ......//该处省略了部分代码，读者可以自行查阅随书光盘中的源代码
24         gotoMainMenuView();                                 //进入主菜单界面
25        }
26        ......//该处省略了跳转到主菜单界面与游戏界面的代码，将在后面给出
27        @Override
28        public boolean onKeyDown(int keyCode,KeyEvent e){    //重写的 onKeyDown 方法
29        if(keyCode==4){                                      //点击的是返回按钮
30             if(curr==WhichView.MAIN_MENU_VIEW){             //如果当前界面为主菜单界面
31                  sp=this.getSharedPreferences("actm", Context.MODE_PRIVATE);
32                  SharedPreferences.Editor editor=sp.edit(); //获得 SharedPreferences.
                                                               //Editor 对象
33                  String str=HH_SCORE[0]+"|"+HH_SCORE[1]+"|"+HH_SCORE[2];
34                  editor.putString("score", str);  //添加进 Preferences 数据库中
35                  editor.commit();                 //提交
36                  System.exit(0);                  //退出虚拟机
37             }
38             if(curr==WhichView.GAME_VIEW){        //当前界面为游戏界面
```

```
39                  DRAW_THREAD_FLAG=false;          //设置线程标志位为 false
40                  gotoMainMenuView();              //跳转到主菜单界面
41          } }
42          return true;                             //返回 true
43    } }
```

● 第 5 行为枚举，其主要作用是存储当前所在的界面。在跳转页面时，主要是判断当前页面的 curr 值，然后进行跳转。

● 第 7～13 行是本类的成员变量，有 WhichView（当前的界面）、MainMenuView（主界面）、GameView（游戏界面）、SharedPreferences、SharedPreferences.Editor、SoundUtil（声音类）以及 HashMap<Integer, Integer>的引用。

● 第 14～20 行为内部类，在内部类中主要是通过 handleMessage 方法接收消息，判断消息的值，根据消息值不同跳转到不同的界面。

● 第 24 行为调用 gotoMainMenuView 方法，跳转到主菜单界面，该方法的具体代码将在下面给出。

● 第 28～43 行为重写的 onKeyDown 方法，为“返回”按钮设置监听，并根据 curr 不同的值执行不同的动作，同时在最后返回 true 值。

（2）上述代码为主控制类 MyBox2dActivity 的框架，接下来将要介绍跳转界面的方法，将下列代码插入到 MyBox2dActivity 类框架代码的第 26 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MyBox2dActivity.java。

```
1     public void gotoMainMenuView(){                  //进入开始界面
2           mmv=new MainMenuView(this);                //创建 MainMenuView 对象
3           setContentView(mmv);                       //设置显示页面
4           curr=WhichView.MAIN_MENU_VIEW;             //为 curr 赋值
5     }
6     public void gotoGameView(){                      //进入游戏界面
7           DRAW_THREAD_FLAG=true;                     //线程 DrawThread 标志位设置为 true
8           START_SCORE=false;                         //设置记分标志位为 false
9           xOffset=0;                                 //X 轴偏移量为 0
10          yOffset=0;                                 //Y 轴偏移量为 0
11          SCORE=0;                                   //分数 SCORE 为 0
12          gv=new GameView(this);                     //创建 GameView 对象
13          setContentView(gv);                        //设置显示页面
14          curr=WhichView.GAME_VIEW;                  //为 curr 赋值
15    }
```

● 第 1～5 行为跳转到主菜单界面的方法，首先创建 MainMenuView 类的对象，然后调用 setContentView 跳转到主菜单界面，最后设置 curr 的值为 MAIN_MENU_VIEW。

● 第 6～15 行是跳转到游戏界面的方法，该方法开始时需要将游戏中对应的标志位、偏移量以及记录的分数初始化，然后获得 GameView 对象并跳转到游戏界面，最后需要为 curr 赋值为 GAME_VIEW。

19.4.2　常量类 Constant

前一小节介绍了游戏的主控制类，接下来讲解常量类，该类中存储了本游戏中大量的变量，在

本游戏中起到了非常大的作用。

（1）首先介绍该类的框架，了解该框架有助于读者对该类整体结构的理解，该类的框架代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    package com.bn.box2d.sndls;                                  //声明包名
2    import android.content.res.Resources;                        //导入相关类
3    import android.graphics.Bitmap;                              //导入相关类
4    public class Constant{
5         public static float RATE = 10;                          //屏幕到现实世界的比例 10px: 1m
6         public static boolean DRAW_THREAD_FLAG=true;            //绘制线程工作标志位
7         public static boolean START_SCORE=false;                //开始记分标志位
8         public static final float TIME_STEP = 2f/60.0f;         //模拟的频率
9         public static final int ITERA = 10;                     //迭代次数
10        public static int SCREEN_WIDTH;                         //屏幕宽度
11        public static int SCREEN_HEIGHT;                        //屏幕高度
12        public static float SCREEN_WIDTH_STANDARD=854;          //屏幕标准宽度
13        public static float SCREEN_HEIGHT_STANDARD=480;         //屏幕标准高度
14        public static float xMainRatio;                         //X 缩放比例
15        public static float yMainRatio;                         //Y 缩放比例
16        public static boolean MAIN_DRAW_THREAD_FLAG=false;      //线程的标志位
17        public static boolean TJ_CONTROL_FLAG=false;
18        public static float xOffset=0;                          //绘制偏移量
19        public static float yOffset=0;
20        public static float CJ_WIDTH=2000;                      //场景宽度
21        public static int SCORE=0;                              //得分
22        ......//该处省略了预加载的图片 ID 数组列表的代码，将在后面给出
23        public static Bitmap[] OTHER_PIC_ARRAY;                 //其余自定义的图片数组
24        public static Bitmap[] PIC_ARRAY;                       //预加载的图片数组
25        public static Bitmap[] NUM_ARRAY;                       //预加载的白色数字图片数组
26        public static Bitmap[] Y_NUM_ARRAY;                     //预加载的黄色数字图片数组
27        public static Bitmap[] P_NUM_ARRAY;                     //预加载的紫色数字图片数组
28        ......//该处省略了加载图片方法的代码，将在后面给出
29        public static float DMGD=40.0f;                         //地面高度
30        public static int currStage=0;                          //当前关卡 0-第一关
31        public static int[] GG_SCORE={10000,15000,20000};       //过关分
32        public static int[] HH_SCORE={0,0,0};
33        ......//该处省略了贴图物体是否静止列表的代码，将在后面给出
34        ......//该处省略了贴图物体贴图数组的代码，将在后面给出
35        ......//该处省略了贴图物体类型的代码，将在后面给出
36        ......//该处省略了贴图物体初始速度、初始位置以及目标尺寸的代码，将在后面给出
37        ......//该处省略了贴图物体相对坐标列表的代码，将在后面给出
38        public static boolean changeRatioOkFlag=false;//是否已经执行过 changeRatio 方法
39        ......//该处省略了动态自适应屏幕方法的代码，将在后面给出
40        ......//该处省略了主界面所用到的变量以及加载图片的代码，读者可以查阅随书光盘中的源代码
41   }
```

- 第 5～9 行为 Constant 类中部分成员变量的声明，主要有 RATE（屏幕到现实世界的比例 10px：1m）、DRAW_THREAD_FLAG（绘制线程工作标志位）、START_SCORE（开始记分标志位）、TIME_STEP（模拟的频率）以及 ITERA（迭代次数）。
- 第 10～13 行为 Constant 类中部分成员变量的声明，主要有 SCREEN_WIDTH（屏幕宽度）、SCREEN_HEIGHT（屏幕高度）、SCREEN_WIDTH_STANDARD（屏幕标准宽度）以及 SCREEN_HEIGHT_STANDARD（屏幕标准高度）。
- 第 14～21 行为 Constant 类中部分成员变量的声明，主要有 xMainRatio（*X* 缩放比例）、

yMainRatio（*Y* 缩放比例）、MAIN_DRAW_THREAD_FLAG（线程的标志位）、TJ_CONTROL_FLAG（线程的标志位）、xOffset（绘制 x 偏移量）、yOffset（绘制 y 偏移量）、CJ_WIDTH（场景宽度）以及 SCORE（得分）。

- 第 23～27 行为 Constant 类中预加载图片数组的声明。
- 第 29～32 行为 Constant 类中部分成员变量的声明，主要有 DMGD（地面高度）、currStage（当前关卡）、GG_SCORE（过关分）以及 HH_SCORE（游戏得分）。

（2）接下来将介绍预加载图片 ID 数组列表数组，将下列代码插入到 Constant 类框架代码的第 22 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    public static int[] PIC_ID={                          //预加载的场景图片 ID 数组列表
2          R.drawable.ground,                              //地面
3          R.drawable.ls_zy,                               //睁眼睛的鼠头
4          R.drawable.xm,                                  //猫头
5          R.drawable.wood_hen,                            //横木头
6          R.drawable.wood_hen_1,                          //横木头
7          R.drawable.wood_hen_2,                          //横木头
8          R.drawable.stone_shu,                           //竖石头
9          R.drawable.ice_hen,                             //横冰块
10         R.drawable.ice_hen_1,                           //横冰块
11         R.drawable.ice_hen_2,                           //横冰块
12         R.drawable.ice_hen_3,                           //横冰块
13         R.drawable.ice_shu,                             //竖冰块
14         R.drawable.ice_shu_1,                           //竖冰块
15         R.drawable.ice_shu_2,                           //竖冰块
16         R.drawable.ice_shu_3,                           //竖冰块
17         R.drawable.bottom_left_ntz,                     //左侧灰泥台子
18         R.drawable.bottom_right_ntz,                    //右侧灰泥台子
19         R.drawable.bg,                                  //场景总背景
20         R.drawable.stone_hen,                           //横石头
21         R.drawable.wood_shu,                            //竖直木头
22         R.drawable.wood_shu_1,                          //竖直木头
23         R.drawable.wood_shu_2,                          //竖直木头
24         R.drawable.dg,                                  //弹弓
25         R.drawable.ls_by,                               //闭眼鼠头
26         R.drawable.pj,                                  //皮筋
27         R.drawable.c_xd,                                //小草
28         R.drawable.c_dd,                                //长势强势的草
29   };
30   public static int[] NUM_B={                           //预加载白色数字图片 ID 数组列表 S_ID
31         ......//该处省略了预加载白色数字图片 ID 数组列表的代码，读者可以查阅随书光盘中的源代码
32   };
33   public static int[] NUM_HS_ID={                       //加载黄色数字图片 ID 数组列表
34         ......//该处省略了预加载黄色数字图片 ID 数组列表的代码，读者可以查阅随书光盘中的源代码
35   };
36   public static int[] NUM_ZS_ID={                       //加载紫色数字图片 ID 数组列表
37         ......//该处省略了预加载紫色数字图片 ID 数组列表的代码，读者可以查阅随书光盘中的源代码
38   };
39   public static int[] OTHER_PIC_ID={                    //预加载其余自定义图片的 ID 数组列表
40         ......//该处省略了预加载其余图片 ID 数组列表的代码，读者可以查阅随书光盘中的源代码
41   };
```

- 第 1～29 行为创建预加载的场景图片 ID 数组列表，主要预加载的图片有游戏场景中的各个刚体的不同状态、地面、总背景、弹弓、皮筋、草等。

- 第 30～32 行为创建预加载白色数字图片 ID 数组列表 NUM_B。
- 第 33～35 行为创建预加载黄色数字图片 ID 数组列表 NUM_HS_ID。
- 第 36～38 行为创建预加载紫色数字图片 ID 数组列表 NUM_ZS_ID。
- 第 39～41 行为创建预加载其余自定义图片的 ID 数组列表 OTHER_PIC_ID。

说明 由于在预加载白色数字图片 ID、黄色数字图片 ID、紫色数字图片 ID 以及其余自定义图片的 ID 的方法与加载场景图片结构相同，所以在这里就不再对其进行赘述。

（3）接下来将介绍加载图片，并调用 PicLoadUtil 类中的 scaleToFitXYRatio 缩放图片的方法，将下列代码插入到 Constant 类框架代码的第 28 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    public static void loadPic(Resources res){                    //加载图片的方法
2          PIC_ARRAY=new Bitmap[PIC_ID.length];                    //声明数组 PIC_ARRAY 长度
3          NUM_ARRAY=new Bitmap[NUM_BS_ID.length];                 //声明数组 NUM_ARRAY 长度
4          Y_NUM_ARRAY=new Bitmap[NUM_HS_ID.length];               //声明数组 Y_NUM_ARRAY 长度
5          P_NUM_ARRAY=new Bitmap[NUM_ZS_ID.length];               //声明数组 P_NUM_ARRAY 长度
6          OTHER_PIC_ARRAY=new Bitmap[OTHER_PIC_ID.length];//声明数组 OTHER_PIC_ARRAY 长度
7          for(int i=0;i<PIC_ID.length;i++){                       //for 循环
8               PIC_ARRAY[i]=PicLoadUtil.loadBM(res, PIC_ID[i]);//调用 loadBM 方法加载图片
9          }
10         for(int i=0;i<NUM_BS_ID.length;i++){                    //for 循环
11              NUM_ARRAY[i]=PicLoadUtil.loadBM(res, NUM_BS_ID[i]);
                                                                   //调用 loadBM 方法加载图片
12              NUM_ARRAY[i]=PicLoadUtil.scaleToFitXYRatio(NUM_ARRAY[i],
13                                                                 yMainRatio,yMainRatio);
14         }
15         for(int i=0;i<NUM_HS_ID.length;i++){                    //for 循环
16              Y_NUM_ARRAY[i]=PicLoadUtil.loadBM(res, NUM_HS_ID[i]);
                                                                   //调用 loadBM 方法加载图片
17              Y_NUM_ARRAY[i]=PicLoadUtil.scaleToFitXYRatio(Y_NUM_ARRAY[i],
18                                                                     yMainRatio,yMainRatio);
19         }
20         for(int i=0;i<NUM_ZS_ID.length;i++){                    //for 循环
21              P_NUM_ARRAY[i]=PicLoadUtil.loadBM(res, NUM_ZS_ID[i]);
                                                                   //调用 loadBM 方法加载图片
22              P_NUM_ARRAY[i]=PicLoadUtil.scaleToFitXYRatio(P_NUM_ARRAY[i],
23                                                                     yMainRatio,yMainRatio);
24         }
25         for(int i=0;i<OTHER_PIC_ID.length;i++){                 //自定义的图片数组
26              OTHER_PIC_ARRAY[i]=PicLoadUtil.loadBM(res, OTHER_PIC_ID[i]);
27              OTHER_PIC_ARRAY[i]=PicLoadUtil.scaleToFitXYRatio(OTHER_PIC_ARRAY[i],
28                                                                         yMainRatio,
                                                                           yMainRatio);
29         }                                                       //图片按照高度缩放
30         PIC_ARRAY[17]=PicLoadUtil.scaleToFitXYRatio(PIC_ARRAY[17],yMainRatio,
           yMainRatio);
31         PIC_ARRAY[22]=PicLoadUtil.scaleToFitXYRatio(PIC_ARRAY[22],yMainRatio*0.7f,
           yMainRatio*0.7f);
32         PIC_ARRAY[1]=PicLoadUtil.scaleToFitXYRatio(PIC_ARRAY[1], yMainRatio,
           yMainRatio);
33         PIC_ARRAY[25]=PicLoadUtil.scaleToFitXYRatio(PIC_ARRAY[25],yMainRatio,
           yMainRatio);
34         PIC_ARRAY[26]=PicLoadUtil.scaleToFitXYRatio(PIC_ARRAY[26],yMainRatio,
           yMainRatio);
```

```
35    }
```

- 第 2～6 行为创建长度固定的 PIC_ARRAY、NUM_ARRAY、Y_NUM_ARRAY、P_NUM_ARRAY 以及 OTHER_PIC_ARRAY 数组。
- 第 7～9 行为 for 循环，主要是将图片通过 PicLoadUtil 类调用 loadBM 方法加载图片并存入到 PIC_ARRAY 数组中。
- 第 10～14 行为 for 循环，主要是将图片通过 PicLoadUtil 类调用 loadBM 方法加载图片存入到 NUM_ARRAY 数组中。然后通过 PicLoadUtil 类调用 scaleToFitXYRatio 方法缩放图片，并存入到 NUM_ARRAY 数组中。
- 第 30～34 行为通过 PicLoadUtil 类调用 scaleToFitXYRatio 方法缩放图片，并存入到 PIC_ARRAY 数组中。

说明　第 15～19 行、第 20～24 行以及第 25～29 行的构造与第 10～14 行的构造相同，所以在这里就不再赘述，感兴趣的读者可以参照第 10～14 行来理解。

（4）接下来介绍贴图物体是否静止列表的数组，将下列代码插入到 Constant 类框架代码的第 33 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    public static boolean[][] IS_MOVE={                    //贴图物体是否静止列表
2         {                                                 //第一关的是否静止列表
3              true,true,true,                              //全部为 true
4              false,false,false,false,false,               //全部为 false
5              false,false,false,false,false,               //全部为 false
6              false,false,false,false,false,               //全部为 false
7              false,false,false,false,false,               //全部为 false
8              false,false,false                            //全部为 false
9         }
10        ......//该处省略了第二关、第三关的贴图物体是否静止列表的代码，读者可以查阅随书光盘中的源代码
11   };
```

说明　在 IS_MOVE 数组中，true 表示的是对应的刚体为静态，false 表示的是对应的刚体为动态。

（5）接下来将要介绍贴图物体贴图的图片数组索引列表，将下列代码插入到 Constant 类框架代码的第 34 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    public static int[][][] IMG_ID={                       //贴图物体贴图的图片数组索引列表
2         {                                                 //第一关的物体贴图数组
3              {0},{15},{16},{6},{2},                       //地面、灰泥条子以及石条
4              {11,12,13,14},{3,4,5},                       //冰块、木条以及石条
5              {18},                                        //左侧上方横着的石头
6              {6},{2},{19,20,21},                          //石条、猫头以及木条
7              {7,8,9,10},{18},{18},{18},                   //冰块以及石条
8              {18},{18},{6},{6},{18},                      //横着与竖直的石条
```

```
9                {6},{6},{18},                                    //竖直与横着的石条
10               {19,20,21},{19,20,21},{7,8,9,10},                //木条以及冰块
11           },
12           ......//该处省略了第二关与第三关物体贴图数组的代码，读者可以自行查阅随书光盘中的源代码
13   };
```

- 第 3～5 行以及第 8～9 行为声明游戏场景中左侧刚体的索引值。
- 第 6～7 行以及第 10 行为声明游戏场景中右侧刚体的索引值。

说明 由于第二关与第三关物体贴图数组与第一关物体贴图数组的结构相同，所以在这里就不再赘述，读者可以自行查看本书中随书光盘的源代码。

（6）接下来将介绍贴图物体的类型的数组，将下列代码插入到 Constant 类框架代码的第 35 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    public static BodyType[][] typeA={                          //贴图物体的类型
2        {                                                       //第一关的贴图物体类型
3            BodyType.DM,BodyType.HNTZ,BodyType.HNTZ,            //声明刚体的类型
4            BodyType.ST,BodyType.XM,BodyType.BK,BodyType.MT,BodyType.ST,
5            BodyType.ST,BodyType.XM,BodyType.MT,BodyType.BK,BodyType.ST,
6            BodyType.ST,BodyType.ST,BodyType.ST,BodyType.ST,BodyType.ST,
7            BodyType.ST,BodyType.ST,BodyType.ST,BodyType.ST,BodyType.ST,
8            BodyType.MT,BodyType.MT,BodyType.BK,                //声明刚体的类型
9        },
10       ......//该处省略了第二关与第三关贴图物体类型的代码，读者可以自行查阅随书光盘中的源代码
11   };
```

说明 第 3～8 行为本游戏中第一关所有实体的类型（不包括背景以及草），该类型存储在 BodyType 中，同时由于第二关与第三关的贴图物体类型的代码与第一关的相似，所以在这里就不再赘述。

（7）接下来介绍贴图物体初始速度、初始位置以及目标尺寸的数组，将下列代码插入到 Constant 类框架代码的第 36 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    public static float[][][] SPEED={                           //贴图物体初始速度列表
2        {                                                       //第一关的贴图物体初始速度
3            {0,0},{0,0},{0,0},                                  //速度全部为 0
4            {0,0},{0,0},{0,0},{0,0},{0,0},                      //速度全部为 0
5            {0,0},{0,0},{0,0},{0,0},{0,0},                      //速度全部为 0
6            {0,0},{0,0},{0,0},{0,0},{0,0},                      //速度全部为 0
7            {0,0},{0,0},{0,0},{0,0},{0,0},                      //速度全部为 0
8            {0,0},{0,0},{0,0}                                   //速度全部为 0
9        },
10       ......//该处省略了第二关与第三关贴图物体初始速度的代码，读者可以自行查阅随书光盘中的源代码
11   };
12   public static float[][][] LOCATION={                        //贴图物体的初始位置
13       {                                                       //第一关贴图物体的初始位置
14           {0,440},{760,400},{1000,360},                       //对应贴图位置
15           {840,340},{870,360},{920,340},{840,320},{920,300},  //对应贴图位
16           {1000,300},{1030,320},{1080,300},{1000,280},{1080,260}, //对应贴图位
```

```
17              {840,300},{1000,260},{880,300},{1040,260},{1240,260},    //对应贴图位
18              {1340,260},{1240,240},{1240,60},{1340,60},{1240,40},     //对应贴图位
19              {1120,300},{1180,300},{1120,280}                         //对应贴图位
20          },
21          ......//该处省略了第二关与第三关贴图物体的初始位置的代码，读者可以自行查阅随书光盘中的源代码
22      };
23      public static float[][][] SIZE={                                //贴图物体的目标尺寸
24          {                                                           //第一关贴图物体的目标尺寸
25              {2000,40},{200,40},{240,80},                            //相关贴图物体的目标尺寸
26              {20,60},{40,40},{20,60},{100,20},{20,20},               //相关贴图物体的目标尺寸
27              {20,60},{40,40},{20,60},{100,20},{20,20},               //相关贴图物体的目标尺寸
28              {20,20},{20,20},{20,20},{20,20},{20,180},               //相关贴图物体的目标尺寸
29              {20,180},{120,20},{20,180},{20,180},{120,20},           //相关贴图物体的目标尺寸
30              {20,60},{20,60},{80,20}                                 //相关贴图物体的目标尺寸
31          },
32          ......//该处省略了第二关与第三关贴图物体目标尺寸的代码，读者可以自行查阅随书光盘中的源代码
33      };
```

- 第 2～9 行为本游戏第一关中所有实体的初速度（x 与 y 方向的，但是不包括背景和草）。该数据存储在三维数组中，主要涉及到的实体有地面、灰泥台子、石条、猫头、冰块以及木条。
- 第 13～19 行为本游戏第一关中所有实体的初始位置，即摆放位置（不包括背景和草），该数据存储在一个三维数组中。主要涉及的实体有地面、灰泥台子、石条、猫头、冰块以及木条。
- 第 24～31 行为本游戏第一关贴图物体的目标尺寸（不包括背景和草），该数据也是存储在一个三维数组中。主要涉及的实体有地面、灰泥台子、石条、猫头、冰块以及木条。

说明　由于第二关与第三关的贴图物体初始速度、初始位置以及目标尺寸与第一关的结构类似，所以在这里就不再对这两关进行赘述。

（8）接下来将介绍贴图物体相对坐标的列表，将下列代码插入到 Constant 类框架代码的第 37 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1       public static float[][][][] STONE_VERTEX={                  //贴图物体相对坐标列表
2               {                                                   //第一关贴图物体相对坐标列表
3                   {                                               //地面
4                       {0,0},{2000,0},{2000,40},{0,40}
5                   },{                                             //左侧灰泥台子
6                       {20,0},{200,0},{200,40},{0,40}
7                   },{                                             //右侧灰泥台子
8                       {0,0},{220,0},{240,80},{0,80}
9                   },{                                             //左侧物件中的石条
10                      {0,0},{20,0},{20,60},{0,60}
11                  },{                                             //左侧猫头
12                      {0,0},{38,0},{40,20},{32,36},{6,36},{0,20}
13              },{                                                 //左侧物件中的冰块
14                      {0,0},{20,0},{20,60},{0,60}
15              },{                                                     //左侧物件中上方的木条
16                      {0,0},{100,0},{100,20},{0,20}
17              },{                                                     //左侧上方横着的石头
18                      {0,0},{20,0},{20,20},{0,20}
19              },{                                                     //右侧物件中的石条
20                      {0,0},{20,0},{20,60},{0,60}
21                  },{                                                 //右侧猫头
22                      {0,0},{38,0},{40,20},{32,36},{6,36},{0,20}
```

```
23                  },{                                                     //右侧物件中的木条
24                          {0,0},{20,0},{20,60},{0,60}
25                  },{                                                     //右侧物件中上方的冰块
26                          {0,0},{100,0},{100,20},{0,20}
27                  },{                                                     //右侧上方横着的石头
28                          {0,0},{20,0},{20,20},{0,20}
29                  },{                                                     //左侧上方横着的石头
30                          {0,0},{20,0},{20,20},{0,20}
31                  },{                                                     //右侧上方横着的石头
32                          {0,0},{20,0},{20,20},{0,20}
33                  },{                                                     //左侧上方中间横着的石头
34                          {0,0},{20,0},{20,20},{0,20}
35                  },{                                                     //右侧上方中间横着的石头
36                          {0,0},{20,0},{20,20},{0,20}
37                  },{                                                     //右侧石头
38                          {0,0},{20,0},{20,180},{0,180}
39                    },{                                                   //右侧石头
40                          {0,0},{20,0},{20,180},{0,180}
41                    },{                                                   //右侧上方横着的石头
42                          {0,0},{120,0},{120,20},{0,20}
43                  },{                                                     //右侧石头
44                          {0,0},{20,0},{20,180},{0,180}
45                    },{                                                   //右侧石头
46                          {0,0},{20,0},{20,180},{0,180}
47                    },{                                                   //右侧上方横着的石头
48                          {0,0},{120,0},{120,20},{0,20}
49                  },{                                                     //右侧物件中的木条
50                          {0,0},{20,0},{20,60},{0,60}
51                    },{                                                   //右侧物件中的木条
52                          {0,0},{20,0},{20,60},{0,60}
53                  },{                                                     //右侧物件中上方的冰块
54                          {0,0},{80,0},{80,20},{0,20}
55                  }
56              },
57          ......//该处省略了第二关与第三关贴图物体相对坐标列表的代码，读者可以自行查阅随书光盘中的
            源代码
58          };
```

- 第 5～7 行、9～19 行、29～31 行以及 32～35 行为本游戏中第一关场景中左侧的实体，主要有灰泥台子、石条、猫头、冰块以及木条。
- 第 7～9 行、19～29 行、31～33 行以及 35～55 行为本游戏中第一关场景中右侧的实体，主要有灰泥台子、石条、猫头、冰块以及木条。

说明　由于第二关与第三关贴图物体相对坐标列表结构与第一关的相同，所以在这里就不再对其进行赘述，有兴趣的读者可以查看随书光盘中的源代码。

（9）接下来将介绍动态适应屏幕的方法，了解该方法有助于拓展读者在开发软件时处理屏幕问题的思路，将下列代码插入到 Constant 类框架代码的第 39 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    public static void changeRatio(){                  //动态自适应屏幕的方法
2        if(changeRatioOkFlag){                         //判断 changeRatioOkFlag 是否为 true
3            return;                                    //直接返回
4        }
5        changeRatioOkFlag=true;                        //设置 changeRatioOkFlag 为 true
6        xMainRatio=SCREEN_WIDTH/SCREEN_WIDTH_STANDARD;
```

```
7          yMainRatio=SCREEN_HEIGHT/SCREEN_HEIGHT_STANDARD;
8          CJ_WIDTH=CJ_WIDTH*yMainRatio;              //场景宽度缩放
9          RATE=RATE*yMainRatio;                      //比值缩放
10         DMGD=DMGD*yMainRatio;                      //地面高度缩放
11         for(int k=0;k<SIZE.length;k++){            //for 循环
12              for(int i=0;i<SIZE[k].length;i++){                  //for 循环
13                   SIZE[k][i][0]=SIZE[k][i][0]*yMainRatio;        //获得缩放的大小
14                   SIZE[k][i][1]=SIZE[k][i][1]*yMainRatio;        //获得缩放的大小
15              }
16              for(int i=0;i<STONE_VERTEX[k].length;i++){          //for 循环
17                   for(int j=0;j<STONE_VERTEX[k][i].length;j++){ //for 循环
18                        STONE_VERTEX[k][i][j][0]=STONE_VERTEX[k][i][j][0]*yMainRatio;
19                        STONE_VERTEX[k][i][j][1]=STONE_VERTEX[k][i][j][1]*yMainRatio;
20              } }
21              for(int i=0;i<LOCATION[k].length;i++){              //for 循环
22                   LOCATION[k][i][0]=LOCATION[k][i][0]*yMainRatio;
23                   LOCATION[k][i][1]=LOCATION[k][i][1]*yMainRatio;
24   } } }
```

- 第 2～4 行为判断 changeRatioOkFlag 是否为 true，如果为 true，则直接返回。
- 第 5～10 行为设置 changeRatioOkFlag 为 true，得到 x 方向的缩放比 xMainRatio 以及 y 方向的缩放比 yMainRatio，并将场景宽度、屏幕到现实世界的比例以及地面的高度按照 yMainRatio 缩放。
- 第 12～15 行为按照 yMainRatio 将贴图物体的目标尺寸进行缩放。
- 第 16～20 行为按照 yMainRatio 将贴图物体相对坐标进行缩放。
- 第 21～24 行为按照 yMainRatio 将贴图物体的初始位置进行缩放。

（10）接下来将介绍主界面所用到的变量、预加载图片的数组、加载图片方法以及获得相关按钮的位置方法，将下列代码插入到 Constant 类框架代码的第 41 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Constant.java。

```
1    public static int SLEEPTIME=15;                    //线程休眠时间
2    public static float XOFFSET=1;                     //图片连接处的修正值
3    public static float SET_X_OFFSET=10;               //设置按钮距离左右的位置
4    public static float SET_Y_OFFSET=10;               //设置按钮距离上下的位置
5    public static float SET_BACK_CK_OFFSET=10;         //偏移量
6    public static int[] PIC_NUM={                      //得到图片的编号
7         R.drawable.bg,                                //背景图片
8         R.drawable.ground,                            //地面图片
9         R.drawable.c_dd,                              //长势好的草
10        R.drawable.c_xd,                              //小草
11        R.drawable.ls_zy,                             //睁眼鼠头
12        R.drawable.ls_by,                             //闭眼鼠头
13        R.drawable.play,                              //开始按钮
14        R.drawable.set,                               //设置按钮
15        R.drawable.xm,                                //猫头
16        R.drawable.set_back,                          //设置中的背景
17        R.drawable.set_sy,                            //设置子选项的声音按钮
18        R.drawable.set_xx,                            //设置子选项的信息查看按钮
19        R.drawable.set_ck,                            //设置子选项获取分数信息查看按钮
20        R.drawable.bg_fg,
21        R.drawable.about,                             //关于界面
22        R.drawable.close,                             //关闭按钮
23        R.drawable.set_close,                         //关闭声音按钮
24        R.drawable.set_cl,                            //齿轮
25        R.drawable.sndls                              //标题
```

```
26    };
27    public static Bitmap[] PIC_BITMAP;                    //图片数组
28    public static Bitmap[] PE_ARRAY;                      //缩放后的图片数组
29    public static void initBitmap(Resources e){           //加载图片数组的方法
30          xMainRatio=SCREEN_WIDTH/SCREEN_WIDTH_STANDARD;  //计算比例
31          yMainRatio=SCREEN_HEIGHT/SCREEN_HEIGHT_STANDARD;
32          PIC_BITMAP=new Bitmap[PIC_NUM.length];          //缩放数组
33          PE_ARRAY=new Bitmap[PIC_NUM.length];            //缩放数组
34          for(int i=0;i<PIC_NUM.length;i++){              //for 循环
35                PIC_BITMAP[i]=PicLoadUtil.loadBM(e, PIC_NUM[i]);
36          }
37          for(int j=0;j<PIC_NUM.length;j++){              //for 循环
38                PE_ARRAY[j]=PicLoadUtil.scaleToFitXYRatio(PIC_BITMAP[j],   yMainRatio,
                  yMainRatio);
39          }
40    }
41    public static float[][] LOCALTION_BUTTON;             //坐标的二维数组
42    public static float[][] getLocaltionAndWH(){          //得到各个间距
43          //设置按钮与弹出按钮的间距
44          float SET_SPACE=(PE_ARRAY[7].getWidth()-PE_ARRAY[9].getWidth())/2;
45          LOCALTION_BUTTON=new float[][]{                 //位置坐标数组
46                ……//该类省略了部分按钮位置信息的代码，读者可以查阅随书光盘中的源代码
47          };
48          return LOCALTION_BUTTON;
49    }
```

- 第 1～5 行为相应的成员变量的声明，主要有 SLEEPTIME（线程休眠时间）、XOFFSET（图片连接处的修正值）、SET_X_OFFSET（设置按钮距离左右的位置）、SET_Y_OFFSET（设置按钮距离上下的位置）以及 SET_BACK_CK_OFFSET（偏移量）。
- 第 6～28 行为与主界面相关的预加载的图片数组，并声明相应的缩放后的 PIC_BITMAP 数组与 PE_ARRAY 数组。
- 第 29～40 行为加载图片数组的方法，首先计算屏幕的宽度比例以及高度比例，然后创建相应的 PIC_BITMAP 与 PE_ARRAY 数组，最后通过调用 PicLoadUtil 类的 loadBM 方法加载图片以及调用 scaleToFitXYRatio 缩放图片。
- 第 41～49 行首先声明坐标的二维数组，然后创建 getLocaltionAndWH 方法，该方法返回值为二维数组，计算设置按钮与弹出按钮的间距 SET_SPACE，初始化位置坐标数组 LOCALTION_BUTTON，最后用 LOCALTION_BUTTON 数组作为本方法的返回值。

19.5 主界面的设计与实现

前一节已经介绍了本游戏的主控制类 MyBox2dActivity，本节中将介绍实现主界面的具体代码。主界面是游戏开始时的界面，主要包括 Taj（鼠头和猫头类）、TJThread（鼠头和猫头控制类）、MainMenuView（主界面）以及 MainMenuDrawThread（刷帧线程）。

19.5.1 鼠头和猫头类 Taj

本类的主要作用是根据鼠头或者猫头的速度，动态地绘制鼠头或者猫头，从而达到鼠头或者猫头飞出的效果，使得主界面更加绚丽，该类的具体代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Taj.java。

```
1    package com.bn.box2d.sndls;                                    //声明包名
2    import static com.bn.box2d.sndls.Constant.*;                   //静态导入
3    ......//该处省略了部分导入相关类的代码，读者可以查阅随书光盘的源代码
4    import android.graphics.Paint;                                 //导入相关包
5    public class Taj{
6         int tajIndex;                                             //对应鼠头或者猫头的编号
7         float xoffset;                                            //x 位置
8         float yoffset;                                            //y 位置
9         float vx;                                                 //x 速度
10        float vy;                                                 //y 速度
11        int timeSpan=0;                                           //时间间隔
12        float maxOffset=1f;                                       //鼠头或者猫头开始的宽度
13        public Taj(int tajIndex,float xoffset,float yoffset,float vx,float vy){
14             this.tajIndex=tajIndex;                              //初始化鼠头或者猫头编号
15             this.xoffset=xoffset;                                //初始化 x 位置
16             this.yoffset=yoffset;                                //初始化 y 位置
17             this.vx=vx;                                          //初始化 x 速度
18             this.vy=vy;                                          //初始化 y 速度
19        }
20        public void drawSelf(Canvas canvas,Paint paint){//绘制方法
21             float x=xoffset+vx*timeSpan;              //得到 x 位移
22             float y=yoffset-vy*timeSpan+0.5f*timeSpan*timeSpan*1.0f;//得到 y 位移
23             float offset=0.3f;                        //设置鼠头或者猫头大小的浮动值
24             Matrix m1=new Matrix();                   //创建 Matrix 对象
25             m1.setTranslate(x, y);                    //平移至(x, y)处
26             Matrix m2=new Matrix();                   //创建 Matrix 对象
27             if(vx>=5&&vy>=5){                         //判断 vx>=5 与 vy>=5 是否同时成立
28                  m2.setScale(maxOffset-offset, maxOffset-offset);//设置鼠头或者猫头缩放
29             }else if(vx<5&&vy<5){                          //判断 vx<5 与 vy<5 是否同时成立
30                  m2.setScale(maxOffset+offset, maxOffset+offset);//设置鼠头或者猫头缩放
31             }else{
32                  m2.setScale(maxOffset, maxOffset);    //设置鼠头或者猫头缩放
33             }
34             Matrix m3=new Matrix();                        //创建 Matrix 对象
35             m3.setConcat(m1, m2);                          //组合矩阵 m1 与 m2
36             if(xoffset<0||xoffset>SCREEN_WIDTH||yoffset>SCREEN_HEIGHT){
37                  return;                                   //返回
38             }else{
39                  canvas.drawBitmap(PE_ARRAY[tajIndex], m3, paint);//绘制鼠头或者猫头
40   } } }
```

- 第 6～12 行为本类成员变量的声明，包括 tajIndex（鼠头或者猫头的编号）、xoffset（x 位置）、yoffset（y 位置）、vx（x 速度）、vy（y 速度）、timeSpan（时间间隔）以及 maxOffset（鼠头或者猫头原始大小）。
- 第 13～19 行为本类的构造器，在该构造器中主要是初始化鼠头或者猫头编号、位置以及速度。
- 第 21～22 行为根据鼠头或者猫头的速度以及时间动态地计算其 x 位移以及 y 位移。
- 第 24～25 行为创建 Matrix 对象，同时调用 setTranslate 方法将鼠头或者猫头平移到（x, y）处。
- 第 26～33 行是创建 Matrix 对象，根据 vx 以及 vy 的大小调用 setScale 缩放图像。

- 第 36～40 行为判断鼠头或者猫头是否在屏幕范围内。如果不在屏幕范围内，直接返回，否则，再绘制鼠头或者猫头。

19.5.2 鼠头和猫头控制线程——TJThread

前一小节介绍的是绘制鼠头和猫头的类 Taj，接下来介绍鼠头和猫头控制类 TJThread。该类的主要作用是每隔固定的时间重新计算鼠头和猫的位置以及速度，将得到的 Taj 对象存入 List<Taj>集合中，同时在主界面中遍历该集合，并绘制鼠头和猫头。该类具体代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 TJThread.java。

```
1    package com.bn.box2d.sndls;                          //声明包名
2    import static com.bn.box2d.sndls.Constant.*;         //静态导入
3    import java.util.ArrayList;                          //导入相关包
4    import java.util.List;                               //导入相关包
5    public class TJThread extends Thread{
6         static float lsxoffset;                         //鼠头或者猫头的 X 位置
7         static float lsyoffset;                         //鼠头或者猫头的 Y 位置
8         static float vx;                                //鼠头或者猫头 x 方向的速度
9         static float vy;                                //鼠头或者猫头 y 方向的速度
10        int[] num={4,5,8,4,5,5,4,5,5,4,5,8};            //飞出的鼠头或者猫头编号
11        int index;                                      //飞出鼠头或者猫头的索引值
12        MainMenuView tjview;                            //创建 MainMenuView 对象
13        List<Taj> tlist=new ArrayList<Taj>();           //创建 List<Taj>集合
14        public TJThread(MainMenuView tjview){           //构造器
15             this.tjview=tjview;                        //初始化 tjview
16        }
17        public void run(){                              //继承 Thread 类需要重写的 run()方法
18             while(TJ_CONTROL_FLAG){                    //循环标志位 TJ_CONTROL_FLAG
19                  lsxoffset=(float) (SCREEN_WIDTH/2*Math.random());
20                  lsyoffset=SCREEN_HEIGHT;              //得到飞出鼠头或者猫头的 y 位置
21                  vx=(float) (25*Math.random())*yMainRatio; //得到 x 速度
22                  vy=(float) (25*Math.random())*yMainRatio; //得到 y 速度
23                  index=(int)(Math.random()*num.length);    //得到鼠头或者猫头的索引值
24                  this.tlist.add(new Taj(num[index],lsxoffset,lsyoffset,vx,vy));
25                  try{
26                       Thread.sleep(2000);                  //线程休息 2000ms
27                  }catch(Exception e){                      //捕获异常
28                       e.printStackTrace();                 //打印堆栈信息
29   } } } }
```

- 第 6～13 行为本类成员变量的声明，主要有鼠头或者猫头的 x 位置（lsxoffset）、y 位置（lsyoffset）、x 方向的速度（vx）、y 方向的速度（vy）、飞出的鼠头或者猫头编号（num）、飞出鼠头或者猫头的索引值（index）、MainMenuView 的引用（tjview）以及 List<Taj>集合（tlist）。
- 第 14～16 行是本类的构造器，在该构造器中主要是初始化 MainMenuView 的引用。
- 第 17～29 行为继承 Thread 类需要重写的 run()方法，该方法中首先根据 TJ_CONTROL_FLAG 的值，判断 while 循环是否成立。如果成立，则随机得到鼠头或者猫头初始的位置以及速度，并将得到的 Taj 对象存入 List<Taj>集合中。

19.5.3 主界面 MainMenuView

前两小节主要介绍的是鼠头和猫头类 Taj、控制鼠头和猫头的位置以及速度的类 TJThread，接

下来在本小节将介绍主界面类 MainMenuView。该类的主要作用是对主界面进行绘制，并为界面中的部分按钮添加监听器。

（1）首先介绍 MainMenuView 类的框架，了解该框架有助于读者对 MainMenuView 类整体结构的理解，该类的框架代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MainMenuView.java。

```
1    package com.bn.box2d.sndls;                          //声明包名
2    import static com.bn.box2d.sndls.Constant.*;//静态导入
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import android.view.SurfaceView;                     //导入相关包
5    public class MainMenuView extends SurfaceView implements SurfaceHolder.Callback{
6         Paint paint;                                    //声明画笔的引用
7         List<Taj> tlist=new ArrayList<Taj>();           //鼠头和猫头的集合
8         int flag;                                       //按钮的标志位
9         boolean isTouch;                                //设置按钮是否被按下的标志位
10        boolean syisTouch;                              //子菜单中的声音按钮是否按下标志位
11        boolean ckisTouch;                              //子菜单中的公司信息查看按钮是否被按下
12        boolean szisTouch;                              //子菜单中的分数信息查看按钮是否被按下
13        boolean isClose;                                //是否关闭按钮
14        TJThread thd;                                   //控制鼠头和猫头飞出的线程
15        MainMenuDrawThread mmt;                         //界面刷帧线程
16        MyBox2dActivity activity;                       //主控制类 MyBox2dActivity 的引用
17        String str;                                     //临时存储分数
18        boolean degreesFlag=true;                       //旋转的标志位
19        public MainMenuView(MyBox2dActivity activity){          //构造器
20             super(activity);                           //调用父类
21             this.activity=activity;                    //初始化 MyBox2dActivity 类的引用
22             this.getHolder().addCallback(this);    //设置生命周期回调接口的实现者
23             paint=new Paint();                         //创建画笔
24             paint.setAntiAlias(true);                  //打开抗锯齿
25        }
26        public void onDraw(Canvas canvas){              //绘制
27             drawSelf(canvas,paint,PE_ARRAY[0],bgxoffset,0,1);   //绘制背景
28             drawSelf(canvas,paint,PE_ARRAY[2],bdxoffset,        //绘制强势的草
29                     SCREEN_HEIGHT-PE_ARRAY[1].getHeight()-PE_ARRAY[2].getHeight(),
                       0);
30             if(this.tlist.size()!=0){                  //绘制飞的鼠头
31                  float SPEED=1.3f;                     //设置时间间隔
32                  for(Taj tj:this.tlist){               //循环遍历 List<Taj>集合
33                       tj.timeSpan+=SPEED;              //增加时间
34                       tj.drawSelf(canvas, paint); //绘制鼠头或者猫头
35             } }
36             for(int i=0;i<thd.tlist.size();i++){//循环遍历 TJThread 中的 List<Taj>集合
37                  this.tlist.clear();          //清空本类中 List<Taj>集合中的数据
38                  if(thd.tlist.size()!=0){   //判断 TJThread 中的 List<Taj>集合是否为空
39                       this.tlist.add(thd.tlist.get(i));     //将数据存入 List<Taj>集合
40             } }
41             ......//该处省略了绘制草、地面以及游戏名称的代码，读者可以自行查阅随书光盘中的源代码
42             paint.setAlpha(128);                                  //设置透明度
43        canvas.drawBitmap(PE_ARRAY[6],LOCALTION_BUTTON[4][0],LOCALTION_BUTTON[4]
          [1],paint);
44             paint.reset();                                        //恢复画笔设置
45             paint.setAlpha(190);                                  //设置透明度
46             ......//该处省略了绘制设置按钮以及设置按钮子菜单的代码，将在后面给出
47        }
48        ......//该处省略了自定义的绘制方法，将在后面给出
49        ......//该处省略了触控监听的 onTouchEvent 方法，将在后面给出
```

```
50          public void surfaceChanged(SurfaceHolder holder, int format, int width,int
            height){ }
51          public void surfaceCreated(SurfaceHolder holder){    //需要重写的方法
52               MAIN_DRAW_THREAD_FLAG=true;                    //刷帧线程标志位为 true
53               TJ_CONTROL_FLAG=true;                          //线程标志位为 true
54               if(mmt==null){                                 //判断 mmt 是否为空
55                    mmt=new MainMenuDrawThread(this);         //创建 MainMenuDrawThread 对象
56                    mmt.start();                              //开启线程
57               }
58               if(thd==null){                                 //判断 thd 是否为空
59                    thd=new TJThread(this);                   //创建 TJThread 对象
60                    thd.start();                              //开启线程
61          } }
62          public void surfaceDestroyed(SurfaceHolder holder){ //需要重写的方法
63               MAIN_DRAW_THREAD_FLAG=false;                     //刷帧线程标志位为 false
64               TJ_CONTROL_FLAG=false;                           //线程标志位为 false
65               mmt=null;                                        //mmt 赋值为空
66               thd=null;                                        //thd 赋值为空
67          }
68          ......//该处省略了自定义的刷帧方法，将在后面给出
69     }
```

- 第 6～18 行为本类的成员变量的声明，主要有画笔 Paint、存储鼠头或者猫头的集合 List<Taj>、按钮是否被按下的标志位、界面刷帧线程 MainMenuDrawThread、鼠头或者猫头的控制线程 TJThread 以及主控制类 MyBox2dActivity 的引用。
- 第 19～25 行为本类的构造器，在该构造器中主要是初始化 MyBox2dActivity 类的对象、设置生命周期回调接口的实现者、创建画笔以及打开抗锯齿。
- 第 27～29 行为调用自定义的方法绘制主界面中的背景以及草。
- 第 30～40 行为循环遍历本类中的 List<Taj>集合绘制飞出的鼠头或者猫头，然后循环遍历 TJThread 中的 List<Taj>集合，将该集合中的数据存入本类中的 List<Taj>集合中。
- 第 51～61 行为实现 SurfaceHolder.Callback 接口需要重写 surfaceCreated(SurfaceHolder holder)的方法，在该方法中主要设置 MAIN_DRAW_THREAD_FLAG、TJ_CONTROL_FLAG 标志位为 true，并判断 mmt 与 thd 是否为空，如果为空则创建 mmt 与 thd 对象，并开启线程。
- 第 62～67 行为实现 SurfaceHolder.Callback 接口需要重写 surfaceDestroyed(SurfaceHolder holder)的方法，该方法中将 MAIN_DRAW_THREAD_FLAG、TJ_CONTROL_FLAG 标志位设置为 false，同时设置 mmt 与 thd 是否为空。

（2）接下来将介绍绘制设置按钮以及设置按钮子菜单的具体代码，将下列代码插入到 MainMenuView 类框架中的第 46 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MainMenuView.java。

```
canvas.save();                                          //保存画布状态
     Matrix m1=new Matrix();                            //平移矩阵
     Matrix m2=new Matrix();                            //旋转矩阵
     Matrix m3=new Matrix();                            //组合以上矩阵
     if(degreesFlag){                                   //degreesFlag 值为 true 时
          m1.setTranslate(LOCALTION_BUTTON[0][0], LOCALTION_BUTTON[0][1]);
          m2.setRotate(mmt.degrees,LOCALTION_BUTTON[0][0]+PE_ARRAY[7].getWidth()/2,
                       LOCALTION_BUTTON[0][1]+PE_ARRAY[7].getHeight()/2);
          m3.setConcat(m2, m1);                         //先旋转后平移
```

```
10          }else{                                          //直接绘制
11               m3.setTranslate(LOCALTION_BUTTON[0][0], LOCALTION_BUTTON[0][1]);
12          }                                               //绘制设置按钮
13          canvas.drawBitmap(PE_ARRAY[7],LOCALTION_BUTTON[0][0],LOCALTION_BUTTON[0]
            [1],paint);
14          canvas.drawBitmap(PE_ARRAY[17], m3, paint);//绘制齿轮
15          canvas.restore();                               //恢复画布状态
16          if(flag==1){                                    //绘制上移子菜单
17               if(syoffset<=PE_ARRAY[9].getHeight()){//syoffset 的值小于半透明图片的高度
18                    degreesFlag=true;                     // degreesFlag 为 true
19                    canvas.clipRect(LOCALTION_BUTTON[1][0],   //clip 矩形区域
20                                    SCREEN_HEIGHT-PE_ARRAY[1].getHeight()-
                                      syoffset,
21                                    LOCALTION_BUTTON[1][0]+PE_ARRAY[9].getWidth(),
22                                    SCREEN_HEIGHT-PE_ARRAY[1].getHeight());
23                    ......//该处省略了绘制上移时的子菜单的代码，读者可以自行查阅随书光盘中的源代码
24               }else{
25                    ......//该处省略了绘制子菜单的代码，读者可以自行查阅随书光盘中的源代码
26                    isTouch=true;                         //设置 isTouch 为 true
27                    degreesFlag=false;                    //设置 degreesFlag 为 false
28          } }
29          if(flag==2){                                    //绘制下移子菜单
30               if(xyoffset<=PE_ARRAY[9].getHeight()){// xyoffset 的值小于半透明图片的高度
31                    degreesFlag=true;                     // degreesFlag 为 true
32                    canvas.clipRect(LOCALTION_BUTTON[1][0],   //clip 矩形区域
33                         SCREEN_HEIGHT-PE_ARRAY[1].getHeight()-PE_ARRAY[9].getHeight()
                           +xyoffset,
34                                    LOCALTION_BUTTON[1][0]+PE_ARRAY[9].getWidth(),
35                                    SCREEN_HEIGHT-PE_ARRAY[1].getHeight());
36                    ......//该处省略了绘制下移时的子菜单的代码，读者可以自行查阅随书光盘中的源代码
37               }else{
38                    szisTouch=false;                          //设置 szisTouch 为 false
39                    isTouch=false;                            //设置 isTouch 为 false
40                    degreesFlag=false;                        //设置 degreesFlag 为 false
41          } }
42          canvas.save();                                    //保存画布状态
43          canvas.drawBitmap(PE_ARRAY[7],LOCALTION_BUTTON[0][0],LOCALTION_BUTTON[0][1],
            paint);
44          canvas.drawBitmap(PE_ARRAY[17], m3, paint);       //绘制齿轮
45          canvas.restore();                                 //恢复画布状态
46          paint.reset();                                    //恢复画笔
47          if(ckisTouch){                                    //此处点击可以查看本游戏的信息
48               isTouch=false;                               //isTouch 设置为 false
49               if(zyoffset<=PE_ARRAY[14].getWidth()){       // zyoffset 小于信息图片的宽度
50                    canvas.clipRect(0, 0,zyoffset+PE_ARRAY[15].getWidth()/2,SCREEN_
                      HEIGHT);
51                    canvas.drawBitmap(PE_ARRAY[14],-PE_ARRAY[14].getWidth()+zyoffset,
                      0,paint);
52                    canvas.drawBitmap(PE_ARRAY[15],-PE_ARRAY[15].getWidth()/2+zyoffset,
53                                    SCREEN_HEIGHT-PE_ARRAY[15].getHeight()*3/2,
                                      paint);
54               }else{
55                    canvas.drawBitmap(PE_ARRAY[14],0,0,paint);     //绘制信息窗口
56               canvas.drawBitmap(PE_ARRAY[15],PE_ARRAY[14].getWidth()-PE_ARRAY[15].
                 getWidth()/2,
57                                    SCREEN_HEIGHT-PE_ARRAY[15].getHeight()*3/2,
                                      paint);
58          } }
59          if(isClose){                                                  //点击的是“关闭”按钮
60               if(yyoffset<=PE_ARRAY[14].getWidth()+PE_ARRAY[15].getWidth()/2){
61                    canvas.clipRect(0,0,PE_ARRAY[14].getWidth()+PE_ARRAY[15].getWidth()/
                      2-yyoffset,
```

```
62                                          SCREEN_HEIGHT);          //clip 矩形区域
63                  canvas.drawBitmap(PE_ARRAY[14], -yyoffset, 0, paint);
64                  canvas.drawBitmap(PE_ARRAY[15],                  //绘制关闭按钮
65                                        PE_ARRAY[14].getWidth()-PE_ARRAY[15].
                                          getWidth()/2-yyoffset,
66                                        SCREEN_HEIGHT-PE_ARRAY[15].getHeight()*3/2,
                                          paint);
67          } }
68          if(szisTouch){                                           //绘制分数查看对话框
69               canvas.drawBitmap(OTHER_PIC_ARRAY[6],               //绘制背景
70                                  (SCREEN_WIDTH-OTHER_PIC_ARRAY[6].getWidth())/2,
71                                  (SCREEN_HEIGHT-OTHER_PIC_ARRAY[6].getHeight())/
                                    2,paint);
72               final float yoffset=OTHER_PIC_ARRAY[0].getHeight()*73/100;
73               for(int i=0;i<3;i++){                               //for 循环
74                    float xsStart=SCREEN_WIDTH/2;                  //获得 x 坐标
75                    float ysStart=(SCREEN_HEIGHT+OTHER_PIC_ARRAY[6].getHeight())/2-
76                                     yoffset*(3-i)+5*yMainRatio; //获得 y 坐标
77                    str=HH_SCORE[i]+"";                            //得到分数
78                    for(int j=0;j<str.length();j++){               //循环
79                         canvas.drawBitmap(NUM_ARRAY[str.charAt(j)-'0'],   //绘制分数
80                                        xsStart+j*NUM_ARRAY[0].getWidth(),
                                          ysStart,paint);
81    } } }
```

- 第 2～4 行为需要使用的矩阵 Matrix，主要是平移矩阵、缩放矩阵以及两个矩阵的组合矩阵。
- 第 5～14 行为根据 degreesFlag 是否为 true，设置平移矩阵或者缩放矩阵，然后绘制设置按钮的背景以及设置按钮上的齿轮。
- 第 16～28 行为点击“设置”按钮后，通过 clipRect 方法动态绘制弹出的子菜单选项。
- 第 29～41 行为弹出子菜单选项后，再次点击“设置”按钮，如果 xyoffset 的值小于半透明图片的高度，则通过 clipRect 方法动态绘制收回的子菜单选项，如果 xyoffset 值大于半透明图片的高度，则设置 szisTouch、isTouch 以及 degreesFlag 为 false。
- 第 42～45 行为保存画布状态，绘制设置按钮的背景以及设置按钮上的齿轮，并恢复画布状态。
- 第 46～58 行为恢复画笔，同时，如果 ckisTouch 为 true，设置 isTouch 为 true，并且在 zyoffset 小于游戏信息图片的宽度时，动态绘制游戏信息窗口，zyoffset 值大于或者等于游戏信息图片的宽度时，绘制整个游戏信息图片以及“关闭”按钮。
- 第 59～67 行为点击“关闭”按钮后，动态绘制游戏信息图片以及“关闭”按钮。
- 第 68～81 行为点击“分数信息”按钮后，绘制分数对话框以及各个关卡所得的分数。

（3）上述代码为 MainMenuView 类的框架，接下来将介绍自定义的绘制方法，该方法主要是在背景、草地以及地面平移时用，将下列代码插入到 MainMenuView 类框架中的第 48 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MainMenuView.java。

```
1    public void drawSelf(Canvas canvas,Paint paint,Bitmap bm,
2                         float xlocation,float ylocation,int flag){//自定义的绘制方法
3         if(xlocation-bm.getWidth()>=0){                  //判断 xlocation 是否大于图片的宽度
4              if(flag==0){                                //如果 flag 为 0
5                   bdxoffset=0;                           //将 bdxoffset 赋值为 0
6              }
```

```
7                if(flag==1){                                  //如果 flag 为 1
8                     bgxoffset=0;                             //将 bgxoffset 赋值为 0
9                }
10               xlocation=0;                                  //将 xlocation 赋值为 0
11               canvas.drawBitmap(bm, 0, ylocation, paint);     //绘制图形
12          }else if(xlocation>bm.getWidth()-SCREEN_WIDTH){      //xlocation 大于图片宽度与
                                                                 //屏幕宽度差
13               canvas.drawBitmap(bm, 0-xlocation, ylocation, paint);   //绘制前一张图片
14               canvas.drawBitmap(bm, bm.getWidth()-xlocation-XOFFSET, ylocation, paint);
15          }else{
16               canvas.drawBitmap(bm, 0-xlocation, ylocation, paint);   //绘制图形
17     } }
```

- 第 3～11 行首先判断 xlocation 是否大于图片的宽度，如果 xlocation 大于图片的宽度，则根据 flag 的值将对应的 bdxoffset 或者 bgxoffset 赋值为 0，然后将 xlocation 赋值为 0，最后绘制图形。
- 第 12～15 行首先判断 xlocation 是否大于图片宽度与屏幕宽度差，如果大于，则绘制前一张图像以及绘制该图像的开始部分。
- 第 16 行为上述两种条件均不成立后，调用的绘制方法。

（4）接下来将要介绍重写的 onToucheEvent 方法，该方法主要是监听屏幕的触控事件，并根据触控的范围执行不同的动作，将下列代码插入到 MainMenuView 类框架中的第 49 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MainMenuView.java。

```
1     public boolean onTouchEvent(MotionEvent e){          //设置监听
2          int currentNUm=e.getAction();
3          float x=e.getX();                                //得到触控点的 x 坐标
4          float y=e.getY();                                //得到触控点的 y 坐标
5          switch(currentNUm){
6               case MotionEvent.ACTION_DOWN:               //触控事件为按下
7                    if(x>LOCALTION_BUTTON[0][0]&&          //判断触控点的范围
8                         x<LOCALTION_BUTTON[0][0]+LOCALTION_BUTTON[0][2]&&
9                         y>LOCALTION_BUTTON[0][1]&&
10                        y<LOCALTION_BUTTON[0][1]+LOCALTION_BUTTON[0][3]){
11                        if(!isTouch){                     //判断设置按钮是否被按下
12                             flag=1;                      //flag 值为 1
13                             xyoffset=0;                  // xyoffset 设置为 0
14                        }else{                            //已经点击了一次，再次点击设置按钮
15                             flag=2;                      //flag 值为 2
16                             syoffset=0;                  // syoffset 设置为 0
17                   } }
18                   if(isTouch&&x>LOCALTION_BUTTON[1][0]&&     //点击的是声音按钮
19                        x<LOCALTION_BUTTON[1][0]+LOCALTION_BUTTON[1][2]&&
20                        y>LOCALTION_BUTTON[1][1]&&
21                        y<LOCALTION_BUTTON[1][1]+LOCALTION_BUTTON[1][3]){
22                        syisTouch=!syisTouch;                 //更改 syisTouch 值
23                   }else if(isTouch&&!szisTouch&&x>LOCALTION_BUTTON[2][0]&&
24                        x<LOCALTION_BUTTON[2][0]+LOCALTION_BUTTON[2][2]&&
25                        y>LOCALTION_BUTTON[2][1]&&            //点击的是游戏信息按钮
26                        y<LOCALTION_BUTTON[2][1]+LOCALTION_BUTTON[2][3]){
27                        ckisTouch=!ckisTouch;                 //更改 ckisTouch 值
28                        yyoffset=0;                           //yyoffset 设置为 0
29                        isClose=false;                        //isClose 设置为 false
30                   }else if(isTouch&&!ckisTouch&&x>LOCALTION_BUTTON[3][0]&&
31                        x<LOCALTION_BUTTON[3][0]+LOCALTION_BUTTON[3][2]&&
32                        y>LOCALTION_BUTTON[3][1]&&            //点击的是分数信息按钮
33                        y<LOCALTION_BUTTON[3][1]+LOCALTION_BUTTON[3][3]){
```

```
34                      szisTouch=!szisTouch;                    //更改 szisTouch 值
35                  }
36                  if(ckisTouch&&x>(PE_ARRAY[14].getWidth()-PE_ARRAY[15].getWidth()
                    /2)&&
37                  x<(PE_ARRAY[14].getWidth()+PE_ARRAY[15].getWidth()/2)&&
38                      y>(SCREEN_HEIGHT-PE_ARRAY[15].getHeight()*3/2)&&
39                      y<(SCREEN_HEIGHT-PE_ARRAY[15].getHeight()/2)){
                                                             //如果点击的是关闭按钮
40                      isClose=true;                        // isClose 设置为 true
41                      ckisTouch=false;                     // ckisTouch 设置为 false
42                      zyoffset=0;                          // zyoffset 设置为 0
43                  }
44                  if(!szisTouch&&x>LOCALTION_BUTTON[4][0]&& //点击的是 "PLAY" 按钮
45                      x<LOCALTION_BUTTON[4][0]+LOCALTION_BUTTON[4][2]&&
46                      y>LOCALTION_BUTTON[4][1]&&
47                      y<LOCALTION_BUTTON[4][1]+LOCALTION_BUTTON[4][3]){
48                      activity.hd.sendEmptyMessage(0);//调用 sendEmptyMessage 发送消息
49                      MAIN_DRAW_THREAD_FLAG=false; //MAIN_DRAW_THREAD_FLAG 为 false
50                      TJ_CONTROL_FLAG=false;       //设置 TJ_CONTROL_FLAG 为 false
51                  }
52              break;                                   //退出
53          }
54          return true;                                 //返回 true
55      }
```

- 第 2～4 行为得到当前触控的值，并获得触控点的 x 坐标以及 y 坐标。
- 第 6～17 行首先判断点击的是否为“设置”按钮，如果点击的是“设置”按钮，则根据 isTouch 的值为 flag 赋值，并将 xyoffset 或者 syoffset 设置为 0。
- 第 18～22 行首先判断点击的是否为声音按钮，如果点击的是“声音”按钮，则将声音标志位 syisTouch 置反。
- 第 23～29 行首先判断点击的是否为“查看游戏信息”按钮，如果点击的是“查看游戏信息”按钮，则将 ckisTouch 置反，并将 yyoffset 设置为 0，isClose 设置为 false。
- 第 30～35 行首先判断点击的是否为“分数查看”按钮，如果点击的是“分数查看”按钮，则将信息查看标志位 szisTouch 置反。
- 第 36～43 行首先判断点击的是否为“关闭游戏信息”按钮，如点击的是“关闭游戏信息”按钮，则将 isClose 设置为 true，ckisTouch 设置为 false，zyoffset 设置为 0。
- 第 44～51 行首先判断点击的是否为“PLAY”按钮，如果点击的是“PLAY”按钮，则通过 Handler 调用 sendEmptyMessage 发送消息，并将 MAIN_DRAW_THREAD_FLAG 与 TJ_CONTROL_FLAG 设置为 false

（5）最后将介绍自定义的刷帧方法，该方法主要是在刷帧线程中调用，将下列代码插入到 MainMenuView 类框架中的第 68 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MainMenuView.java。

```
1    public void repaint(){
2            SurfaceHolder holder=this.getHolder();      //获得 SurfaceHolder 的对象
3            Canvas canvas = holder.lockCanvas();        //锁住画布
4            try{
5                synchronized(holder){                   //锁住 holder
6                    onDraw(canvas);                     //绘制
```

```
7                 } }catch(Exception e){                          //捕获异常
8                      e.printStackTrace();                       //打印堆栈信息
9                 }finally{
10                     if(canvas != null){                        //判断 canvas 是否为空
11                         holder.unlockCanvasAndPost(canvas);  //为画布解锁
12    } } }
```

- 第 2～3 行为获得 SurfaceHolder 的对象，并通过 SurfaceHolder 对象锁住画布。
- 第 4～9 行为捕获异常，使用 synchronized 同步时必须捕获异常。
- 第 10～12 行为判断 canvas 是否为空，如果 canvas 不为空，则通过 SurfaceHolder 的对象调用 unlockCanvasAndPost 方法为画布解锁。

19.5.4　刷帧线程 MainMenuDrawThread

前一小节介绍的是主界面 MainMenuView，接下来介绍刷帧线程 MainMenuDrawThread。该类继承自 Thread，其主要作用是通过调用 MainMenuView 中的 repaint 方法刷新主界面，代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MainMenuDrawThread.java。

```
1     package com.bn.box2d.sndls;                                //声明包名
2     import static com.bn.box2d.sndls.Constant.*;               //静态导入
3     public class MainMenuDrawThread extends Thread{            //创建继承 Thread 的线程类
4          static float syoffset;                                //上移偏移量
5          static float xyoffset;                                //下移偏移量
6          static float zyoffset;                                //右移偏移量
7          static float yyoffset;                                //左移偏移量
8          static float bdxoffset;                               //地面、草地的位置
9          static float bgxoffset;                               //背景的位置
10         MainMenuView mmv;                                     //声明 MainMenuView 引用
11         float degrees;                                        //旋转角度
12         public MainMenuDrawThread(MainMenuView tjview){//构造器
13              this.mmv=tjview;                                 //初始化 mmv
14         }
15         public void run(){                            //继承 Thread 类需要重写的方法
16              while(MAIN_DRAW_THREAD_FLAG){            //循环标志位 MAIN_DRAW_THREAD_FLAG
17                   bdxoffset+=2f*yMainRatio;           //增加 bdxoffset 的值
18                   bgxoffset+=1f*yMainRatio;           //增加 bgxoffset 的值
19                   mmv.repaint();                      //重新绘制
20                   try{
21                        Thread.sleep(SLEEPTIME);       //休息 SLEEPTIME
22                   }catch(Exception e){                //捕获异常
23                        e.printStackTrace();           //打印堆栈信息
24                   }
25                   if(mmv.flag==1){                    //子菜单上移
26                        syoffset+=60*yMainRatio;       //改变 syoffset 的值
27                        degrees=(float)(Math.toDegrees(syoffset))/1.3f;
28                   }
29                   if(mmv.flag==2){                    //子菜单下移
30                        xyoffset+=60*yMainRatio;       //改变 xyoffset 的值
31                        degrees=(float)(Math.toDegrees(-xyoffset))/1.3f;
32                   }
33                   if(mmv.ckisTouch){                  //左移
34                        zyoffset+=30*yMainRatio;       //改变 zyoffset 的值
35                   }
36                   if(mmv.isClose){                    //如果点击关闭游戏信息按钮
37                        yyoffset+=30*yMainRatio;       //改变 yyoffset 的值
```

```
38                  }
39                  try{
40                      Thread.sleep(15);           //休息 15ms
41                  }catch(Exception e){            //捕获异常
42                      e.printStackTrace();        //打印堆栈信息
43   } } } }
```

- 第 4～11 行是线程类 MainMenuDrawThread 中成员变量的声明，主要有 syoffset（上移偏移量）、xyoffset（下移偏移量）、zyoffset（右移偏移量）、yyoffset（左移偏移量）、bdxoffset（地面、草地的位置）、bgxoffset（背景的位置）、mmv（MainMenuView 引用）以及 degrees（旋转角度）。
- 第 12～14 行为本类的构造器，在该构造器中是获得 MainMenuView 的对象。
- 第 17～19 行为增加 bdxoffset 的值，增加 bgxoffset 的值，并调用 MainMenuView 中的 repaint() 方法重新绘制。
- 第 25～28 行为点击“设置”按钮后子菜单上移，改变 syoffset 的值，同时根据 syoffset 值计算出对应的角度。
- 第 29～32 行为点击“设置”按钮后子菜单下移，改变 xyoffset 的值，同时根据 xyoffset 值计算出对应的角度。
- 第 36～38 行为点击“关闭游戏信息”按钮后，不断地改变 yyoffset 的值。

19.6 工具类

开发本游戏时用到了许多工具类，主要有加载及缩放图片的工具类 PicLoadUtil、生成刚体工具类 Box2DUtil 以及声音工具类 SoundUtil。在本节中将会对本游戏用到的工具类进行详细的介绍。

19.6.1 加载及缩放图片的工具类 PicLoadUtil

本类的主要作用是加载图片以及通过 Matrix 对象调用 postScale 方法缩放图片，代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 PicLoadUtil.java。

```
package com.bn.box2d.sndls;                                  //声明包名
import android.content.res.Resources;                        //导入相关包
......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
import android.graphics.Matrix;                              //导入相关包
public class PicLoadUtil{
     public static Bitmap loadBM(Resources res,int id){      //加载图片
          return BitmapFactory.decodeResource(res,id);       //返回 Bitmap 对象
     }
     public static Bitmap scaleToFitXYRatio(Bitmap bm,float xRatio,float yRatio){
                                                             //缩放图片的方法
          float width = bm.getWidth();                       //图片宽度
          float height = bm.getHeight();                     //图片高度
          Matrix m1 = new Matrix();                          //创建 Matrix 对象
          m1.postScale(xRatio, yRatio);                      //缩放图片
          Bitmap bmResult = Bitmap.createBitmap(bm, 0, 0, (int)width, (int)height,
          m1, true);                                         //声明位图
          return bmResult;                                   //返回 Bitmap 对象
} }
```

- 第 6～8 行为加载图片的方法，在该方法中主要通过 BitmapFactory 调用 decodeResource 方法加载本游戏中用到的图片。
- 第 9～16 行为缩放图片的方法，首先获得图片的宽度与高度，创建 Matrix 对象，并调用 postScale 缩放图片，然后通过 Bitmap 调用 createBitmap 方法声明位图，最后返回 Bitmap 的对象。

19.6.2　生成刚体工具类 Box2DUtil

前一小节介绍的是加载及缩放图片的工具类 PicLoadUtil，接下来介绍生成刚体的工具类 Box2DUtil。在该类中主要是生成对应的刚体，并设置刚体的具体属性，代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Box2DUtil.java。

```
1    package com.bn.box2d.sndls;                          //声明包名
2    import org.jbox2d.collision.PolygonDef;              //导入相关包
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import static com.bn.box2d.sndls.Constant.*;         //静态导入
5    public class Box2DUtil{                              //生成物理形状的工具类
6         public static MyPolygonImg createPolygonImg(//创建多边形（贴图）
7              float x,                                   //x 坐标
8              float y,                                   //y 坐标
9              float[][] vData,                           //顶点坐标
10             boolean isStatic,                          //是否为静止的标志位
11             World world,                               //世界类的引用
12             Bitmap[] bm,                               //图片
13             float width,                               //目标宽度
14             float height,                              //目标高度
15             BodyType lx,                               //类型
16             GameView gv                                //GameView 对象
17       ){
18             PolygonDef shape = new PolygonDef();       //创建多边形描述对象
19             if(isStatic){                              //判断 isStatic 标志位是否为 true
20                  shape.density = 0;                    //设置密度为 0
21             }else{
22                  shape.density = 2.0f;                 //设置密度为 2
23             }
24             shape.friction = 0.8f;                     //设置摩擦系数
25             shape.restitution = 0.5f;                  //设置能量损失率（反弹）
26             for(float[] fa:vData){                     //增强的 for 循环，循环遍历二维数组
27                  shape.addVertex(new Vec2(fa[0]/RATE,fa[1]/RATE));
28             }
29             BodyDef bodyDef = new BodyDef();               //创建刚体描述对象
30             bodyDef.position.set(x/RATE, y/RATE);          //设置位置
31             Body bodyTemp= world.createBody(bodyDef);      //在世界中创建刚体
32             bodyTemp.createShape(shape);                   //指定刚体形状
33             bodyTemp.setMassFromShapes();                  //设置质量
34             MyPolygonImg result=null;                      //创建 MyPolygonImg 引用为空
35             switch(lx){
36               case MT:                                     //木头条
37                 result=new BodyWood(bodyTemp,bm,width,height,gv);
38               break;                                       //退出
39               case BK:                                     //冰块
40                 result=new BodyIce(bodyTemp,bm,width,height,gv);
41               break;                                       //退出
42               case XM:                                     //猫头
43                 result=new BodyCat(bodyTemp,bm,width,height,gv);
44               break;                                       //退出
```

```
45                  default:                                        //其他
46                    result=new MyPolygonImg(bodyTemp,bm,width,height,gv);
47              }
48                  return result;                                  //返回值为 true
49    } }
```

- 第 7～16 行为该工具类中 createPolygonImg 方法的参数，主要有 *x*（x 坐标）、*y*（y 坐标）、vData（顶点坐标）、isStatic（是否为静止的标志位）、world（世界）、bm（图片）、width（目标宽度）、height（目标高度）、lx（类型）以及 gv（GameView 对象）。该方法主要是创建多边形，并设置多边形的属性。
- 第 18 行为通过 PolygonDef 创建多边形描述对象。
- 第 19～23 行为判断 isStatic 为 true 或为 false，如果为 true，则设置密度为 0，否则设置密度为 2。
- 第 24～28 行为设置摩擦系数与能量损失率，并循环遍历 vData 数组向多边形中添加顶点。
- 第 29～33 行为创建多边形描述对象，设置其位置、在世界中创建刚体、指定刚体形状以及设置刚体质量。
- 第 34～49 行为声明 MyPolygonImg 的引用，根据刚体的类型创建不同的刚体对象，并返回。

说明　本游戏中使用的物理引擎为 JBox2D，如果读者对本引擎不了解，可以查看本书中的第 10 章游戏的心脏——物理引擎，或者到相关的网站查阅资料。

19.6.3　声音工具类 SoundUtil

本小节中主要是初始化声音缓冲池，同时创建 playSound 方法，该方法根据参数的不同，使得在调用此方法时发出不同的声音，代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 SoundUtil.java。

```
1     package com.bn.box2d.sndls;                          //声明包名
2     import java.util.HashMap;                            //导入相关包
3     ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4     import android.media.SoundPool;                      //导入相关包
5     public class SoundUtil{
6          SoundPool soundPool;                            //创建 SoundPool 的引用
7          HashMap<Integer, Integer> soundPoolMap;//创建 HashMap<Integer, Integer>集合
8          MyBox2dActivity activity;                       //创建 MyBox2dActivity 的引用
9          public SoundUtil(MyBox2dActivity activity){     //构造器
10              this.activity=activity;                    //初始化 activity
11         }
12       public void initSounds(){                         //声音缓冲池的初始化
13             soundPool = new SoundPool(                  //创建声音缓冲池
14                    6,                                   //同时能最多播放的个数
15                    AudioManager.STREAM_MUSIC,           //音频的类型
16                    100                                  //声音的播放质量，目前无效
17             );
18             soundPoolMap = new HashMap<Integer, Integer>(); //创建声音资源 Map
19             soundPoolMap.put(0, soundPool.load(activity, R.raw.pijin, 1));
                                                                //皮筋拉伸的声音
20             soundPoolMap.put(1, soundPool.load(activity, R.raw.ls_fx, 1));
                                                                //鼠头飞
21             soundPoolMap.put(2, soundPool.load(activity, R.raw.wood, 1));     //木头
```

```
22            soundPoolMap.put(3, soundPool.load(activity, R.raw.boli, 1));     //玻璃
23            soundPoolMap.put(4, soundPool.load(activity, R.raw.stond, 1));    //石头
24            soundPoolMap.put(5, soundPool.load(activity, R.raw.cat, 1));      //猫头
25        }
26      public void playSound(int sound, int loop) {        //播放声音的方法
27        if(!activity.mmv.syisTouch){                      //是否点击了声音按钮
28          AudioManager mgr = (AudioManager)activity.getSystemService(Context.AUDIO_
            SERVICE);
29          float streamVolumeCurrent = mgr.getStreamVolume(AudioManager.STREAM_MUSIC);
                                                            //当前音量
30          float streamVolumeMax = mgr.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
                                                            //最大音量
31          float volume = streamVolumeCurrent / streamVolumeMax; //得到音量
32          soundPool.play(
33                  soundPoolMap.get(sound),                //声音资源 id
34                  volume,                                 //左声道音量
35                  volume,                                 //右声道音量
36                  1,                                      //优先级
37                  loop,                                   //循环次数 -1 代表永远循环
38                  1f                                      //回放速度 0.5f~2.0f 之间
39                );
40    } } }
```

- 第 6～8 行为该工具类 SoundUtil 中的成员变量，主要分为 SoundPool 的引用、HashMap<Integer, Integer>集合的引用以及 MyBox2dActivity 的引用。
- 第 9～11 行为本类的构造器，在该构造器中主要是初始化 MyBox2dActivity 的引用。
- 第 12～25 行为创建 initSounds 方法，该方法主要是初始化声音缓冲池，并将相应的声音添加到 HashMap<Integer, Integer>集合中。
- 第 26～40 行为创建播放声音的方法，在该方法中首先通过 MyBox2dActivity 的对象调用 getSystemService 方法，然后通过 AudioManager 获得当前音量以及最大音量，最后通过 SoundPool 的对象调用 play 方法播放声音。

19.7 刚体相关类

前一节介绍的是工具类，接下来介绍刚体相关类。主要有多边形类 MyPolygonImg、猫头类 BodyCat、冰块类 BodyCat、木条类 BodyWood、刚体类型声明类 BodyType，以及查询类 BodySearchUtil。其中 BodyCat、BodyCat 以及 BodyWood 表示的是具体的刚体形状。

19.7.1　多边形类 MyPolygonImg

首先介绍多边形类 MyPolygonImg，该类是猫头类 BodyCat、冰块类 BodyCat 以及木条类 BodyWood 的父类，该类中主要有自定义的绘制方法 drawSelf 以及执行动作的方法 doAction，其代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 MyPolygonImg.java。

```
1    package com.bn.box2d.sndls;                                    //声明包名
2    import org.jbox2d.dynamics.Body;                               //导入相关包
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
```

```
4    import static com.bn.box2d.sndls.Constant.*;          //静态导入
5    public class MyPolygonImg{                            //自定义的多边形类（图片）
6         Bitmap bma[];                                    //用于绘制的图片
7         int bmIndex=0;                                   //当前图片索引
8         Body body;                                       //对应物理引擎中的刚体
9         float width;                                     //目标宽度
10        float height;                                    //目标高度
11        float xScale;                                    //x 缩放值
12        float yScale;
13        boolean isLive=true;                             //是否存在
14        boolean isDeleted=false;                         //是否已经删除
15        GameView gv;                                     //GameView 的引用
16        long timeStamp=0;                                //时间间隔
17        public MyPolygonImg(Body body,Bitmap[] bma,float width,float height,GameView gv){
18             this.body=body;                             //初始化 body
19             this.bma=bma;                               //初始化 bma
20             this.width=width;                           //初始化宽度
21             this.height=height;                         //初始化高度
22             xScale=width/bma[0].getWidth();             //初始化 xScale
23             yScale=height/bma[0].getHeight();           //初始化 yScale
24             this.gv=gv;                                 //初始化 GameView 对象
25        }
26        public void drawSelf(Canvas canvas,Paint paint){//自定义绘制方法
27             if(!isLive){return;}                        //判断 isLive 是否为 false
28             float x=body.getPosition().x*RATE;          //得到屏幕像素 x 值
29             float y=body.getPosition().y*RATE;          //得到屏幕像素 y 值
30             float angle=body.getAngle();                //得到刚体的角度
31           canvas.save();                                //保存画布状态
32           Matrix m1=new Matrix();                       //创建 Matrix 对象
33           m1.setRotate((float)Math.toDegrees(angle),x+xOffset, y+yOffset);
34           canvas.setMatrix(m1);                         //设置矩阵
35           Matrix m2=new Matrix();                       //创建 Matrix 对象
36           m2.setScale(xScale, yScale);                  //调用 setScale 方法缩放
37           Matrix m3=new Matrix();                       //创建 Matrix 对象
38           m3.setTranslate(x+xOffset, y+yOffset);        //调用 setTranslate 方法平移
39           Matrix mz=new Matrix();                       //创建 Matrix 对象
40           mz.setConcat(m3, m2);
41           canvas.drawBitmap(bma[bmIndex], mz, paint);   //绘制图形
42         canvas.restore();                               //恢复画布状态
43        }
44        public void doAction(float x,float y){           //碰撞后执行不同的动作
45             if(body.isDynamic()&&gv.hero!=this){
46                  long tempST=System.currentTimeMillis();   //得到系统的时间(毫秒)
47                  if(tempST-timeStamp>800){                 //判断时间差是否大于 800ms
48                       gv.activity.soundutil.playSound(4, 0);   //调用 playSound
                                                                  //方法，发出声音
49                       timeStamp=tempST;                        //设置 timeStamp 的值
50   } } } }
```

- 第 6～16 行为本类成员变量的声明，主要有 bma（用于绘制的图片）、bmIndex（图片索引）、body（物理引擎中的刚体）、width（目标宽度）、height（目标高度）、xScale（x 缩放值）、yScale（y 缩放值）、isLive（是否存在）、isDeleted（是否已经删除）、gv（GameView 的引用）以及 timeStamp（时间间隔）。
- 第 17～25 行为本类的构造器，在该构造器中主要是初始化相应的成员变量。
- 第 26～43 行为自定义的绘制方法，在该方法中首先判断 isLive 是否为 false，如果为 false，则直接返回。获得屏幕上实际的 *x*,*y* 坐标，并通过创建 Matix 对象，平移图像以及缩放图像，调用 drawBitmap 方法绘制图形。

- 第 44～50 行为自定义的碰撞后执行不同动作的方法 doAction。

19.7.2　猫头类 BodyCat

前一小节介绍的是多边形类 MyPolygonImg，接下来将介绍猫头类 BodyCat，其代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 BodyCat.java。

```
1    package com.bn.box2d.sndls;                                    //声明包名
2    import static com.bn.box2d.sndls.Constant.*;                   //静态导入
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import android.graphics.Paint;                                 //导入相关类
5    public class BodyCat extends MyPolygonImg{                     //猫头
6         static int YZ1=3;                                         //域值 1
7         static int YZ2=28;                                        //域值 2
8         static int YZ3=53;                                        //域值 3
9         static int YZ4=78;                                        //域值 4
10        public BodyCat(Body body,Bitmap[] bma,float width,float height,GameView gv){
11             super(body,bma,width,height,gv);                     //调用父类
12        }
13        int count=0;                                              //计数为 0
14        public void doAction(float x,float y){                    //碰撞后执行不同的动作
15             if(count==0){                                        //count 为 0
16                  SCORE=SCORE+100;                                //分数增加 100
17                  gv.scoreList.add(new Score(Y_NUM_ARRAY,100,x,y));
18             }
19             count++;                                             //count 值加 1
20        }
21        public void drawSelf(Canvas canvas,Paint paint) {        //自定义的绘制方法
22             super.drawSelf(canvas, paint);                       //调用父类
23             if(count>0){                                         //count 大于 0
24                  count++;                                        //count 值加 1
25             }
26             float x=body.getWorldCenter().x*RATE;                //得到屏幕的 x 坐标
27             float y=body.getWorldCenter().y*RATE;                //得到屏幕的 y 坐标
28             if(count>=YZ2&&count<=YZ3&&bmIndex==1){
29                  SCORE=SCORE+100;                                //分数增加 100
30                  gv.scoreList.add(new Score(Y_NUM_ARRAY,100,x,y));
31             }else if(count>=YZ3&&count<YZ4&&bmIndex==2){//count 在域值的某一范围之内
32                  SCORE=SCORE+100;                            //分数增加 100
33                  gv.scoreList.add(new Score(Y_NUM_ARRAY,100,x,y));
34             }else if(count>YZ4&&this.isLive){               // count 在域值的某一范围之内
35                  this.isLive=false;                          //isLive 的值设置为 false
36                  SCORE=SCORE+5000;                              分数增加 5000
37                  gv.scoreList.add(new Score(Y_NUM_ARRAY,5000,x,y));
38                  long tempST=System.currentTimeMillis();//得到系统的时间(毫秒)
39                  if(tempST-timeStamp>1000){                  //时间差大于 1000ms
40                       gv.activity.soundutil.playSound(5, 0);//调用 SoundUtil 中的
                                                                 //playSound 方法
41                       timeStamp=tempST;                          //设置 timeStamp 的值
42   } } } }
```

- 第 6～9 行为本类的域值的声明。
- 第 14～20 行为继承 MyPolygonImg 类重写的 doAction 方法，在该方法中首先判断 count 是否为 0，如果为 0，则将 SCORE 加 100，创建 Score 对象并添加进 List<Score>集合中。
- 第 21～42 行为继承 MyPolygonImg 类重写的 drawSelf 方法，该方法首先判断 count 的值是否大于零，如果大于零，则 count 加 1。然后获取屏幕上的 x 坐标与 y 坐标，最后根据 count 所处的

域值范围，使得 SCORE 加上对应的分数，生成 Score 对象并存入 List<Score>集合。

19.7.3 冰块类 BodyIce

前一小节介绍的是猫头类，在本小节将介绍冰块类，该类继承自 MyPolygonImg，并重写的 doAction 方法以及 drawSelf 方法，其代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 BodyIce.java。

```
1    package com.bn.box2d.sndls;                                    //声明包名
2    import static com.bn.box2d.sndls.Constant.*;                   //静态导入
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import android.graphics.Paint;                                 //导入相关类
5    public class BodyIce extends MyPolygonImg{
6         static int YZ1=3;                                         //域值 1
7         static int YZ2=18;                                        //域值 2
8         static int YZ3=33;                                        //域值 3
9         static int YZ4=48;                                        //域值 4
10        public BodyIce(Body body,Bitmap[] bma,float width,float height,GameView gv){
11             super(body,bma,width,height,gv);                     //调用父类
12        }
13        int count=0;                                              //count 为 0
14        public void doAction(float x,float y){                    //碰撞后执行不同的动作
15             count++;                                             //count 值加 1
16             if(bmIndex==0){                                      //判断 bmIndex 的值是否为 0
17                  bmIndex=1;                                      //初始化 bmIndex 的值为 1
18                  SCORE=SCORE+200;                                //分数增加 200
19                  gv.scoreList.add(new Score(Y_NUM_ARRAY,200,x,y));
20        } }
21        public void drawSelf(Canvas canvas,Paint paint){          //自定义的绘制方法
22             super.drawSelf(canvas, paint);                       //调用父类
23             if(count>0){                                         //count 大于 0
24                  count++;                                        //count 值加 1
25             }
26             float x=body.getWorldCenter().x*RATE;            //得到屏幕的 x 坐标
27             float y=body.getWorldCenter().y*RATE;            //得到屏幕的 y 坐标
28             if(count>=YZ2&&count<=YZ3&&bmIndex==1){          //count 在域值的某一范围之内
29                  bmIndex=2;                                  //bmIndex 的值为 2
30                  SCORE=SCORE+300;                            //分数增加 300
31                  gv.scoreList.add(new Score(Y_NUM_ARRAY,300,x,y));
32             }else if(count>=YZ3&&count<YZ4&&bmIndex==2){//count 在域值的某一范围之内
33                  bmIndex=3;                                  //bmIndex 的值为 3
34                  SCORE=SCORE+300;                            //分数增加 300
35                  gv.scoreList.add(new Score(Y_NUM_ARRAY,300,x,y));
36             }else if(count>YZ4&&this.isLive){                //count 在域值的某一范围之内
37                  this.isLive=false;                          //设置 isLive 为 false
38                  SCORE=SCORE+500;                            //分数增加 500
39                  gv.scoreList.add(new Score(Y_NUM_ARRAY,500,x,y));
40                  long tempST=System.currentTimeMillis();     //得到系统的时间(毫秒)
41                  if(tempST-timeStamp>800){                   //时间差大于 800ms
42                       gv.activity.soundutil.playSound(3, 0);   //调用 SoundUtil 中的
                                                                  //playSound 方法
43                       timeStamp=tempST;                        //设置 timeStamp 的值
44   } } } }
```

- 第 6～9 行为本类的域值的声明。
- 第 14～20 行为继承 MyPolygonImg 类重写的 doAction 方法，在该方法中首先使得 count 值加 1，然后判断 dmIndex 的值是否为 0，如果为 0，则初始化 bmIndex 的值为 1，使得 SCORE 值加

200，同时将生成的 Score 对象存入 List<Score>集合中。

- 第 21～44 行为继承 MyPolygonImg 类重写的 drawSelf 方法，该方法首先调用父的方法并判断 count 的值是否大于零，如果大于零，则 count 加 1。然后获取屏幕上的 x 坐标与 y 坐标，最后根据 count 所处的域值范围，使得 SCORE 加上对应的分数，生成 Score 对象并存入 List<Score>集合。

19.7.4　木条类 BodyWood

前一小节介绍了冰块类 BodyIce，接下来在本小节将介绍木条类 BodyWood，该类继承自 MyPolygonImg，同时重写的 doAction 方法，其代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 BodyWood.java。

```
1    package com.bn.box2d.sndls;                                  //声明包名
2    import org.jbox2d.dynamics.Body;                             //导入相关类
3    import android.graphics.Bitmap;                              //导入相关类
4    import static com.bn.box2d.sndls.Constant.*;                 //静态导入
5    public class BodyWood extends MyPolygonImg{                  //木头条
6         static int YZ1=3;                                       //域值 1
7         static int YZ2=6;                                       //域值 2
8         public BodyWood(Body body,Bitmap[] bma,float width,float height,GameView gv){
9              super(body,bma,width,height,gv);                   //调用父类
10        }
11        int count=0;                                            //count 为 0
12        public void doAction(float x,float y){                  //碰撞后执行不同的动作
13             count++;                                           //count 值加 1
14             if(count>YZ1&&count<YZ2&&bmIndex==0){              //count 在域值的某一范围之内
15                  bmIndex=1;                                    //bmIndex 的值为 1
16                  SCORE=SCORE+200;                              //分数增加 200
17                  gv.scoreList.add(new Score(P_NUM_ARRAY,200,x,y));
18             }else if(count>=YZ2&&bmIndex==1){                  //count 在域值的某一范围之内
19                  bmIndex=2;                                    //bmIndex 的值为 2
20                  SCORE=SCORE+200;                              //分数增加 200
21                  gv.scoreList.add(new Score(P_NUM_ARRAY,200,x,y));
22                  long tempST=System.currentTimeMillis();   //得到系统的时间(毫秒)
23                  if(tempST-timeStamp>800){                 //时间差大于 1800ms
24                       gv.activity.soundutil.playSound(2, 0);   //调用 SoundUtil 中的
                                                                  //playSound 方法
25                       timeStamp=tempST;                        //设置 timeStamp 的值
26   } } } }
```

- 第 6～7 行为本类域值的声明。
- 第 8～9 行为本类的构造器，在该构造器中主要是调用父类。
- 第 11 行为将 count 值赋值为 0。
- 第 12～26 行为继承 MyPolygonImg 类后重写的 doAction 方法，该方法首先将 count 值加 1，然后根据 count 所处的域值范围，使得 SCORE 加上对应的分数，生成 Score 对象并存入 List<Score>集合。最后，如果时间间隔大于 800 毫秒，则调用 SoundUtil 中的 playSound 方法播放声音。

19.7.5　刚体类型枚举类 BodyType

前四小节已经介绍了多边形类 MyPolygonImg，及其子类 BodyCat、BodyCat 以及 BodyWood，接下来在本节中将介绍刚体类型的声明类 BodyType，该类中主要存储的是刚体的类型，其代码

如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 BodyType.java。

```
1    package com.bn.box2d.sndls;                          //声明包名
2    public enum BodyType{
3       LS_ZY,                                            //睁眼鼠头
4       DM,                                               //地面
5       ST,                                               //石条
6       MT,                                               //木条
7       XM,                                               //猫头
8       BK,                                               //冰块
9       HNTZ,                                             //灰泥台子
10   }
```

说明 该类中声明了游戏场景中用到的刚体的类型，主要有 LS_ZY（睁眼鼠头）、DM（地面）、ST（石条）、MT（木条）、XM（猫头）、BK（冰块）以及 HNTZ（灰泥台子）。

19.7.6 刚体查询工具类 BodySearchUtil

本节主要介绍查询类 BodySearchUtil，该类主要是根据 START_SCORE 为 true 或者 false，执行不同的动作，具体代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 BodySearchUtil.java。

```
1    package com.bn.box2d.sndls;                          //声明包名
2    import java.util.ArrayList;                          //导入相关类
3    import org.jbox2d.dynamics.Body;                     //导入相关类
4    import static com.bn.box2d.sndls.Constant.*;         //静态导入
5    public class BodySearchUtil{
6         public static void doAction(GameView gv,Body body1,Body body2,
7                                     ArrayList<MyPolygonImg> bl,float x,float y){

8              if(gv.hero==null){                         //hero 为空
9                   return;                               //返回
10             }
11             Body heroBody=gv.hero.body;                //创建 Body 对象
12             for(MyPolygonImg mpi:bl){                  //增强 for 循环
13                  if(!START_SCORE){                     //判断 START_SCORE 是否为 false
14                       if(body1==heroBody||body2==heroBody){ //判断 body1 与 body2 是否
                                                           //符合条件
15                            START_SCORE=true;            // START_SCORE 为 true
16                  } }
17                  if(START_SCORE){                      // START_SCORE 为 true 时
18                       if(body1==mpi.body||body2==mpi.body){//判断 body1 与 body2 是否
                                                           //符合条件
19                            mpi.doAction(x,y);           //调用 doAction 方法
20   } } } } }
```

- 第 8～11 行为判断 hero 是否为空，如果为空，则直接返回；否则，通过 hero 创建 Body 对象。
- 第 12～20 行为增强的 for 循环，首先判断 START_SCORE 是否为 false，如果为 false，就判断 body1 与 body2 是否符合条件，符合条件，则设置 START_SCORE 为 true；如果 START_SCORE

为 true，则通过 MyPolygonImg 类的对象调用 doActioin 方法。

19.8 游戏界面相关类

本节将介绍游戏界面相关类，涉及的类主要有皮筋类 Pijin、记录分数类 Score、刷帧线程 DrawThread 以及游戏界面类 GameView，其中 GameView 类中加载游戏数据、添加碰撞监听、绘制本游戏中游戏界面以及添加触控监听。

19.8.1 皮筋类 Pijin

首先介绍游戏界面相关类中的皮筋类，本游戏中弹弓上的皮筋是根据鼠头的位置，动态地在游戏界面上绘制的，其具体代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Pijin.java。

```
1    package com.bn.box2d.sndls;                                  //声明包名
2    import static com.bn.box2d.sndls.Constant.*;                 //静态导入
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import android.graphics.Paint;                               //导入相关类
5    public class Pijin{
6         public static float[][] lcon={                          //弹弓上两个拴住皮筋的位置坐标
7              {54,20},                                           //弹弓左侧
8              {93,20} };                                         //弹弓右侧
9         float lx;                                               //鼠头的 X 坐标
10        float ly;                                               //鼠头的 Y 坐标
11        float dx;                                               //弹弓上拴住皮筋的 X 坐标
12        float dy;                                               //弹弓上拴住皮筋的 Y 坐标
13        static boolean flag=false;                              //设置 flag 为 false
14        public Pijin(float lx,float ly,float dx,float dy){   //构造器
15             this.lx=lx;                                        //初始化鼠头的 X 坐标
16             this.ly=ly;                                        //初始化鼠头的 Y 坐标
17             this.dx=dx;                                  //初始化弹弓上拴住皮筋的 X 坐标
18             this.dy=dy;                                  //初始化弹弓上拴住皮筋的 Y 坐标
19        }
20        public float[] getDegrees(float arg0,float arg1,float arg2,float arg3){
21             float[] degrees=new float[2];                //创建二维数组
22             float yoffset=arg3-arg1;                     //得到 y 轴偏移量
23             float xoffset=arg2-arg0;                     //得到 x 轴偏移量
24             degrees[0]=(float) Math.sqrt(xoffset*xoffset+yoffset*yoffset);
25             degrees[1]=(float) Math.toDegrees(Math.asin(yoffset/degrees[0]));
26             return degrees;                              //返回转动的度数
27        }
28        public void drawSelf(Canvas canvas,Paint paint){//自定义的绘制方法
29             if(lx<150*yMainRatio&&!flag){                //判断鼠头的坐标是否符合条件
30                  float[] degrees=getDegrees(lx,ly,dx,dy);  //得到转动的度数
31                  setMatrix(canvas,paint,degrees);          //调用 setMatrix 方法
32             }else{
33                  flag=true;                                //flag 值为 true
34                  float[] degrees=getDegrees(lx,ly,dx,dy);  //得到转动的度数
35                  setMatrix(canvas,paint,degrees);          //调用 setMatrix 方法
36        } }
37        public void setMatrix(Canvas canvas,Paint paint,float[] degrees){//设置矩阵的旋转
38             canvas.save();                                    //保存画布状态
39             Matrix m1=new Matrix();                           //平移矩阵
40             m1.setTranslate(lx+PIC_ARRAY[1].getWidth(), ly+PIC_ARRAY[1].getHeight()/2);
41             Matrix m2=new Matrix();                           //旋转矩阵
```

```
42              m2.setRotate(degrees[1]);                          //设置旋转一定的角度
43              Matrix m3=new Matrix();                            //创建 Matrix 对象
44              m3.setConcat(m1, m2);                              //组合两个矩阵
45              Matrix m4=new Matrix();                            //创建 Matrix 对象
46              float xrotate=(float) (degrees[0])/PIC_ARRAY[24].getWidth();
47              m4.setScale(xrotate, 1);                           //设置缩放
48              Matrix m5=new Matrix();                            //创建 Matrix 对象
49              m5.setConcat(m3, m4);                              //组合矩阵
50              canvas.drawBitmap(PIC_ARRAY[24], m5, paint);       //绘制图像
51              canvas.restore();                                  //恢复画布状态
52    } }
```

- 第 6～13 行为本类成员变量的声明，主要有 lcon（弹弓上两个拴住皮筋的位置坐标）、lx（鼠头的 X 坐标）、ly（鼠头的 Y 坐标）、dx（弹弓上拴住皮筋的 X 坐标）、dy（弹弓上拴住皮筋的 Y 坐标）以及 flag（设置 flag 为 false）。
- 第 14～19 行为本类的构造器，在该构造器中主要是初始化鼠头的 x 坐标、y 坐标、初始化链接弹弓的橡皮筋的 x 坐标以及 y 坐标。
- 第 20～27 行为获得皮筋转动度数的方法，在该方法中，首先创建一个二维数组，然后得到 y 轴偏移量以及 x 轴的偏移量，最后通过 Math 调用 toDegrees 方法获取转动的角度。
- 第 37～52 行为设置矩阵旋转的方法，在该方法中首先需要保存画布状态，然后创建 Matrix 对象，并调用 setTranslate 设计平移，调用 setRotate 设置旋转，调用 setConcat 组合两矩阵，调用 setScale 设置缩放，并绘制图像，最后恢复画布状态。

19.8.2 记录分数类 Score

前一小节介绍的是皮筋类 Pijin，接下来介绍记录分数的类 Score。该类主要是在鼠头撞击冰块、木条以及猫头时调用绘制得分，其代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 Score.java。

```
1     package com.bn.box2d.sndls;                                //声明包名
2     import android.graphics.Bitmap;                            //导入相关类
3     ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4     import static com.bn.box2d.sndls.Constant.*;               //静态导入
5     public class Score{                                        //得分类
6        Bitmap[] bmNum;                                         //创建 Bitmap 数组
7        char[] score;                                           //创建分数数组
8        float x;                                                //成员变量 x 坐标
9        float y;                                                //成员变量 y 坐标
10       int count=0;                                            //设置 count 值为 0
11       final float YZ=25;                                      //设置域值为 25
12       public Score(Bitmap[] bmNum,int score,float x,float y){//构造器
13            this.bmNum=bmNum;                                  //初始化 Bitmap 数组
14            this.score=(score+"").toCharArray();               //初始化分数数组
15            this.x=x;                                          //初始化 x 坐标
16            this.y=y;                                          //初始化 y 坐标
17       }
18       public void drawSelf(Canvas canvas,Paint paint){        //自定义绘制方法
19            if(count>YZ){                                      //count 值大于域值
20                 return;                                       //直接返回
21            }else{
22                 count++;                                      //count 值加 1
23            }
24            int alpha=(int) (255-255.0f*(count/YZ));           //设置透明度
```

```
25          if(alpha<=0){                                          //如果透明度 alpha 值小于 0
26               alpha=0;                                          //设置 alpha 值为 0
27          }
28          paint.setAlpha(alpha);                                 //设置画笔透明度
29          float width=score.length*bmNum[0].getWidth();          //得到宽度
30          float realX=x-width/2;                                 //得到 x 坐标
31          float realY=y-bmNum[0].getHeight()/2;                  //得到高度
32          for(int i=0;i<score.length;i++){                       //for 循环
33             int index=score[i]-'0';                             //设置 index 值
34             canvas.drawBitmap(bmNum[index],   xOffset+realX+i*bmNum[0].getWidth(),
               yOffset+realY,paint);
35          }
36          paint.reset();                                         //恢复画笔设置
37    } }
```

- 第 6～11 行为本类的成员变量的声明，主要有 bmNum（创建 Bitmap 数组）、score（创建分数数组）、x（成员变量 x 坐标）、y（成员变量 y 坐标）、count（设置 count 值为 0）以及 YZ（设置域值为 25）。
- 第 12～17 行为本类的构造器，在该构造器中主要是初始化 Bitmap 数组、分数数组以及 x,y 坐标。
- 第 19～23 行为判断 count 值是否大于域值，如果大于域值则直接跳转，否则 count 值加 1。
- 第 24～28 行为根据 count 与 YZ 的大小设置透明度 alpha，alpha 大于等于零，并设置画笔透明度。
- 第 29～36 行为根据分数的宽度与高度，计算出分数左上角的 x，y 坐标，循环遍历该分数，并绘制分数，最后需要恢复画笔设置。

19.8.3　刷帧线程 DrawThread

本小节主要介绍的是线程 DrawThread，该类的主要作用是每隔固定的时间定时地调用 repaint 方法重新绘制界面，还有判断是否过关以及是否输掉本局。

（1）首先介绍线程类 DrawThread 的框架，了解该框架有助于读者对本类整体结构的理解，该类的框架代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 DrawThread.java。

```
1    package com.bn.box2d.sndls;                    //声明包名
2    import static com.bn.box2d.sndls.Constant.*;//静态导入
3    import org.jbox2d.common.Vec2;                 //导入相关类
4    import android.graphics.Bitmap;                //导入相关类
5    public class DrawThread extends Thread{   //绘制线程
6         GameView gv;                         //GameView 的引用
7         boolean addTask=false;               //boolean 值为 false
8         float xst;                           //x 坐标
9         float yst;                           //y 坐标
10        boolean flag=false;                  //过关后绘制对话框的标志位
11        boolean levelFlag;                   //level_cleared 与 level_Failure 的标志位
```

```
12          public DrawThread(GameView gv){        //构造器
13               this.gv=gv;                        //获得 GameView 的对象
14          }
15          public void run(){                      //继承 Thread 类需要重写的 run()方法
16               while(DRAW_THREAD_FLAG){           //while 循环
17                    if(!flag){                    //判断 flag 是否为 false
18                         if(addTask){             //判断 addTask 是否为 true
19                              doAddtask();        //调用 doAddtask 方法
20                              addTask=false;      //设置 addTask 为 false
21                    } }
22                    gv.repaint();                 //调用 repaint 方法刷帧
23                    if(!flag){                    //判断 flag 是否为 false
24                         gv.world.step(TIME_STEP, ITERA);      //开始模拟
25                    }
26                    try{
27                         Thread.sleep(10);                     //休息 10ms
28                    }catch(InterruptedException e){            //捕获异常
29                         e.printStackTrace();                  //打印堆栈信息
30                    }
31                    if(!flag){                                 //判断 flag 是否为 false
32                         for(MyPolygonImg mpi:gv.bl){          //增强 for 循环
33                              if(!mpi.isLive&&!mpi.isDeleted){//判断是否符合条件
34                                   mpi.isDeleted=true;         //设置 isDeleted 为 true
35                                   gv.world.destroyBody(mpi.body);
36                    } } }
37                    if(!flag){                                 //判断是否过关
38                         if(SCORE>=GG_SCORE[currStage]){       //判断是否符合条件
39                              if(SCORE>HH_SCORE[currStage]){   //判断是否符合条件
40                                   HH_SCORE[currStage]=SCORE;
41                              }
42                              flag=true;                        //flag 设置为 true
43                              levelFlag=true;                   //levelFlag 设置为 true
44                    } }
45                    if(START_SCORE){                  //判断是否输了
46                         boolean isStop=true;         //设置 isStop 为 true
47                         for(MyPolygonImg mpi:gv.bl){
                                                        //循环遍历 ArrayList<MyPolygonImg>集合
48                              if(mpi.isLive&&mpi.body.getLinearVelocity().length-
                                Squared()>0.1f){
49                                   isStop=false;               //设置 isStop 为 false
50                                   break;                      //退出
51                         } }
52                         if(isStop){                     //判断 isStop 是否为 true
53                              flag=true;                 //设置 flag 为 true
54                              levelFlag=false;           //设置 levelFlag 为 false
55                    } }
56                    try{
57                         Thread.sleep(5);               //线程休眠 5ms
58                    }catch (InterruptedException e){ //捕获异常
59                         e.printStackTrace();           //打印堆栈信息
60          }
61          ……//该处省略了 doAddtask 方法，将在下面给出
62     } } }
```

- 第 6～11 行为本类的成员变量的声明，主要有 gv（GameView 的引用）、addTask（boolean 值为 false）、xst(x 坐标)、yst(y 坐标)、flag(过关后绘制对话框的标志位)以及 levelFlag(level_cleared 与 level_Failure 的标志位)。
- 第 12～14 为本类的构造器，在该构造器中主要是初始化 GameView 对象。

- 第 17～21 行为判断 flag 是否为 false，如果为 false，就判断 addTask 是否为 true，如果 addTask 为 true，则调用 doAddtask 方法，并设置 addTask 为 false。
- 第 23～25 行为判断 flag 是否为 false，如果为 false，则调用 Word（世界）中的 step 方法开始模拟。
- 第 31～36 行为判断各个刚体是否已经不存在，如果不存在，则设置 isDeleted 为 true，并通过 World 类的对象调用 destroyBody 方法删除对应刚体。
- 第 37～44 行为判断分数是否达到过关的要求，如果达到过关的要求，判断是不是本关的最高分，如果是，则存入当前数组。同时设置 flag 设置为 true 及 levelFlag 设 true。
- 第 45～55 行为判断是否未通过本关，如果满足其中的条件，则说明输掉本关，设置 isStop 为 false 并退出；如果 isStop 为 true，则设置 flag 设置为 true，并将 levelFlag 设置为 false。

（2）接下来将介绍 doAddtask 方法，该方法主要是创建鼠头刚体，设置其速度，然后将创建的刚体添加进 ArrayList<MyPolygonImg>集合中，将下列代码插入到 DrawThread 类框架的第 61 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 DrawThread.java。

```
1    public void doAddtask(){
2          Bitmap[] bma=new Bitmap[1];                    //添加鼠头刚体
3          bma[0]=PIC_ARRAY[1];                           //为 bma 添加数据
4          float[][] dataF=new float[][]{                 //包住鼠头的多边形坐标
5               {16,0},{32,4},{40,20},{32,38},{8,38},{0,28},{2,17},{8,5}
6          };
7          for(int i=0;i<dataF.length;i++){               //循环遍历 dataF 数组
8               dataF[i][0]=dataF[i][0]*yMainRatio;       //顶点 x 坐标
9               dataF[i][1]=dataF[i][1]*yMainRatio;       //顶点 y 坐标
10         }
11         gv.hero=Box2DUtil.createPolygonImg(  //调用 Box2DUtil 中的 createPolygonImg 方法
12              xst,                            //x 坐标
13              yst,                            //y 坐标
14              dataF,                          //顶点坐标
15              false,                          //是否为静止的标志位
16              gv.world,                       //世界
17              bma,                            //图片
18              42*yMainRatio,                  //目标宽度
19              42*yMainRatio,                  //目标高度
20              BodyType.LS_ZY,                 //类型
21              gv                              //GameView 的对象
22        );
23        float sdx=gv.ysXst-xst;               //x 偏移量
24        float sdy=gv.ysYst-yst;               //y 偏移量
25        double len=Math.sqrt(sdx*sdx+sdy*sdy);          //获得偏移的距离
26        final float maxSpeed=50;                        //声明最大速度
27        float vx=(float) (sdx*maxSpeed/len);            //得到 x 方向速度
28        float vy=(float) (sdy*maxSpeed/len);            //得到 y 方向速度
29        gv.hero.body.setLinearVelocity(new Vec2(vx,vy)); //设置刚体的质心线速度
30        gv.bl.add(gv.hero);                   //将刚体添进 ArrayList<MyPolygonImg>集合
31   }
```

- 第 2～6 行为该方法中局部变量的声明，主要有 bma（添加鼠头刚体）和 dataF（鼠头的顶点坐标）。
- 第 7～10 行为循环遍历 dataF 数组，根据缩放比例 yMainRatio 缩放坐标。

- 第 11～22 行为通过调用 Box2DUtil 中的 createPolygonImg 方法得到鼠头刚体。
- 第 23～30 行为计算 x 和 y 的偏移量，设置最大速度，计算 x 方向的速度和 y 方向的速度，设置刚体的速度，并将刚体添进 ArrayList<MyPolygonImg>集合。

19.8.4 游戏界面类 GameView

本小节将介绍游戏中最重要的类——GameView，在该类中主要是调用 loadGameData 加载游戏数据、initContactListener 添加碰撞监听、onDraw 绘制本游戏中游戏界面以及调用 onTouchEvent 为该游戏界面添加触控监听。

（1）首先介绍该游戏界面类 GameView 的框架结构，了解该框架有助于读者对本类整体结构的理解，该类的框架代码如下。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 GameView.java。

```
1    package com.bn.box2d.sndls;                          //声明包名
2    import static com.bn.box2d.sndls.Constant.*;//静态导入
3    ......//该处省略了部分导入相关类的代码，读者可以自行查阅随书光盘的源代码
4    import android.view.SurfaceView;                     //导入相关类
5    public class GameView extends SurfaceView implements SurfaceHolder.Callback{
6         MyBox2dActivity activity;                       //声明 MyBox2dActivity 的引用
7         Paint paint;                                    //画笔
8         DrawThread dt;                                  //绘制线程
9         AABB worldAABB;                                 //创建一个管理碰撞的世界
10       World world;                                     //声明 World 类的引用
11       Pijin pjc;                                       //皮筋
12       String hightScore;                               //最高分
13       float x_pj;                                      //最后选择关卡菜单减去两个按钮后的平均值
14       ArrayList<MyPolygonImg> bl=new ArrayList<MyPolygonImg>(); //物体列表
15       MyPolygonImg hero;                                   //英雄
16       List<Score> scoreList=new ArrayList<Score>();//得分列表
17       boolean flagSf=false;                                //是否已经释放鼠头的标志位
18       float xst=180*yMainRatio;                            //鼠头初始位置
19       float yst=SCREEN_HEIGHT-DMGD-PIC_ARRAY[22].getHeight();
20       final float ysXst=xst;                               //x 轴偏移量
21       final float ysYst=yst;                               //y 轴偏移量
22         public GameView(MyBox2dActivity activity){ //构造器
23              super(activity);                            //调用父类
24              this.activity = activity;                   //初始化 MyBox2dActivity 的引用
25              this.loadGameData();                        //调用 loadGameData()方法
26              this.initContactListener();                 //调用 initContactListener()方法
27              this.getHolder().addCallback(this);         //设置生命周期回调接口的实现者
28              paint = new Paint();                        //创建画笔
29              paint.setAntiAlias(true);                   //打开抗锯齿
30              dt=new DrawThread(this);                    //启动绘制线程
31              dt.start();                                 //开启线程
32         }
33         ......//该处省略了加载游戏数据的代码，将在后面给出
34         ......//该处省略了加载碰撞监听器的代码，将在后面给出
35         ......//该处省略了绘制方法的代码，将在后面给出
36         ......//该处省略了重写的 onTouchEvent 方法的代码，将在后面给出
37         public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3){ }
38         public void surfaceCreated(SurfaceHolder holder){    //创建时被调用
39              repaint();                                      //调用 repaint 方法
40         }
41         public void surfaceDestroyed(SurfaceHolder arg0){ } //需要重写的方法
```

```
42        public void repaint(){                                  //自定义的 repaint 方法
43             SurfaceHolder holder=this.getHolder();             //得到 SurfaceHolder 的对象
44             Canvas canvas = holder.lockCanvas();               //获取画布
45             try{
46                  synchronized(holder){                         //同步
47                       onDraw(canvas);                          //绘制
48             } }catch(Exception e){                             //捕获异常
49                  e.printStackTrace();                          //打印堆栈信息
50             }finally{                                          //最后需要执行的
51                  if(canvas != null){                           //判断 canvas 是否为空
52                       holder.unlockCanvasAndPost(canvas);  //解锁画布
53   } } } }
```

- 第 6～13 行为本类的部分成员变量，主要有 activity（声明 MyBox2dActivity 的引用）、paint（画笔）、dt（绘制线程）、worldAABB（创建一个管理碰撞的世界）、world（World 类的引用）、pjc（皮筋）、hightScore（最高分）以及 x_pj（最后选择关卡菜单减去两个按钮后的平均值）。
- 第 14～21 行为本类的部分成员变量，主要有 bl（物体列表）、hero（英雄）、scoreList（得分列表）、flagSf（是否已经释放鼠头的标志位）、xst（鼠头初始 x 坐标）、yst（鼠头初始 y 坐标）、ysXst（x 轴偏移量）以及 ysYst（y 轴偏移量）。
- 第 22～32 行为本类的构造器，在该构造器中主要是初始化 MyBox2dActivity 的引用、调用 loadGameData()方法加载游戏数据、调用 initContactListener()方法加载碰撞监听器、设置生命周期回调接口的实现者、创建画笔、打开抗锯齿并开启绘制线程。
- 第 38～40 行为实现 SurfaceHolder.Callback 接口需要重写的 surfaceCreated(SurfaceHolder holder)方法，该方法在进入界面时执行，同时在该方法中调用 repaint 方法重新绘制界面。
- 第 42～53 行为自定义的 repaint 方法，在该方法中首先获得 SurfaceHolder 的对象、获取画布、调用 onDraw 方法绘制图形并在最后解锁画布。

（2）接下来将介绍加载游戏数据的方法——loadGameData，在该方法中主要是加载游戏中用到的刚体，将下列代码插入到 GameView 类框架的第 33 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 GameView.java。

```
1     public void loadGameData(){
2          worldAABB = new AABB();                                  //创建 AABB 对象
3        worldAABB.lowerBound.set(0f,-100.0f);                      //设置包围盒子
4        worldAABB.upperBound.set(200.0f, 100.0f);
5        Vec2 gravity = new Vec2(0.0f,20.0f);                       //设置重力加速度
6        boolean doSleep = true;                                    //设置 doSleep 为 true
7        world = new World(worldAABB, gravity, doSleep);            //创建世界
8        for(int i=0;i<STONE_VERTEX[currStage].length;i++){         //创建场景中的物体
9            Bitmap[] bma=new Bitmap[IMG_ID[currStage][i].length];  //创建 Bitmap 一维数组
10           for(int j=0;j<IMG_ID[currStage][i].length;j++){        //循环遍历 IMG_ID 数组
11                   bma[j]=PIC_ARRAY[IMG_ID[currStage][i][j]];
                                                  //将遍历所得数据添加进 Bitmap 数组中
12           }
13           MyPolygonImg myBodyTemp=Box2DUtil.createPolygonImg(
14               LOCATION[currStage][i][0],      //x 坐标
15               LOCATION[currStage][i][1],      //y 坐标
16               STONE_VERTEX[currStage][i],     //顶点坐标
17               IS_MOVE[currStage][i],          //是否为静止的标志位
18               world,                          //世界
19               bma,                            //图片
```

```
20                SIZE[currStage][i][0],          //目标宽度
21                SIZE[currStage][i][1],          //目标高度
22                typeA[currStage][i],            //类型
23                this                            //GameView 的对象
24            );
25            myBodyTemp.body.setLinearVelocity  //设置质心线速度
26                    (new Vec2(SPEED[currStage][i][0],SPEED[currStage][i][1]));
27            bl.add(myBodyTemp);                //数据添加进 ArrayList<MyPolygonImg>集合
28    } }
```

- 第 2～7 行首先创建 AABB 对象，并设置包围盒子、重力加速度以及 doSleep 标志位，最后需要创建 World 类的对象。
- 第 9～12 行为循环遍历 IMG_ID 数组，并将遍历所得数据添加进 Bitmap 数组中。
- 第 13～24 行为通过调用 Box2DUtil 类中的 createPolygonImg 方法创建要绘制游戏界面中的刚体。
- 第 25～27 行为设置质心线速度，并将 MyPolygonImg 对象添加进 ArrayList<MyPolygonImg>集合。

（3）接下来将介绍加载碰撞监听器的方法——initContactListener，在该方法中主要是为世界添加监听器，将下列代码插入到 GameView 类框架的第 34 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 GameView.java。

```
1     public void initContactListener(){
2          ContactListener cl=new ContactListener(){   //匿名内部类
3               public void add(ContactPoint arg0){    //需要重写的 add 方法
4                    BodySearchUtil.doAction(  //从当前的 body 查找出对应物件并执行事件处理
5                         GameView.this,               //GameView 的对象
6                         arg0.shape1.getBody(),       //刚体 1
7                         arg0.shape2.getBody(),       //刚体 2
8                         bl,                          // ArrayList<MyPolygonImg>集合
9                         arg0.position.x*RATE,        //x 坐标
10                        arg0.position.y*RATE         //y 坐标
11              ); }
12              public void persist(ContactPoint arg0){}   //需要重写的 persisit 方法
13              public void remove(ContactPoint arg0){}    //需要重写的 remove 方法
14              public void result(ContactResult arg0){}   //需要重写的 result 方法
15         };
16         world.setContactListener(cl);                   //为世界添加监听器
17    }
```

- 第 3～11 行为创建 ContactListener 匿名内部类需要重写的方法，在该方法中主要是调用 BodySearchUtil 中的 doAction 方法从当前的 body 查找出对应实体并执行事件处理。
- 第 12～14 行为创建 ContactListener 匿名内部类需要重写的 persisit、remove 以及 result 方法。
- 第 16 行为通过 World 类的对象调用 setContactListener，主要是为世界添加监听器。

（4）接下来将介绍绘制本游戏场景的方法——onDraw，该方法主要是绘制背景、草地、地面、木条、石条、冰块以及猫头等，将下列代码插入到 GameView 类框架的第 35 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 GameView.java。

```
1     public void onDraw(Canvas canvas){
2          if(canvas==null){                          //判断 canvas 是否为空
```

```
3            return;                                          //直接返回
4        }
5        if(hero!=null){                                      //判断 hero 是否为空
6            float hx=hero.body.getPosition().x*RATE;         //得到实际屏幕中的 x 位置
7            float tempXOffset=-(hx-SCREEN_WIDTH/2);          //得到实际屏幕中的 y 位置
8            if(tempXOffset>0){                               //tempXOffset 大于零
9                tempXOffset=0;                               //将 tempXOffset 设置为 0
10           }else if(tempXOffset<SCREEN_WIDTH-CJ_WIDTH){//tempXOffset 值小于屏幕宽度
11               tempXOffset=SCREEN_WIDTH-CJ_WIDTH;           //将 tempXOffset 设置为屏幕宽度
12           }
13           if(tempXOffset<xOffset){                         //tempXOffset 小于偏移量
14               xOffset=tempXOffset;                         //xOffset 的职位 tempXOffset
15       } }
16       canvas.save();                                       //保存画布状态
17       canvas.clipRect(new RectF(0,0,SCREEN_WIDTH,SCREEN_HEIGHT));
18       canvas.drawBitmap(PIC_ARRAY[17],0+xOffset*0.6f, 0, paint);   //绘制背景
19       canvas.drawBitmap(PIC_ARRAY[17],0+xOffset*0.6f+PIC_ARRAY[17].getWidth(), 0,
         paint);
20       canvas.drawBitmap(PIC_ARRAY[26],0+xOffset,           //绘制背景草
21                         SCREEN_HEIGHT-DMGD-PIC_ARRAY[26].getHeight(), paint);
22       canvas.drawBitmap(PIC_ARRAY[26],0+xOffset+PIC_ARRAY[26].getWidth(),
23                         SCREEN_HEIGHT-DMGD-PIC_ARRAY[26].getHeight(), paint);

24       for(MyPolygonImg mb:bl){                     //遍历 ArrayList<MyPolygonImg>集合
25           mb.drawSelf(canvas, paint);              //绘制场景中的物体
26       }
27       String scoreStr=SCORE+"";                    //绘制总得分
28       float xsStart=SCREEN_WIDTH-NUM_ARRAY[0].getWidth()*(scoreStr.length()+1);
29       float ysStart=NUM_ARRAY[0].getHeight()/2;                    //得到 y 坐标
30       for(int i=0;i<scoreStr.length();i++){                        //循环所得分数
31           canvas.drawBitmap(NUM_ARRAY[scoreStr.charAt(i)-'0'],     //绘制分数
32                             xsStart+i*NUM_ARRAY[0].getWidth() ,ysStart, paint);
33       }
34       if(!flagSf){                                 //绘制拖动鼠头以及皮筋
35           pjc=new Pijin(xst-PIC_ARRAY[1].getWidth(),yst,180*yMainRatio,
36                         SCREEN_HEIGHT-DMGD-PIC_ARRAY[22].getHeight());
37           pjc.drawSelf(canvas, paint);             //绘制橡皮筋
38           canvas.drawBitmap(PIC_ARRAY[1], xst+xOffset,yst, paint);
                                                      //绘制被手拖拉的鼠头
39           pjc=new  Pijin(xst-PIC_ARRAY[1].getWidth(),yst,180*yMainRatio-PIC_ARRAY[1].
             getWidth(),
40                         SCREEN_HEIGHT-DMGD-PIC_ARRAY[22].getHeight());
41           pjc.drawSelf(canvas, paint);             //绘制橡皮筋
42       }
43       if(flagSf&&!Pijin.flag){                     //松开皮筋
44           pjc=new Pijin(hero.body.getPosition().x*RATE-PIC_ARRAY[1].getWidth(),
45                         hero.body.getPosition().y*RATE,180*yMainRatio,
46                         SCREEN_HEIGHT-DMGD-PIC_ARRAY[22].getHeight());
47           pjc.drawSelf(canvas, paint);             //绘制橡皮筋
48           pjc=new Pijin(hero.body.getPosition().x*RATE-PIC_ARRAY[1].getWidth(),
49                         hero.body.getPosition().y*RATE,180*yMainRatio-PIC_
                           ARRAY[1].getWidth(),
50                         SCREEN_HEIGHT-DMGD-PIC_ARRAY[22].getHeight());
51           pjc.drawSelf(canvas, paint);             //绘制橡皮筋
52       }
53       if(flagSf&&Pijin.flag&&xOffset<=0&&xOffset>-190*yMainRatio){
                                                      //绘制链接弹弓两侧的橡皮筋
54           pjc=new Pijin(180*yMainRatio-PIC_ARRAY[1].getWidth()+xOffset,
55                 SCREEN_HEIGHT-DMGD-PIC_ARRAY[22].getHeight(),180*yMainRatio+
                   xOffset,
56                 SCREEN_HEIGHT-DMGD-PIC_ARRAY[22].getHeight());
57           pjc.drawSelf(canvas, paint);             //绘制橡皮筋
```

```
58          }
59          ......//该处省略了部分绘制的代码，读者可以自行查阅随著光盘中的源代码
60          for(Score sc:scoreList){                    //循环遍历 ArrayList<Score>集合
61               sc.drawSelf(canvas, paint);            //绘制小的分列表
62          }
63          canvas.restore();                           //恢复画布状态
64          if(dt.flag==true){                          //判断 flag 是否为 true
65               ......//该处省略了绘制对话框的代码，读者可以自行查阅随著光盘中的源代码
66     } } }
```

- 第 2～4 行为判断 canvas 是否为空，如果为空，则直接返回。
- 第 5～15 行为判断 hero 是否为空，如果不为空，则根据刚体的位置计算出鼠头的 x 坐标以及 y 坐标，并为 xOffset 赋值。
- 第 17～26 行为绘制背景、草以及由此场景中所用到的刚体，如猫头、鼠头、石条等。
- 第 27～33 行为根据所得的分数，在屏幕的左上角绘制关卡所得分数。
- 第 34～42 行为手指拖动鼠头时，根据鼠头的位置绘制橡皮筋以及鼠头。
- 第 43～52 行为松开手之后，动态的绘制橡皮筋。
- 第 53～58 行为鼠头在弹弓右侧后，绘制链接弹弓两侧的橡皮筋。
- 第 60～62 行为循环遍历 ArrayList<Score>集合并绘制小的分列表。

（5）最后将介绍屏幕触控监听方法——onTouchEvent，在该方法中主要是在对应的位置添加触控监听，将下列代码插入到 GameView 类框架的第 36 行。

代码位置：见随书光盘中源代码/第 19 章/Box2DSNDLS/src/com.bn.box2d.sndls 目录下的 GameView.java。

```
1     boolean isTouched=false;                                  //是否摸到了鼠头
2     float startX=0;                                           //开始的 x 坐标
3     float startY=0;                                           //开始的 y 坐标
4     public boolean onTouchEvent(MotionEvent e){               //onTouchEvent 方法
5          if(!flagSf){                                         //判断 flagStr 是否为 false
6               int action=e.getAction();
7               float x=e.getX();                               //得到触控点的 x 坐标
8               float y=e.getY();                               //得到触控点的 y 坐标
9               float w=PIC_ARRAY[1].getWidth();                //得到图片宽度
10              float h=PIC_ARRAY[1].getHeight();               //得到图片高度
11              switch(action){
12                case  MotionEvent.ACTION_DOWN:                //为按下事件
13                     if(x>xst&&x<xst+w&&y>yst&&y<yst+h){      //如果点击的是鼠头
14                          if(!isTouched){                     //判断 isTouched 是否为 true
15                               activity.soundutil.playSound(0, 0);
                                                               //调用 playSound 方法开启声音
16                          }
17                          isTouched=true;                     //设置 isTouched 为 true
18                          startX=x;                           //设置 startX 的值
19                          startY=y;                           //设置 startY 的值
20                     }
21                break;                                         //退出
22                case  MotionEvent.ACTION_MOVE:                 //触控事件为移动
23                     if(isTouched){                           //判断 isTouched 是否为 true
24                          float dx=x-startX;                  //x 方向位移
25                          float dy=y-startY;                  //y 方向位移
26                          final float xYZ=-80*yMainRatio;//x 方向向左移动最大的距离
27                          final float yYZ=60*yMainRatio;//y 方向移动最大的距离
28                          if(dx>0){                           //dx 大于 0 时
```

```
29                          dx=0;                          //dx 赋值为 0(不可以向右拖动鼠头)
30                      }else if(dx<xYZ){                  //dx 的值小于 x 方向移动最大的距离
31                          dx=xYZ;                        //为 dx 赋值为 xYZ
32                      }
33                      if(dy>yYZ){                        //dy 的值大于 y 方向移动最大的距离
34                          dy=yYZ;                        //dy 值为 yYZ
35                      }else if(dy<-yYZ){                 //dy 的值向上移动至最大距离
36                          dy=-yYZ;                       //dy 的职位-yYZ
37                      }
38                      xst=ysXst+dx;                      //得到鼠头的 x 坐标
39                      yst=ysYst+dy;                      //得到鼠头的 y 坐标
40                  }
41              break;                                     //退出
42              case  MotionEvent.ACTION_UP:               //触控事件为抬起
43                  if(isTouched){                         //判断 isTouched 是否为 true
44                      flagSf=true;                       //添加鼠头刚体
45                      dt.xst=xst;                        //为 xst 赋值
46                      dt.yst=yst;                        //为 yst 赋值
47                      dt.addTask=true;                   //设置 addTask 为 true
48                      activity.soundutil.playSound(1, 0);//调用 playSound 开启声音
49              }
50              break;                                     //退出
51          }
52          return true;                                   //返回值为 true
53      }else{
54          int action=e.getAction();                      //得到动作的编号
55          float x=e.getX();                              //触控点的 x 坐标
56          float y=e.getY();                              //触控点的 y 坐标
57          float w=OTHER_PIC_ARRAY[4].getWidth();         //对话框的宽度
58          float h=OTHER_PIC_ARRAY[4].getHeight();        //对话框的高度
59          switch(action){                                //判断触控事件
60              ......//该处省略了输赢对话框的触控事件的代码，读者可以自行查阅随书光盘中的源代码
61      } }
62      return true;
63  }
```

- 第 1～3 行为设置 isTouched 为 false，startX 的值为 0 以及 startY 的值为 0。
- 第 6～10 行为获得触控事件的编号、触控点的坐标以及鼠头的宽度与高度。
- 第 12～21 行为 flagSf 为 false 时，触控事件为按下，通过 SoundUtil 的对象调用 playSound 方法开启声音，设置 isTouched 为 true、startX 为当前点的 x 坐标及 startY 为当前点的 y 坐标。
- 第 22～41 行为 flagSf 为 false 时，触控事件为拖动。根据设定鼠头可以拖动的范围，计算出鼠头被拖动后的坐标值。
- 第 42～50 行为 flagSf 为 false 时，触控事件为抬起。首先需要判断 isTouched 是否为 true，如果为 true，则设置 flagSf 为 true、初始化 xst 与 yst 的值、设置 addTask 为 true 并通过 SoundUtil 的对象调用 playSound 方法开启声音。
- 第 54～58 行为得到触控事件的编号、触控点的坐标以及鼠头的宽度与高度。

19.9 游戏优化与改进

到本节为止，本游戏的基本功能已经介绍完毕。但是本游戏仍然存在许多需要改进的地方，有兴趣的读者可以继续深入地开发本游戏，增强游戏的可玩性。本游戏可以改进的地方主要有以下几

点。

- 物体碰撞时的音效

由于本游戏只是书中的一个案例，因此在搜集音效时并不是做的很好。有兴趣的读者可以到网上搜集一些音效作为本游戏中的碰撞音效，从而大大增加游戏的可玩性。

- 缩放特效

《愤怒的小鸟》的游戏界面可以缩放，而本游戏的游戏界面并不可以。有兴趣的读者可以到网上查阅相关资料，自行开发该特效。

- 游戏未过关的对话框

开发完本游戏的过关对话框后，笔者再次开发未过关的对话框时并未开发新的对话框，只是对过关对话框做了些修改，读者可以改善该对话框的样式。

- 游戏分享

读者可以开发游戏分享模块，将玩游戏的成绩或者心得发到“新浪微博”、“facebook”以及“twitter”中，其他人可以看到自己的成绩或者心得。