Broadview
www.broadview.com.cn

疯狂 Android讲义

李刚 编著

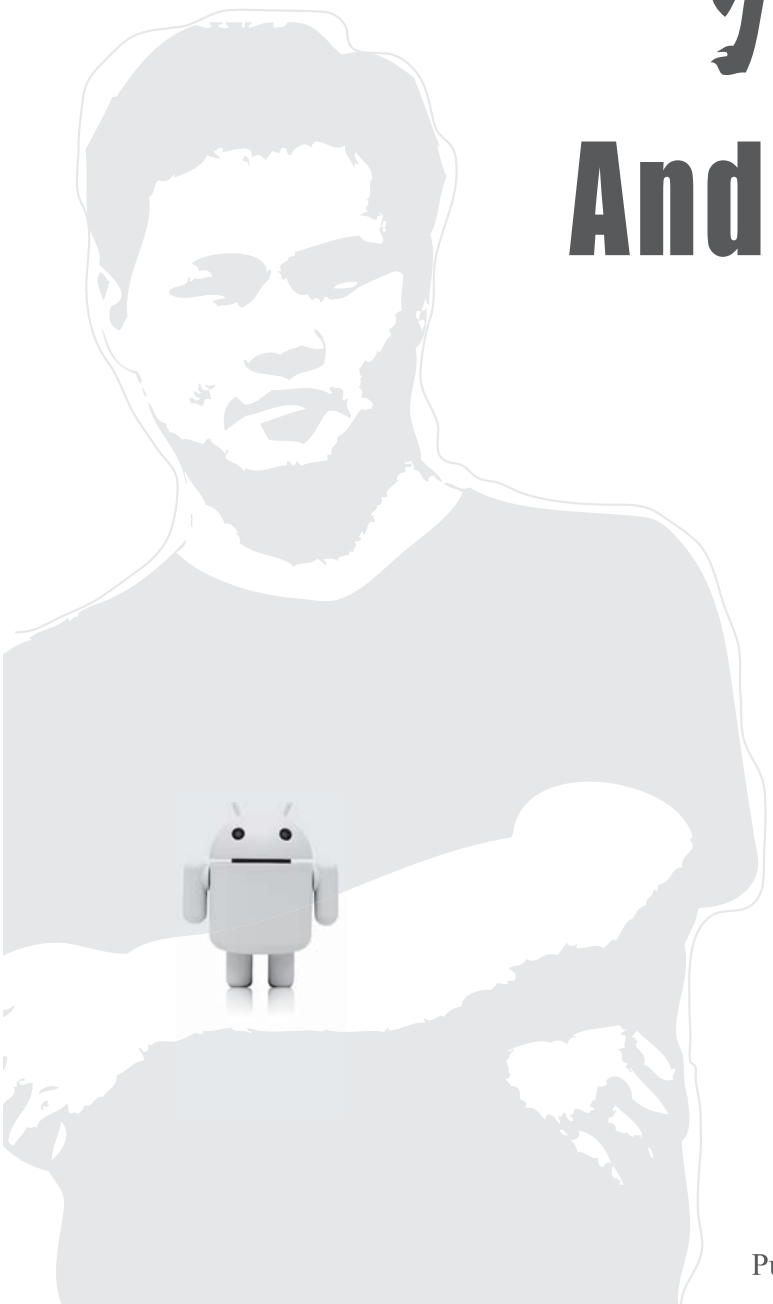
疯狂源自梦想
技术成就辉煌

疯狂源自梦想
技术成就辉煌

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

疯狂 Android讲义

李 刚 编著



電子工業出版社
Publishing House of Electronics Industry
北京 BEIJING

内 容 简 介

计算机便携化是未来的发展趋势，而 Android 作为最受欢迎的手机、平板电脑操作之一，其发展的上升势头是势不可当的。而 Android 应用选择了 Java 作为其开发语言，对于 Java 来说也是一次极好的机会。

本书全面地介绍了 Android 应用开发的相关知识，全书内容覆盖了 Android 用户界面编程、Android 四大组件、Android 资源访问、图形/图像处理、事件处理机制、Android 输入/输出处理、音频/视频多媒体应用开发、OpenGL 与 3D 应用开发、网络通信编程、Android 平台的 Web Service、传感器应用开发、GPS 应用开发、Google Map 服务等。

本书并不局限于介绍 Android 编程的各种理论知识，而是从“项目驱动”的角度来讲授理论，全书一共包括近百个实例，这些示范性的实例既可帮读者更好地理解各知识点在实际开发中的应用，也可供读者在实际开发时作为参考、拿来就用。本书最后还提供了两个实用的案例：疯狂连连看和电子拍卖系统 Android 客户端，具有极高的参考价值。本书提供了配套的答疑网站，如果读者在阅读本书时遇到了技术问题，可以登录疯狂 Java 联盟（http://www.crazyit.org）发帖，笔者将会及时予以解答。

本书适合于有一定 Java 编程基础的读者。如果读者已熟练掌握 Java 编程语法并具有一定图形界面编程经验，阅读本书将十分合适。否则，阅读本书之前建议先认真阅读疯狂 Java 体系之《疯狂 Java 讲义》。

图书在版编目（CIP）数据

疯狂 Android 讲义 / 李刚编著. —北京：电子工业出版社，2011.7
ISBN 978-7-121-13576-7

Ⅰ. ①疯… Ⅱ. ①李… Ⅲ. ①移动电话机－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2011）第 090681 号

策划编辑：张月萍
责任编辑：高洪霞
印　　刷：北京东光印刷厂
装　　订：三河市皇庄路通装订厂
出版发行：电子工业出版社
　　地址：北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：44　字数：1103 千字　彩插：1
印　　次：2011 年 7 月第 1 次印刷
印　　数：4000 册　　定价：89.00 元（含光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

计算机便携化是一种趋势。在人们的习惯里，很容易把电脑理解成主机、显示器、键盘的“组合”，即使后来出现了笔记本电脑，其实依然脱不了主机、显示器与键盘的组合。对于这种传统的电脑，用户必须“安静”地坐下来，打开计算机，然后才能使用计算机。但用户并不能完全满足通过这种方式使用电脑，有时用户需要在车上查看、管理公司的运营状况，有时用户需要在等飞机时查看、管理自己的证券交易情况，有时用户需要随时玩玩游戏松弛神经……在这些需求场景下，用户需要更加便携化的计算机，这也是目前智能手机、平板电脑大行其道的重要原因。

Android 系统就是一个开发式的手机和平台电脑的操作系统，目前的发展势头十分迅猛。虽然 Android 面世的时间不长，但 Android 已经对传统的手机平台（如 Symbian）构成了强大冲击，最近 Nokia 宣布与 Microsoft 结盟，可能会逐步采用 Windows Phone 来代替自己的 Symbian 系统，不过业界大多并不看好这次结盟，因为 Windows 的手机操作系统本身并不太受市场欢迎。业界部分人士预测，Android 将会成为应用最为广泛的手机操作系统。

对于 Java 语言而言，Android 系统给了 Java 一个新的机会。在过去的岁月中，Java 语言作为服务器端编程语言，已经取得了极大的成功，Java EE 平台发展得非常成熟，而且一直是电信、移动、银行、证券、电子商务应用的首选平台、不争的王者。但在客户端应用开发方面，Java 语言一直表现不佳，虽然 Java 既有 AWT/Swing 界面开发库，也有 SWT/JFace 界面开发库，但对于客户端应用开发人员而言，大多不愿意选择 Java 语言。Android 系统的出现改变了这种局面。Android 是一个非常优秀的手机、平板电脑操作系统，它将会逐渐蚕食传统的桌面操作系统，而 Android 平台应用的开发语言就是 Java，这意味着 Java 语言将可以在客户端应用开发上大展拳脚。

随着 Android 平台在市场占有率上的稳步上升，采用 Java 语言开发的 Android 应用会越来越多。不过需要指出的是，运行 Android 平台的硬件只是手机、平台电脑等便携式设备，这些设备的计算能力、数据存储能力都是有限的，不太可能在 Android 平台上部署大型企业级应用，因此 Android 应用可能以纯粹客户端应用的角色出现，然后通过网络与传统大型应用交互，充当大型企业应用的客户端，比如现在已经出现的淘宝 Android 客户端、赶集网 Android 客户端，它们都是这种发展趋势下的产物。

对于 Java 开发者来说，以前主要在 Java EE 平台上从事服务器端应用开发，但在计算机便携化的趋势下，Java 开发者必然面临着为这些应用开发客户端的需求。对于 Java 开发者来说，Android 应用开发既是一个挑战，也是一个机遇——挑战是：掌握 Android 应用开发需要重新投入学习成本；机遇是：掌握 Android 开发之后将可让职业生涯达到一个新的高度，而且 Android 系统是一个新的发展趋势，这必然带来更多的就业机会与创业机会，这都值得当下的开发者好好把握。

本书有什么特点

本书是一本介绍 Android 应用开发的实用图书，全面介绍了 Android 2.3 平台上应用开发各方面的知识。与市面上有些介绍 Android 编程的图书不同，本书并没有花太多篇幅介绍 Android 的发展历史（因为这些内容到处都是），完全没有介绍 Android 市场（因为它只是一个交易网站，与 Android 开发无关，但有些图书甚至用整整一章来介绍它），也没有介绍 JDK 安装、环境变量配置等内容——笔者假设读者已经具有一定的 Java 功底。换句话来说，如果你对 JDK 安装、Java 基本语法还不熟，本书并不适合你。

本书只用了一章来介绍如何搭建 Android 开发环境、Android 应用结构，当然也简要说明了 Android 的发展历史。可能依然会有人觉得本书篇幅很多，这是由于本书覆盖了 Android 开发绝大部分知识，而且很多知识不仅介绍了相应的理论，并通过相应的实例程序给出了示范。

需要说明的是，本书只是一本介绍 Android 实际开发的图书，这不是一本关于所谓“思想”的书，不要指望学习本书能提高你所谓的“Android 思想”，所以奉劝那些希望提高编程思想的读者不要阅读本书。

本书更不是一本看完之后可以“吹嘘、炫耀”的书——因为本书并没有堆砌一堆“深奥”的新名词、一堆“高深”的思想，本书保持了“疯狂 Java 体系”的一贯风格：操作步骤详细、编程思路清晰，语言平实。只要读者有基本的 Java 基础，阅读本书不会有任何问题，看完本书不会让你觉得自己突然“高深”了，“高深”到自己都理解不了。

认真看完本书、把书中所有示例都练习一遍，本书带给你的只是 9 个字：“看得懂、学得会、做得出”。本书不能让你认识一堆新名词，只会让你学会实际的 Android 应用开发。

如果读者有非常扎实的 Java 基本功、良好的英文阅读能力，而且对图形用户界面编程也有丰富的经验，不管是 AWT/Swing 编程的经验，还是 SWT 编程的经验，抑或是 Windows 界面编程的经验都行，那没有多大必要购买本书，只要花几天时间快速浏览本书即可动手编程了。如果遇到某个类、某个功能不太确定，直接查阅 Android Dev Guide 和 API 参考文档即可。

不管怎样，只要读者在阅读本书时遇到知识上的问题，都可以登录疯狂 Java 联盟（http://www.crazyit.org）与广大 Java 学习者交流，笔者也会通过该平台与大家一起交流、学习。

本书还具有如下几个特点。

1．知识全面，覆盖面广

本书深入阐述了 Android 应用开发的 Activity、Service、BroadcastReceiver 与 ContentProvider 四大组件，并详细介绍了 Android 全部图形界面组件的功能和用法，Android 各种资源的管理与用法，Android 图形、图像处理，事件处理，Android 输入/输出处理，视频/音频等多媒体开发，OpenGL-ES 开发，网络通信，传感器和 GPS 开发等内容，全面覆盖 Android 官方指南，在某些内容上更加具体、深入。

2．内容实际，实用性强

本书并不局限于枯燥的理论介绍，而是采用了“项目驱动”的方式来讲授知识点，全书包括近百个实例，几乎每个知识点都可找到对应的参考实例。本书最后还提供了“疯狂连连

看”、“电子拍卖系统 Android 客户端”两个应用，具有极高的参考价值。

3．讲解详细，上手容易

本书保持了“疯狂 Java 体系”的一贯风格：操作步骤详细、编程思路清晰，语言平实。只要读者有一定的 Java 编程基础，阅读本书将可以很轻松地上手 Android 应用开发；学习完本书最后的两个案例后，读者即可完全满足实际企业中 Android 应用开发的要求。

光盘说明

1．光盘内容

光盘中的代码按章节存放，即第 2 章、第 2 节所使用的代码放在 02\2.2 文件夹下，依此类推。另外，书中每份源代码也给出与光盘源文件的对应关系，方便读者查找。

本光盘中有 19 个文件夹，其内容和含义说明如下：

（1）文件夹名 01～19 对应于书中的章号，即第 2 章所使用的代码放在 02 文件夹下，依此类推。

（2）10 文件夹下有 HRSystem 和 HRSystem_Eclipse 两个文件夹，它们是同一个项目的源文件，其中 HRSystem 是 IDE 平台无关的项目，使用 Ant 来编译即可；而 HRSystem_Eclipse 是该项目在 Eclipse IDE 工具中的项目文件。

（3）本书的绝大部分项目都是 Eclipse 项目，因此项目文件夹下包含.classpath、.project 等文件，它们是 Eclipse 项目文件，请不要删除。

2．运行环境

本书中的程序在以下环境调试通过：

（1）安装 jdk-6u22-windows-i586-p.exe，安装完成后，添加 CLASSPATH 环境变量，该环境变量的值为;%JAVA_HOME%/lib/tools.jar;%JAVA_HOME%/lib/dt.jar。如果为了可以编译和运行 Java 程序，还应该在 PATH 环境变量中增加%JAVA_HOME%/bin。其中 JAVA_HOME 代表 JDK（不是 JRE）的安装路径。

（2）安装 Android 2.3。创建 AVD 虚拟设备。安装 Android SDK 的方法请参考本书第 1 章。

（3）安装 Apache 的 Tomcat7.0.6，不要使用安装文件安装，而是采用解压缩的安装方式。安装 Tomcat 请参看疯狂 Java 体系的《轻量级 Java EE 企业应用实战》第 1 章。安装完成后，将 Tomcat 安装路径的 lib 下的 jsp-api.jar 和 servlet-api.jar 两个 JAR 文件添加到 CLASSPATH 环境变量之后。

（4）安装 apache-ant-1.8.1。

将下载的 Ant 压缩文件解压缩到任意路径，然后增加 ANT_HOME 的环境变量，让变量的值为 Ant 的解压缩路径。

并在 PATH 环境变量中增加%ANT_HOME%/bin 环境变量。

（5）安装 Eclipse-jee-helios 版（也就是 Eclipse 3.6 for Java EE Developers）。

并安装 ADT 插件，安装插件后在 Eclipse 中设置 Android SDK 的路径。

关于如何安装上面的工具，请参考本书的第 1 章。

3．注意事项

（1）本书所有 Android 项目都是 Eclipse 工程，读者只要将它们导入 Eclipse 工具中即可。

（2）介绍网络编程章节涉及少数 Web 应用，将该 Web 应用复制到%TOMCAT_HOME%/webapps 路径下，然后进入 build.xml 所在路径，执行如下命令：

```
ant compile -- 编译应用
```

启动 Tomcat 服务器，使用浏览器即可访问该应用。

（3）对于 Eclipse 项目文件 ，导入 Eclipse 开发工具即可。

（4）第 19 章的案例，请参看项目下的 readme.txt。

（5）本书有部分按案例需要连接数据库，读者应修改数据库 URL 及用户名、密码让这些代码与读者运行环境一致。如果项目下有 SQL 脚本，导入 SQL 脚本即可，如果没有 SQL 脚本，系统将在运行时自动建表，读者只需创建对应数据库即可。

（6）本书关于网络编程、传感器编程等部分章节需要连接 PC。笔者 PC 的 IP 地址为 192.168.1.88，读者可以将自己的 IP 地址设为该地址，或将程序中用到该 IP 地址的地方修改为自己的 PC 的 IP 地址。

（7）在使用本光盘的程序时，请将程序复制到硬盘上，并去除文件的只读属性。

4．技术支持

如果您使用本光盘的过程中遇到不懂的技术问题，可以登录如下网站与作者联系：

网站：http://www.crazyit.org

衷心感谢

衷心感谢可爱的儿子。大年三十，新年初一，当爸爸依然端坐着创作本书时，你用天籁般的哭声把爸爸从电脑前拉开，努力告诉爸爸：生活除了编程，还有其他更多的乐趣。无数个眼干、腰痛的时刻，你用天使般的笑容舒缓爸爸的心情，带给爸爸生活的轻松。当爸爸抱着你在电脑之前编写代码时，你多次努力给爸爸输入一些“上帝的提示”。

还要感谢博文视点的张月萍编辑，她是一个非常务实的好朋友，因为她的敦促，才有了本书的诞生。在本书的创作过程中，她亦提供了大量切实、有用的帮助。

本书写给谁看

如果你已经具备一定的 Java 基础和 XML 基础，或已经学完了《疯狂 Java 讲义》一书，那么你阅读此书将会比较适合；如果你有不错的 Java 基础，而且有一定的图形界面编程经验，那么阅读本书将可以很快掌握 Android 应用开发。如果你对 Java 的掌握还不熟练，比如对 JDK 安装、Java 基本语法都不熟练，建议遵从学习规律，循序渐进，暂时不要购买、阅读此书。

2011-4-14

目 录 CONTENTS

CHAPTER

18

第 18 章
疯狂连连看

本章要点

- 开发单机休闲游戏的基本方法
- 单机游戏的界面分析
- 单机游戏的游戏界面与数据建模
- 开发单机游戏的界面组件
- 初始化单机游戏的状态
- 自定义 View 开发游戏主界面
- 实现游戏的 Activity
- 定义事件监听器实现游戏的人机交互
- 分情况分析游戏的逻辑处理
- 针对不同情况提供实现

本章将会介绍一个非常常见的小游戏：连连看。这个游戏是单机的休闲小游戏。连连看的游戏界面上均匀分布 2*N* 个尺寸相同的图片，每张图片在游戏中都会出现偶数次，游戏玩家需要依次找到两张相同图片，而且这两张图片之间只用横线、竖线相连（连线上不能有其他图片），并且连线的条数不超过 3 条，那么游戏会消除这两个图片。

对于 Android 学习者来说，学习开发这个小程序难度适中，而且能很好地培养学习者的学习乐趣。开发者需要从程序员的角度来看待玩家面对的游戏界面，游戏界面上的每个图片在底层只要使用一个数值标识来代表即可，不同的图片使用不同的数值表示，只要代表图片的数值相等，即可判断两张图片相同。

开发连连看游戏除了需要理解游戏界面的数据模型之外，程序开发者还需要判断两个方块是否可以相连，为了判断两个方块是否可以相连，开发者需要对两个方块所处的位置进行分类，然后针对不同的情况采用不同的判断算法进行判断，这需要开发者采用条理化的思维方式进行分析、处理，这也是学习本章需要重点掌握的能力。

18.1 连连看游戏简介

连连看是一款广受欢迎的小游戏，它具有玩法简单、耗时少等特征，尤其适合广大白领女性在办公室里休闲、放松。图 18.1 显示了连连看的游戏界面。

从图 18.1 可以看出，在连连看的游戏界面中，平均分布着 2*N* 张图片，每张图片都会出现偶数次，游戏玩家要做的事情就是依次找出两张相同的图片，如果这两张图片之间只用横线、竖线相连（连线上不能有其他图片），并且连线的条数不超过 3 条，那么游戏会消除这两个图片；当整个游戏中所有图片都被消除时，游戏结束。

图 18.2 所示左上角的两个方块之间的连接线只有 3 条，这两个方块将会被消除。

图 18.1 连连看游戏界面

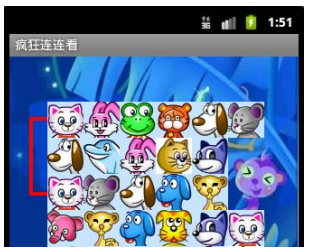


图 18.2 可以消除的方块

连连看可做成单机小游戏，让脱机用户在适当的时候来独自放松；也可做成网络对战游戏——让两个用户进行比赛：谁能最先消除游戏中的所有方块，谁就胜利。

连连看的游戏界面比较简单，而且游戏的实现逻辑也不太复杂，正适合 Android 初学者作为编程进阶的练习项目。

18.2 开发游戏界面

连连看的游戏界面十分简单，大致上可分为两个区域：

- 游戏主界面区。
- 控制按钮与数据显示区。

18.2.1 开发界面布局

本程序将会使用一个 RelativeLayout 作为整体的界面布局元素，界面布局的上面是一个自定义组件，下面是一个水平排列的 LinearLayout。

程序清单：codes\18\Link\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:background="@drawable/room">
<!-- 游戏主界面的自定义组件 -->
<org.crazyit.link.view.GameView
      android:id="@+id/gameView"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" />
<!-- 水平排列的 LinearLayout -->
<LinearLayout
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="horizontal"
      android:layout_marginTop="380px"
      android:background="#1e72bb"
      android:gravity="center">
<!-- 控制游戏开始的按钮 -->
<Button
      android:id="@+id/startButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/button_selector" />
<!-- 显示游戏剩余时间的文本框 -->
<TextView
      android:id="@+id/timeText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:textSize="20dip"
      android:width="150px"
      android:textColor="#ff9" />
</LinearLayout>
</RelativeLayout>
```

这个界面布局很简单，指定按钮的背景色时使用了@drawable/button_selector，这是一个

在 res\drawable 目录下配置的 StateListDrawable 对象，配置文件代码如下。

程序清单：codes\18\Link\res\drawable-mdpi\button_selector.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
     <!-- 指定按钮按下时的图片 -->
     <item android:state_pressed="true"
          android:drawable="@drawable/start_down"
     />
     <!-- 指定按钮松开时的图片 -->
     <item android:state_pressed="false"
          android:drawable="@drawable/start"
     />
</selector>
```

其中 GameView 只是一个 View 的普通子类，开发了上面的界面布局文件之后，运行该程序将可以看到如图 18.3 所示的界面。

18.2.2 开发游戏界面组件

本游戏的界面组件采用了一个自定义 View：GameView，它从 View 基类派生而出，这个自定义 View 的功能就是根据游戏状态来绘制游戏界面上的全部方块。

为了开发这个 GameView，本程序还提供了一个 Piece 类，一个 Piece 对象代表游戏界面上的一个方块，它除了封装方块上的图片之外，还需要封装该方块代表二维数组中的哪个元素；也需要封装它的左上角在游戏界面中 *X*、*Y* 坐标。图 18.4 示意了方块左上角的 *X*、*Y* 坐标的作用。

图 18.3　游戏布局

图 18.4　方块的左上角

方块左上角的 *X*、*Y* 坐标可决定它的绘制位置，GameView 根据这两个坐标值绘制全部方块即可。下面是该程序中 Piece 类的代码。

程序清单：codes\18\Link\src\org\crazyit\link\view\Piece.java

```
public class Piece
{
     // 保存方块对象的所对应的图片
     private PieceImage image;
```

```
        // 该方块的左上角的 x 坐标
        private int beginX;
        // 该方块的左上角的 y 坐标
        private int beginY;
        // 该对象在 Piece[][]数组中第一维的索引值
        private int indexX;
        // 该对象在 Piece[][]数组中第二维的索引值
        private int indexY;
        // 只设置该 Piece 对象在棋盘数组中的位置
        public Piece(int indexX , int indexY)
        {
            this.indexX = indexX;
            this.indexY = indexY;
        }
        public int getBeginX()
        {
            return beginX;
        }
        public void setBeginX(int beginX)
        {
            this.beginX = beginX;
        }
        // 下面省略了各属性的 setter 和 getter 方法
        ...
        // 判断两个 Piece 上的图片是否相同
        public boolean isSameImage(Piece other)
        {
            if (image == null)
            {
                if (other.image != null)
                    return false;
            }
            // 只要 Piece 封装图片 ID 相同，即可认为两个 Piece 相等
            return image.getImageId() == other.image.getImageId();
        }
    }
```

上面的Piece类中封装的PieceImage代表了该方块上的图片，但此处并未直接使用Bitmap对象来代表方块上的图片——因为我们需要使用 PieceImage 来封装两个信息：

- Bitmap 对象。
- 图片资源的 ID。

其中 Bitmap 对象用于在游戏界面上绘制方块；而图片资源的 ID 则代表了该 Piece 对象的标识，当两个 Piece 所封装的图片资源的 ID 相等时，即可认为这两个 Piece 上的图片相同。如以上程序中粗体字代码所示。

下面是 PieceImage 类的代码。

程序清单：codes\18\Link\src\org\crazyit\link\view\PieceImage.java

```
    public class PieceImage
    {
        private Bitmap image;
        private int imageId;
        // 有参数的构造器
        public PieceImage(Bitmap image, int imageId)
```

```
        {
            super();
            this.image = image;
            this.imageId = imageId;
        }
        // 省略了各属性的 setter 和 getter 方法
        ...
    }
```

GameView 主要就是根据游戏的状态数据来绘制界面上的方块，GameView 继承了 View 组件，重写了 View 组件上 onDraw(Canvas canvas)方法，重写该方法主要就是绘制游戏里剩余的方块；除此之外，它还会负责绘制连接方块的连接线。

GamaView 的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\view\GameView.java

```
public class GameView extends View
{
    // 游戏逻辑的实现类
    private GameService gameService;        //①
    // 保存当前已经被选中的方块
    private Piece selectedPiece;
    // 连接信息对象
    private LinkInfo linkInfo;
    private Paint paint;
    // 选中标识的图片对象
    private Bitmap selectImage;
    public GameView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        this.paint = new Paint();
        // 设置连接线的颜色
        this.paint.setColor(Color.RED);
        // 设置连接线的粗细
        this.paint.setStrokeWidth(3);
        this.selectImage = ImageUtil.getSelectImage(context);
    }
    public void setLinkInfo(LinkInfo linkInfo)
    {
        this.linkInfo = linkInfo;
    }
    public void setGameService(GameService gameService)
    {
        this.gameService = gameService;
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        if (this.gameService == null)
            return;
        Piece[][] pieces = gameService.getPieces();          //②
        if (pieces != null)
        {
            // 遍历 pieces 二维数组
            for (int i = 0; i < pieces.length; i++)
```

```
                {
                    for (int j = 0; j < pieces[i].length; j++)
                    {
                        // 如果二维数组中该元素不为空（即有方块），将这个方块的图片画出来
                        if (pieces[i][j] != null)
                        {
                            // 得到这个 Piece 对象
                            Piece piece = pieces[i][j];
                            // 根据方块左上角 X、Y 坐标绘制方块
                            canvas.drawBitmap(piece.getImage().getImage(),
                                piece.getBeginX(), piece.getBeginY(), null);
                        }
                    }
                }
            }
            // 如果当前对象中有 linkInfo 对象，即连接信息
            if (this.linkInfo != null)
            {
                // 绘制连接线
                drawLine(this.linkInfo, canvas);
                // 处理完后清空 linkInfo 对象
                this.linkInfo = null;
            }
            // 画选中标识的图片
            if (this.selectedPiece != null)
            {
                canvas.drawBitmap(this.selectImage, this.selectedPiece.
                getBeginX(),
                    this.selectedPiece.getBeginY(), null);
            }
        }
        // 根据 LinkInfo 绘制连接线的方法
        private void drawLine(LinkInfo linkInfo, Canvas canvas)
        {
            // 获取 LinkInfo 中封装的所有连接点
            List<Point> points = linkInfo.getLinkPoints();
            // 依次遍历 linkInfo 中的每个连接点
            for (int i = 0; i < points.size() - 1; i++)
            {
                // 获取当前连接点与下一个连接点
                Point currentPoint = points.get(i);
                Point nextPoint = points.get(i + 1);
                // 绘制连线
                canvas.drawLine(currentPoint.x, currentPoint.y,
                    nextPoint.x, nextPoint.y, this.paint);
            }
        }
        // 设置当前选中方块的方法
        public void setSelectedPiece(Piece piece)
        {
            this.selectedPiece = piece;
        }
        // 开始游戏方法
        public void startGame()
        {
            this.gameService.start();
```

```
            this.postInvalidate();
        }
    }
```

上面的 GameView 中第一段粗体字代码用于根据游戏的状态数据来绘制界面中的所有方块，第二段粗体字代码则用于根据 LinkInfo 来绘制两个方块之间的连接线。

上面的程序中①号代码处定义了 GameService 对象，②号代码则调用了 GameService 的 getPieces()方法来获取游戏中剩余的方块，GameService 是游戏的业务逻辑实现类。后面会详细介绍该类的实现，此处暂不讲解。

➤➤18.2.3 处理方块之间的连接线

LinkInfo 是一个非常简单的工具类，它用于封装两个方块之间的连接信息——其实就是封装一个 List，List 里保存了连接线需要经过的点。

在实现 LinkInfo 对象之前，先来分析两个方块可以相连的情形。连连看游戏的规则约定：两个方块之间最多只能用 3 条线段相连，也就是说最多只能有 2 个“拐点”，加上两个方块的中心，方块的连接信息最多只需要 4 个连接点。图 18.5 显示了允许出现的连接情况。

图 18.5 方块的连接

考虑到 LinkInfo 最多需要封装 4 个连接点，最少需要封装 2 个连接点，因此程序定义如下 LinkInfo 类。

程序清单：codes\18\Link\src\org\crazyit\link\object\LinkInfo.java

```
public class LinkInfo
{
    // 创建一个集合用于保存连接点
    private List<Point> points = new ArrayList<Point>();
    // 提供第一个构造器，表示两个 Point 可以直接相连，没有转折点
    public LinkInfo(Point p1, Point p2)
    {
        // 加到集合中去
        points.add(p1);
        points.add(p2);
    }
    // 提供第二个构造器，表示三个 Point 可以相连，p2 是 p1 与 p3 之间的转折点
    public LinkInfo(Point p1, Point p2, Point p3)
    {
        points.add(p1);
        points.add(p2);
        points.add(p3);
    }
    // 提供第三个构造器，表示四个 Point 可以相连，p2，p3 是 p1 与 p4 的转折点
    public LinkInfo(Point p1, Point p2, Point p3, Point p4)
    {
        points.add(p1);
        points.add(p2);
        points.add(p3);
        points.add(p4);
```

```
        }
        // 返回连接集合
        public List<Point> getLinkPoints()
        {
            return points;
        }
    }
```

LinkInfo 中所用的 Point 代表一个点，程序直接使用了 android.graphics.Point 类，每个 Point 封装了该点的 *X*、*Y* 坐标。

18.3 连连看的状态数据模型

对于游戏玩家而言，游戏界面上看到“元素”千差万别、变化多端；但对于游戏开发者而言，游戏界面上的元素在底层都是一些数据，不同数据所绘制的图片有差异而已。因此建立游戏的状态数据模型是实现游戏逻辑的重要步骤。

➤➤18.3.1 定义数据模型

连连看的游戏界面是一个 $N \times M$ 的“网格”，每个网格上显示一张图片。但对于游戏开发者来说，这个网格只需要用一个二维数据来定义即可，而每个网格上所显示的图片，对于底层的数据模型来说，不同的图片对应于不同的数值即可。图 18.6 显示了数据模型的示意。

0	1	2					
	1	1					
		3					

图 18.6 连连看的数据模型

对于图 18.6 所示的数据模型，只要让数值为 0 的网络上不绘制图片，其他数值的网格则绘制相应的图片，就可显示出连连看的游戏界面了。

本程序实际上并不是直接使用 int[][]数组来保存游戏的状态数据，而是采用 Piece[][]来保存游戏的状态模型——因为 Piece 对象封装的信息更多，不仅包含了该方块的左上角的 *X*、*Y* 坐标，而且还包含了该 Piece 所显示的图片、图片 ID——这个图片 ID 就可作为该 Piece 的数据。

➤➤18.3.2 初始化游戏状态数据

为了初始化游戏状态，程序需要创建一个 Piece[][]数组，为此程序定义一个 AbstractBoard

抽象类，该抽象类的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\AbstractBoard.java

```
public abstract class AbstractBoard
{
    // 定义一个抽象方法，让子类去实现
    protected abstract List<Piece> createPieces(GameConf config,
        Piece[][] pieces);
    public Piece[][] create(GameConf config)
    {
        // 创建 Piece[][]数组
        Piece[][] pieces = new Piece[config.getXSize()][config.getYSize()];
        // 返回非空的 Piece 集合，该集合由子类去创建
        List<Piece> notNullPieces = createPieces(config, pieces);     //①
        // 根据非空 Piece 对象的集合的大小来取图片
        List<PieceImage> playImages = ImageUtil.getPlayImages(config.get
        Context(),
            notNullPieces.size());
        // 所有图片的宽、高都是相同的
        int imageWidth = playImages.get(0).getImage().getWidth();
        int imageHeight = playImages.get(0).getImage().getHeight();
        // 遍历非空的 Piece 集合
        for (int i = 0; i < notNullPieces.size(); i++)
        {
            // 依次获取每个 Piece 对象
            Piece piece = notNullPieces.get(i);
            piece.setImage(playImages.get(i));
            // 计算每个方块左上角的 X、Y 坐标
            piece.setBeginX(piece.getIndexX() * imageWidth
                + config.getBeginImageX());
            piece.setBeginY(piece.getIndexY() * imageHeight
                + config.getBeginImageY());
            // 将该方块对象放入方块数组的相应位置处
            pieces[piece.getIndexX()][piece.getIndexY()] = piece;
        }
        return pieces;
    }
}
```

上面的程序中粗体字代码块用于初始化 Piece[][]数组，初始化代码负责为各非空的 Piece 元素的 beginX、beginY、image 属性赋值，其中 beginX、beginY 根据该方块在二维数组中的位置动态计算得到。

上面的程序中①号代码调用了 createPieces(config, pieces)抽象方法来创建一个 List<Piece>集合，该抽象方法将会交给其子类去实现，这里是典型的“模板模式”的应用。AbstractBoard 抽象基类完全可以根据 Piece 对象在二维数组中的位置动态地计算它的 beginX、beginY，但 AbstractBoard 不确定 Piece[][]数组的哪些元素是非空的。

由于连连看游戏的初始状态可能有很多种——比如横向分布的方块、竖向分布的方块、矩阵排列的方块、随机分布的方块等，该程序为了考虑以后的扩展性，此处只是采用了模板模式：定义 AbstractBoard 抽象基类来完成通用的代码，而暂时无法确定、需要子类实现的方法定义成 createPieces(GameConf config, Piece[][] pieces)抽象方法。

上面的程序中还用到了一个 ImageUtil 工具类，它的作用是自动搜寻/res/drawable-mdpi

目录下的图片，并根据需要随机地读取该目录下的图片。后面会详细介绍该工具类的用法。

下面为该 AbstractBoard 实现 3 个子类。

1．矩阵排列的方块

矩阵排列的方块会填满二维数组的每个数组元素，只是把四周留空即可，该子类的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\FullBoard.java

```
public class FullBoard extends AbstractBoard
{
    @Override
    protected List<Piece> createPieces(GameConf config,
        Piece[][] pieces)
    {
        // 创建一个 Piece 集合，该集合里面存放初始化游戏时所需的 Piece 对象
        List<Piece> notNullPieces = new ArrayList<Piece>();
        for (int i = 1; i < pieces.length - 1; i++)
        {
            for (int j = 1; j < pieces[i].length - 1; j++)
            {
                // 先构造一个 Piece 对象，只设置它在 Piece[][]数组中的索引值
                // 所需要的 PieceImage 由其父类负责设置
                Piece piece = new Piece(i, j);
                // 添加到 Piece 集合中
                notNullPieces.add(piece);
            }
        }
        return notNullPieces;
    }
}
```

该子类初始化的游戏界面如图 18.7 所示。

图 18.7　矩阵排列的方块

2．竖向排列的方块

竖向排列的方块以垂直的空列分隔开，该子类的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\VerticalBoard. java

```
public class VerticalBoard extends AbstractBoard
{
      protected List<Piece> createPieces(GameConf config,
            Piece[][] pieces)
      {
            // 创建一个 Piece 集合，该集合里面存放初始化游戏时所需的 Piece 对象
            List<Piece> notNullPieces = new ArrayList<Piece>();
            for (int i = 0; i < pieces.length; i++)
            {
                  for (int j = 0; j < pieces[i].length; j++)
                  {
                        // 加入判断，符合一定条件才去构造 Piece 对象，并加到集合中
                        if (i % 2 == 0)
                        {
                              // 如果 x 能被 2 整除，即单数列不会创建方块
                              // 先构造一个 Piece 对象，只设置它在 Piece[][]数组中的索引值
                              // 所需要的 PieceImage 由其父类负责设置
                              Piece piece = new Piece(i, j);
                              // 添加到 Piece 集合中
                              notNullPieces.add(piece);
                        }
                  }
            }
            return notNullPieces;
      }
}
```

上面的程序中粗体字代码控制了只设置 i % 2 == 0 的列，也就是只设置索引为偶数的列，该子类初始化的游戏界面如图 18.8 所示。

图 18.8　竖向排列的方块

3．横向排列的方块

竖向排列的方块以水平的空行分隔开，该子类的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\HorizontalBoard.java

```
public class HorizontalBoard extends AbstractBoard
{
      protected List<Piece> createPieces(GameConf config,
            Piece[][] pieces)
```

```
        {
            // 创建一个 Piece 集合，该集合里面存放初始化游戏时所需的 Piece 对象
            List<Piece> notNullPieces = new ArrayList<Piece>();
            for (int i = 0; i < pieces.length; i++)
            {
                for (int j = 0; j < pieces[i].length; j++)
                {
                    // 加入判断，符合一定条件才去构造 Piece 对象，并加到集合中
                    if (j % 2 == 0)
                    {
                        // 如果 j 能被 2 整除，即单数行不会创建方块
                        // 先构造一个 Piece 对象，只设置它在 Piece[][]数组中的索引值
                        // 所需要的 PieceImage 由其父类负责设置
                        Piece piece = new Piece(i, j);
                        // 添加到 Piece 集合中
                        notNullPieces.add(piece);
                    }
                }
            }
            return notNullPieces;
        }
    }
```

上面的程序中粗体字代码控制了只设置 j % 2 == 0 的行，也就是只设置索引为偶数的行，该子类初始化的游戏界面如图 18.9 所示。

图 18.9　横向分布的方块

18.4　加载界面的图片

正如前面 AbstractBoard 类的代码中看到的，当程序需要创建 *N* 个 Piece 对象时，程序会直接调用 ImageUtil 的 getPlayImages()方法去获取图片，该方法将会随机从 res\drawable-mdpi 目录下取得 *N* 张图片。

为了让 getPlayImages()方法从 res\drawable-mdpi 目录下随机取得 *N* 张图片，程序的实现思路可分为如下几步：

① 通过反射来获取 R.drawable 的所有 Field（Android 的每张图片资源都会自动转换为 R.drawable 的静态 Field），并将这些 Field 值添加到一个 List 集合中。

② 从第一步得到的 List 集合中随机“抽取”*N*/2 个图片 ID。

③ 将第二步得到的 *N*/2 个图片 ID 全部复制一份，这样就得到了 *N* 个图片 ID，而且每个图片 ID 都可以找到与之配对的。

④ 将第三步得到的 *N* 个图片 ID 再次“随机打乱”，并根据图片 ID 加载相应的 Bitmap 对象，最后把图片 ID 及对应的 Bitmap 封装成 PieceImage 后返回。

下面是 ImageUtil 类的代码。

程序清单：codes\18\Link\src\org\crazyit\link\util\ImageUtil.java

```
public class ImageUtil
{
    // 保存所有连连看图片资源值(int 类型)
```

```
private static List<Integer> imageValues = getImageValues();
//获取连连看所有图片的 ID（约定所有图片 ID 以 p_开头）
public static List<Integer> getImageValues()
{
      try
      {
            // 得到 R.drawable 所有的属性，即获取 drawable 目录下的所有图片
            Field[] drawableFields = R.drawable.class.getFields();
            List<Integer> resourceValues = new ArrayList<Integer>();
            for (Field field : drawableFields)
            {
                  // 如果该 Field 的名称以 p_开头
                  if (field.getName().indexOf("p_") != -1)
                  {
                        resourceValues.add(field.getInt(R.drawable.class));
                  }
            }
            return resourceValues;
      }
      catch (Exception e)
      {
            return null;
      }
}
/**
 * 随机从 sourceValues 的集合中获取 size 个图片 ID，返回结果为图片 ID 的集合
 * @param sourceValues 从中获取的集合
 * @param size 需要获取的个数
 * @return size 个图片 ID 的集合
 */
public static List<Integer> getRandomValues(List<Integer> sourceValues,
      int size)
{
      // 创建一个随机数生成器
      Random random = new Random();
      // 创建结果集合
      List<Integer> result = new ArrayList<Integer>();
      for (int i = 0; i < size; i++)
      {
            try
            {
                  // 随机获取一个数字，大于、小于 sourceValues.size()的数值
                  int index = random.nextInt(sourceValues.size());
                  // 从图片 ID 集合中获取该图片对象
                  Integer image = sourceValues.get(index);
                  // 添加到结果集中
                  result.add(image);
            }
            catch (IndexOutOfBoundsException e)
            {
                  return result;
            }
      }
      return result;
}
/**
```

```
        * 从 drawable 目录中获取 size 个图片资源 ID, 其中 size 为游戏数量
        * @param size 需要获取的图片 ID 的数量
        * @return size 个图片 ID 的集合
        */
       public static List<Integer> getPlayValues(int size)
       {
            if (size % 2 != 0)
            {
                 // 如果该数除以 2 有余数，将 size 加 1
                 size += 1;
            }
            // 再从所有的图片值中随机获取 size 的一半数量
            List<Integer> playImageValues = getRandomValues(imageValues, size / 2);
            // 将 playImageValues 集合的元素增加一倍（保证所有图片都有与之配对的图片）
            playImageValues.addAll(playImageValues);
            // 将所有图片 ID 随机“洗牌”
            Collections.shuffle(playImageValues);
            return playImageValues;
       }
       /**
        * 将图片 ID 集合转换 PieceImage 对象集合，PieceImage 封装了图片 ID 与图片本身
        * @param context
        * @param resourceValues
        * @return size 个 PieceImage 对象的集合
        */
       public static List<PieceImage> getPlayImages(Context context, int size)
       {
            // 获取图片 ID 组成的集合
            List<Integer> resourceValues = getPlayValues(size);
            List<PieceImage> result = new ArrayList<PieceImage>();
            // 遍历每个图片 ID
            for (Integer value : resourceValues)
            {
                 // 加载图片
                 Bitmap bm = BitmapFactory.decodeResource(
                      context.getResources(),  value);
                 // 封装图片 ID 与图片本身
                 PieceImage pieceImage = new PieceImage(bm, value);
                 result.add(pieceImage);
            }
            return result;
       }
       // 获取选中标识的图片
       public static Bitmap getSelectImage(Context context)
       {
            Bitmap bm = BitmapFactory.decodeResource(context.getResources(),
                 R.drawable.selected);
            return bm;
       }
  }
```

18.5 实现游戏 Activity

前面已经给出了游戏界面的布局文件，该布局文件需要使用一个 Activity 来负责显示，

除此之外，Activity 还需要为游戏界面的按钮、GameView 组件的事件提供事件监听器。

尤其是对于 GameView 组件，程序需要监听用户的触碰动作，当用户触碰屏幕时，程序需要获取用户触碰的是哪个方块，并判断是否需要“消除”该方块。为了判断能否消除该方块，程序需要进行如下判断：

- 如果程序之前已经选中了某个方块，就判断当前触碰的方块是否能与之前的方块“相连”，如果可以相连，则消除两个方块；如果两个方块不可以相连，则把当前方块设置为选中方块。
- 如果程序之前没有选中方块，直接将当前方块设置为选中方块。

下面是该程序的 Activity 的代码。

程序清单：codes\18\Link\src\org\crazyit\link\Link.java

```
public class Link extends Activity
{
    // 游戏配置对象
    private GameConf config;
    // 游戏业务逻辑接口
    private GameService gameService;
    // 游戏界面
    private GameView gameView;
    // 开始按钮
    private Button startButton;
    // 记录剩余时间的 TextView
    private TextView timeTextView;
    // 失败后弹出的对话框
    private AlertDialog.Builder lostDialog;
    // 游戏胜利后的对话框
    private AlertDialog.Builder successDialog;
    // 定时器
    private Timer timer = new Timer();
    // 记录游戏的剩余时间
    private int gameTime;
    // 记录是否处于游戏状态
    private boolean isPlaying;
    // 振动处理类
    private Vibrator vibrator;
    // 记录已经选中的方块
    private Piece selected = null;
    private Handler handler = new Handler()
    {
        public void handleMessage(Message msg)
        {
            switch (msg.what)
            {
                case 0x123:
                    timeTextView.setText("剩余时间： " + gameTime);
                    gameTime--;
                    // 时间小于 0，游戏失败
                    if (gameTime < 0)
                    {
                        stopTimer();
                        // 更改游戏的状态
                        isPlaying = false;
```

```
                              lostDialog.show();
                              return;
                        }
                        break;
                  }
            }
      };
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 初始化界面
            init();
      }
      // 初始化游戏的方法
      private void init()
      {
            config = new GameConf(8, 9, 2, 10 , 100000, this);
            // 得到游戏区域对象
            gameView = (GameView) findViewById(R.id.gameView);
            // 获取显示剩余时间的文本框
            timeTextView = (TextView) findViewById(R.id.timeText);
            // 获取“开始”按钮
            startButton = (Button) this.findViewById(R.id.startButton);
            // 获取振动器
            vibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);
            gameService = new GameServiceImpl(this.config);
            gameView.setGameService(gameService);
            // 为“开始”按钮的单击事件绑定事件监听器
            startButton.setOnClickListener(new View.OnClickListener()
            {
                  @Override
                  public void onClick(View source)
                  {
                        startGame(GameConf.DEFAULT_TIME);
                  }
            });
            // 为游戏区域的触碰事件绑定监听器
            this.gameView.setOnTouchListener(new View.OnTouchListener()
            {
                  public boolean onTouch(View view, MotionEvent e)
                  {
                        if (e.getAction() == MotionEvent.ACTION_DOWN)
                        {
                              gameViewTouchDown(e);
                        }
                        if (e.getAction() == MotionEvent.ACTION_UP)
                        {
                              gameViewTouchUp(e);
                        }
                        return true;
                  }
            });
            // 初始化游戏失败的对话框
            lostDialog = createDialog("Lost", "游戏失败！ 重新开始", R.drawable.lost)
```

```
                .setPositiveButton("确定", new DialogInterface.OnClickListener()
                {
                      public void onClick(DialogInterface dialog, int which)
                      {
                            startGame(GameConf.DEFAULT_TIME);
                      }
                });
          // 初始化游戏胜利的对话框
          successDialog = createDialog("Success", "游戏胜利！ 重新开始",
                R.drawable.success).setPositiveButton("确定",
                new DialogInterface.OnClickListener()
                {
                      public void onClick(DialogInterface dialog, int which)
                      {
                            startGame(GameConf.DEFAULT_TIME);
                      }
                });
    }
    @Override
    protected void onPause()
    {
          // 暂停游戏
          stopTimer();
          super.onPause();
    }
    @Override
    protected void onResume()
    {
          // 如果处于游戏状态中
          if (isPlaying)
          {
                // 以剩余时间重写开始游戏
                startGame(gameTime);
          }
          super.onResume();
    }
    // 触碰游戏区域的处理方法
    private void gameViewTouchDown(MotionEvent event)      //①
    {
          // 获取 GameServiceImpl 中的 Piece[][]数组
          Piece[][] pieces = gameService.getPieces();
          // 获取用户点击的 x 坐标
          float touchX = event.getX();
          // 获取用户点击的 y 坐标
          float touchY = event.getY();
          // 根据用户触碰的坐标得到对应的 Piece 对象
          Piece currentPiece = gameService.findPiece(touchX, touchY);      //②
          // 如果没有选中任何 Piece 对象(即鼠标点击的地方没有图片)，不再往下执行
          if (currentPiece == null)
                return;
          // 将 gameView 中的选中方块设为当前方块
          this.gameView.setSelectedPiece(currentPiece);
          // 表示之前没有选中任何一个 Piece
          if (this.selected == null)
          {
                // 将当前方块设为已选中的方块，重新将 GamePanel 绘制，并不再往下执行
```

```
                this.selected = currentPiece;
                this.gameView.postInvalidate();
                return;
            }
            // 表示之前已经选择了一个
            if (this.selected != null)
            {
                // 在这里要对 currentPiece 和 prePiece 进行判断并进行连接
                LinkInfo linkInfo = this.gameService.link(this.selected,
                    currentPiece);        //③
                // 两个 Piece 不可连, linkInfo 为 null
                if (linkInfo == null)
                {
                    // 如果连接不成功, 将当前方块设为选中方块
                    this.selected = currentPiece;
                    this.gameView.postInvalidate();
                }
                else
                {
                    // 处理成功连接
                    handleSuccessLink(linkInfo, this.selected
                        , currentPiece, pieces);
                }
            }
        }
        // 触碰游戏区域的处理方法
        private void gameViewTouchUp(MotionEvent e)
        {
            this.gameView.postInvalidate();
        }
        // 以 gameTime 作为剩余时间开始或恢复游戏
        private void startGame(int gameTime)
        {
            // 如果之前的 timer 还未取消, 取消 timer
            if (this.timer != null)
            {
                stopTimer();
            }
            // 重新设置游戏时间
            this.gameTime = gameTime;
            // 如果游戏剩余时间与总游戏时间相等, 则重新开始新游戏
            if(gameTime == GameConf.DEFAULT_TIME)
            {
                // 开始新的游戏
                gameView.startGame();
            }
            isPlaying = true;
            this.timer = new Timer();
            // 启动计时器
            this.timer.schedule(new TimerTask()
            {
                public void run()
                {
                    handler.sendEmptyMessage(0x123);
                }
            }, 0, 1000);
```

```
            // 将选中方块设为 null
            this.selected = null;
        }
        /**
         * 成功连接后处理
         *
         * @param linkInfo 连接信息
         * @param prePiece 前一个选中方块
         * @param currentPiece 当前选择方块
         * @param pieces 系统中还剩的全部方块
         */
        private void handleSuccessLink(LinkInfo linkInfo, Piece prePiece,
                Piece currentPiece, Piece[][] pieces)      //④
        {
            // 它们可以相连，让 GamePanel 处理 LinkInfo
            this.gameView.setLinkInfo(linkInfo);
            // 将 gameView 中的选中方块设为 null
            this.gameView.setSelectedPiece(null);
            this.gameView.postInvalidate();
            // 将两个 Piece 对象从数组中删除
            pieces[prePiece.getIndexX()][prePiece.getIndexY()] = null;
            pieces[currentPiece.getIndexX()][currentPiece.getIndexY()] = null;
            // 将选中的方块设置 null
            this.selected = null;
            // 手机振动(100 毫秒)
            this.vibrator.vibrate(100);
            // 判断是否还有剩下的方块，如果没有，游戏胜利
            if (!this.gameService.hasPieces())
            {
                // 游戏胜利
                this.successDialog.show();
                // 停止定时器
                stopTimer();
                // 更改游戏状态
                isPlaying = false;
            }
        }
        // 创建对话框的工具方法
        private AlertDialog.Builder createDialog(String title, String message,
                int imageResource)
        {
            return new AlertDialog.Builder(this).setTitle(title)
                .setMessage(message).setIcon(imageResource);
        }
        private void stopTimer()
        {
            // 停止定时器
            this.timer.cancel();
            this.timer = null;
        }
    }
```

上面的程序的 init()方法中粗体字代码负责为 gameView 组件的触碰事件绑定事件监听器，当用户触碰该区域时，事件监听器将会被触发；程序中①号粗体字代码定义的 gameViewTouchDown 方法负责处理触碰事件。它会先根据触碰点计算出触碰的方块，如②号

粗体字代码所示；接下来该方法会判断是否之前已有选中的方块，如果没有，直接将当前方块设为选中方块，如果有，判断两个方块是否可以相连，如③号粗体字代码所示。

如果两个方块可以相连，程序将会从 Piece[][]数组中删除这两个方块，该逻辑由上面程序中④号粗体字代码定义的 handleSuccessLink()方法完成。

除此之外，该程序为了控制时间流逝，定义了一个计时器，该计时器每隔 1 秒发送一条消息，程序将会根据该消息减少游戏的剩余时间。上面程序中 startGame(int gameTime)方法内的粗体字代码负责启动计时器。该消息将会交给程序的 Handler 对象处理，如上面程序中开始部分的代码。

该 Link Activity 用的两个类如下。

- GameConf：负责管理游戏的初始化设置信息。
- GameService：负责游戏的逻辑实现。

上面两个工具类中 GameConf 只是一个简单的设置类，此处不再给出介绍，读者自行参考光盘中的代码即可。

18.6 实现游戏逻辑

GameService 组件则是整个游戏逻辑实现的核心，而且 GameService 是一个可以复用的业务逻辑类，它与游戏实现平台无关，既可在 Java Swing 程序中使用，也可在 Android 游戏中使用；甚至只要稍做修改，GameService 也可移植到基于 C#平台的连连看游戏中。

备注：

征得《疯狂 Java 实战演义》作者杨恩雄的同意，本游戏的 GameService 组件基本上使用了《疯狂 Java 实战演义》一书第 7 章案例的 GameService 组件。

18.6.1 定义 GameService 组件接口

根据前面程序对 GameService 组件的依赖，程序需要 GameService 组件包含如下方法。

- start()：初始化游戏状态，开始游戏的方法。
- Piece[][] getPieces()：返回表示游戏状态的 Piece[][]数组。
- boolean hasPieces()：判断 Piece[][]数组中是否还剩 Piece 对象；如果所有 Piece 都被消除了，游戏也就胜利了。
- Piece findPiece(float touchX, float touchY)：根据触碰点的 X、Y 坐标来获取。
- LinkInfo link(Piece p1, Piece p2)：判断 p1、p2 两个方块是否可以相连。

为了考虑以后的可扩展性，先为 GameService 组件定义如下接口。

程序清单：codes\18\Link\src\org\crazyit\link\board\GameService.java

```
public interface GameService
{
      /**
       * 控制游戏开始的方法
       */
      void start();
      /**
```

```
        * 定义一个接口方法，用于返回一个二维数组
        * @return 存放方块对象的二维数组
        */
       Piece[][] getPieces();
       /**
        * 判断参数 Piece[][]数组中是否还存在非空的 Piece 对象
        * @return 如果还剩 Piece 对象则返回 true，没有则返回 false
        */
       boolean hasPieces();
       /**
        * 根据鼠标的 x 坐标和 y 坐标，查找出一个 Piece 对象
        * @param touchX 鼠标点击的 x 坐标
        * @param touchY 鼠标点击的 y 坐标
        * @return 返回对应的 Piece 对象，没有则返回 null
        */
       Piece findPiece(float touchX, float touchY);
       /**
        * 判断两个 Piece 是否可以相连，如果可以连接，则返回 LinkInfo 对象
        * @param p1 第一个 Piece 对象
        * @param p2 第二个 Piece 对象
        * @return 如果可以相连，则返回 LinkInfo 对象，如果两个 Piece 不可以连接，返回 null
        */
       LinkInfo link(Piece p1, Piece p2);
}
```

18.6.2 实现 GameService 组件

GameService 组件的前面三个方法实现起来都比较简单。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
public class GameServiceImpl implements GameService
{
      // 定义一个 Piece[][]数组，只提供 getter 方法
      private Piece[][] pieces;
      // 游戏配置对象
      private GameConf config;
      public GameServiceImpl(GameConf config)
      {
            // 将游戏的配置对象设置到本类中
            this.config = config;
      }
      @Override
      public void start()
      {
            // 定义一个 AbstractBoard 对象
            AbstractBoard board = null;
            Random random = new Random();
            // 获取一个随机数，可取值 0、1、2、3 四值
            int index = random.nextInt(4);
            // 随机生成 AbstractBoard 的子类实例
            switch (index)
            {
                  case 0:
                        // 0 返回 VerticalBoard(竖向)
                        board = new VerticalBoard();
```

```
                    break;
                case 1:
                    // 1 返回 HorizontalBoard(横向)
                    board = new HorizontalBoard();
                    break;
                default:
                    // 默认返回 FullBoard
                    board = new FullBoard();
                    break;
            }
            // 初始化 Piece[][]数组
            this.pieces = board.create(config);
        }
        // 直接返回本对象的 Piece[][]数组
        @Override
        public Piece[][] getPieces()
        {
            return this.pieces;
        }
        // 实现接口的 hasPieces 方法
        @Override
        public boolean hasPieces()
        {
            // 遍历 Piece[][]数组的每个元素
            for (int i = 0; i < pieces.length; i++)
            {
                for (int j = 0; j < pieces[i].length; j++)
                {
                    // 只要任意一个数组元素不为 null，也就是还剩有非空的 Piece 对象
                    if (pieces[i][j] != null)
                    {
                        return true;
                    }
                }
            }
            return false;
        }
        ...
}
```

上面三个方法的实现都很简单，相信读者很容易理解。下面详细介绍剩下的两个方法的实现。

18.6.3 获取触碰点的方块

当用户触碰游戏界面时，事件监听器获取的是该触碰点在游戏界面上的 *X*、*Y* 坐标，但程序需要获取用户触碰的到底是哪个方块，因此程序必须把界面上的 *X*、*Y* 坐标换算成在 Piece[][]二维数组中的两个索引值。

考虑到游戏界面上每个方块的宽度、高度都是相同的，因此想将界面上的 *X*、*Y* 坐标换算成 Piece[][]二维数组中的索引也比较简单，只要拿 *X*、*Y* 坐标值除以图片的宽、高即可。下面的方法是根据触碰点 *X*、*Y* 坐标获取对应方块的代码：

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
// 根据触碰点的位置查找相应的方块
@Override
public Piece findPiece(float touchX, float touchY)
{
      // 由于在创建 Piece 对象的时候，将每个 Piece 的开始坐标加了
      // GameConf 中设置的 beginImageX/beginImageY 值，因此这里要减去这个值
      int relativeX = (int) touchX - this.config.getBeginImageX();
      int relativeY = (int) touchY - this.config.getBeginImageY();
      // 如果鼠标点击的地方比 board 中第一张图片的开始 x 坐标和开始 y 坐标要小，即没有找到相应的方块
      if (relativeX < 0 || relativeY < 0)
      {
            return null;
      }
      // 获取 relativeX 坐标在 Piece[][]数组中的第一维的索引值
      // 第一个参数为每张图片的宽
      int indexX = getIndex(relativeX, GameConf.PIECE_WIDTH);
      // 获取 relativeY 坐标在 Piece[][]数组中的第二维的索引值
      // 第二个参数为每张图片的高
      int indexY = getIndex(relativeY, GameConf.PIECE_HEIGHT);
      // 这两个索引比数组的最小索引还小，返回 null
      if (indexX < 0 || indexY < 0)
      {
            return null;
      }
      // 这两个索引比数组的最大索引还大(或者等于)，返回 null
      if (indexX >= this.config.getXSize()
            || indexY >= this.config.getYSize())
      {
            return null;
      }
      // 返回 Piece[][]数组的指定元素
      return this.pieces[indexX][indexY];
}
```

上面的方法中两行粗体字代码用于根据触碰点 *X*、*Y* 坐标来计算它在 Piece[][]数组中的索引值。该方法调用了 getIndex(int relative , int size)进行计算。

getIndex(int relative , int size)方法的实现就是拿 relative 除以 size，只是程序需要判断可以整除和不能整除两种情况：如果可以整除，说明还在前一个方块内；如果不能整除，则对应于下一个方块。下面是 getIndex(int relative , int size)方法的代码。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
// 工具方法，根据 relative 坐标计算相对于 Piece[][]数组的第一维
// 或第二维的索引值 ，size 为每张图片边的长或者宽
private int getIndex(int relative, int size)
{
      // 表示坐标 relative 不在该数组中
      int index = -1;
      // 让坐标除以边长，没有余数，索引减 1
      // 例如点了 x 坐标为 20，边宽为 10，20 % 10 没有余数
      // index 为 1，即在数组中的索引为 1(第二个元素)
      if (relative % size == 0)
      {
            index = relative / size - 1;
      }
```

```
        else
        {
            // 有余数，例如点了 x 坐标为 21，边宽为 10，21 % 10 有余数，index 为 2
            // 即在数组中的索引为 2(第三个元素)
            index = relative / size;
        }
        return index;
    }
```

18.6.4 判断两个方块是否可以相连

判断两个方块是否可以相连是本程序需要处理的最烦琐的地方：因为两个方块可以相连的情形比较多，大致可分为：

- 两个方块位于同一条水平线，可以直接相连。
- 两个方块位于同一条竖直线，可以直接相连。
- 两个方块以两条线段相连，也就是有 1 个拐角。
- 两个方块以三条线段相连，也就是有 2 个拐角。

下面 link(Piece p1, Piece p2)方法把这四种情况分开进行处理，代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
// 实现接口的 link 方法
@Override
public LinkInfo link(Piece p1, Piece p2)
{
    // 两个 Piece 是同一个，即选中了同一个方块，返回 null
    if (p1.equals(p2))
        return null;
    // 如果 p1 的图片与 p2 的图片不相同，则返回 null
    if (!p1.isSameImage(p2))
        return null;
    // 如果 p2 在 p1 的左边，则需要重新执行本方法，两个参数互换
    if (p2.getIndexX() < p1.getIndexX())
        return link(p2, p1);
    // 获取 p1 的中心点
    Point p1Point = p1.getCenter();
    // 获取 p2 的中心点
    Point p2Point = p2.getCenter();
    // 如果两个 Piece 在同一行
    if (p1.getIndexY() == p2.getIndexY())       //①
    {
        // 它们在同一行并可以相连
        if (!isXBlock(p1Point, p2Point, GameConf.PIECE_WIDTH))
        {
            return new LinkInfo(p1Point, p2Point);
        }
    }
    // 如果两个 Piece 在同一列
    if (p1.getIndexX() == p2.getIndexX())       //②
    {
        if (!isYBlock(p1Point, p2Point, GameConf.PIECE_HEIGHT))
        {
            // 它们之间没有真接障碍，没有转折点
            return new LinkInfo(p1Point, p2Point);
```

```
            }
        }
        // 有一个转折点的情况
        // 获取两个点的直角相连的点，即只有一个转折点
        Point cornerPoint = getCornerPoint(p1Point, p2Point,
            GameConf.PIECE_WIDTH, GameConf.PIECE_HEIGHT);    //③
        if (cornerPoint != null)
        {
            return new LinkInfo(p1Point, cornerPoint, p2Point);
        }
        // 该 map 的 key 存放第一个转折点，value 存放第二个转折点
        // map 的 size()说明有多少种可以连的方式
        Map<Point, Point> turns = getLinkPoints(p1Point, p2Point,    //④
            GameConf.PIECE_WIDTH, GameConf.PIECE_WIDTH);
        if (turns.size() != 0)
        {
            return getShortcut(p1Point, p2Point, turns,
                getDistance(p1Point, p2Point));
        }
        return null;
    }
```

上面的程序中 4 条粗体字代码分别代表了两个方块位于同一个水平线可直接相连，两个方块位于同一个竖直线可直接相连，两个方块需要两条线相连，两个方块需要 3 条线相连 4 种情况。上面的方法分别考虑了这四种情况，但程序还需要为这 4 个方法提供实现。

为了实现上面 4 个方法，可以对两个 Piece 的位置关系进行归纳。

- p1 与 p2 在同一行（indexY 值相同）。
- p1 与 p2 在同一列（indexX 值相同）。
- p2 在 p1 的右上角（p2 的 indexX > p1 的 indexX，p2 的 indexY < p1 的 indexY）。
- p2 在 p1 的右下角（p2 的 indexX > p1 的 indexX，p2 的 indexY > p1 的 indexY）。

至于 p2 在 p1 的左上角，以及 p2 在 p1 的左下角这两种情况，程序可以重新执行 link 方法，将 p1 和 p2 两个参数的位置互换即可。

图 18.10　方块四周的通道

18.6.5　定义获取通道的工具方法

这里所谓的通道，指的是一个方块上、下、左、右四个方向上的空白方块，图 18.10 显示了一个方块四周的通道。

下面是获取某个坐标点四周通道的 4 个方法。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
    /**
     * 给一个 Point 对象,返回它的左边通道
    * @param p
     * @param pieceWidth piece 图片的宽
     * @param min 向左遍历时最小的界限
     * @return 给定 Point 左边的通道
     */
    private List<Point> getLeftChanel(Point p, int min, int pieceWidth)
```

```
{
      List<Point> result = new ArrayList<Point>();
      // 获取向左通道，由一个点向左遍历，步长为 Piece 图片的宽
      for (int i = p.x - pieceWidth; i >= min
            ; i = i - pieceWidth)
      {
            // 遇到障碍，表示通道已经到尽头，直接返回
            if (hasPiece(i, p.y))
            {
                  return result;
            }
            result.add(new Point(i, p.y));
      }
      return result;
}

/**
 * 给一个 Point 对象，返回它的右边通道
* @param p
 * @param pieceWidth
 * @param max 向右时的最右界限
 * @return 给定 Point 右边的通道
 */
private List<Point> getRightChanel(Point p, int max, int pieceWidth)
{
      List<Point> result = new ArrayList<Point>();
      // 获取向右通道，由一个点向右遍历，步长为 Piece 图片的宽
      for (int i = p.x + pieceWidth; i <= max
            ; i = i + pieceWidth)
      {
            // 遇到障碍，表示通道已经到尽头，直接返回
            if (hasPiece(i, p.y))
            {
                  return result;
            }
            result.add(new Point(i, p.y));
      }
      return result;
}

/**
 * 给一个 Point 对象，返回它的上面通道
* @param p
 * @param min 向上遍历时最小的界限
 * @param pieceHeight
 * @return 给定 Point 上面的通道
 */
private List<Point> getUpChanel(Point p, int min, int pieceHeight)
{
      List<Point> result = new ArrayList<Point>();
      // 获取向上通道，由一个点向右遍历，步长为 Piece 图片的高
      for (int i = p.y - pieceHeight; i >= min
            ; i = i - pieceHeight)
      {
            // 遇到障碍，表示通道已经到尽头，直接返回
            if (hasPiece(p.x, i))
```

```
            {
                // 如果遇到障碍，直接返回
                return result;
            }
            result.add(new Point(p.x, i));
        }
        return result;
    }

    /**
     * 给一个 Point 对象，返回它的下面通道
    * @param p
     * @param max 向上遍历时的最大界限
     * @return 给定 Point 下面的通道
     */
    private List<Point> getDownChanel(Point p, int max, int pieceHeight)
    {
        List<Point> result = new ArrayList<Point>();
        // 获取向下通道，由一个点向右遍历，步长为 Piece 图片的高
        for (int i = p.y + pieceHeight; i <= max
            ; i = i + pieceHeight)
        {
            // 遇到障碍，表示通道已经到尽头，直接返回
            if (hasPiece(p.x, i))
            {
                // 如果遇到障碍，直接返回
                return result;
            }
            result.add(new Point(p.x, i));
        }
        return result;
    }
```

➤➤18.6.6 没有转折点的横向连接

如果两个 Piece 的在 Piece[][]数组中的第二维索引值相等，那么这两个 Piece 就位于同一行，如前面的 link(Piece p1, Piece p2)方法中①号代码所示，此时程序需要调用 isXBlock(Point p1, Point p2, int pieceWidth)判断 p1、p2 之间是否有障碍。下面是 isXBlock 方法的代码。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
    /**
     * 判断两个 Y 坐标相同的点对象之间是否有障碍，以 p1 为中心向右遍历
    * @param p1
     * @param p2
     * @param pieceWidth
     * @return 两个 Piece 之间有障碍返回 true，否则返回 false
     */
    private boolean isXBlock(Point p1, Point p2, int pieceWidth)
    {
        if (p2.x < p1.x)
        {
            // 如果 p2 在 p1 左边，调换参数位置调用本方法
            return isXBlock(p2, p1, pieceWidth);
        }
```

```
        for (int i = p1.x + pieceWidth; i < p2.x; i = i + pieceWidth)
        {
            if (hasPiece(i, p1.y))
            {// 有障碍
                return true;
            }
        }
        return false;
    }
```

从上面的判断可以看出，如果两个方块位于同一行，且它们之间没有障碍，那么这两个方块就可以消除，两个方块的连接信息就是它们的中心。

18.6.7 没有转折点的纵向连接

与之相似的是，如果两个 Piece 在 Piece[][]数组中的第一维索引值相等，那么这两个 Piece 就位于同一列，如前面的 link(Piece p1, Piece p2)方法中②号代码所示，此时程序需要调用 isYBlock(Point p1, Point p2, int pieceWidth)判断 p1、p2 之间是否有障碍。下面是 isYBlock 方法的代码。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
    /**
     * 判断两个 X 坐标相同的点对象之间是否有障碍，以 p1 为中心向下遍历
    * @param p1
     * @param p2
     * @param pieceHeight
     * @return 两个 Piece 之间有障碍返回 true，否则返回 false
     */
    private boolean isYBlock(Point p1, Point p2, int pieceHeight)
    {
        if (p2.y < p1.y)
        {
            // 如果 p2 在 p1 的上面，调换参数位置重新调用本方法
            return isYBlock(p2, p1, pieceHeight);
        }
        for (int i = p1.y + pieceHeight; i < p2.y; i = i + pieceHeight)
        {
            if (hasPiece(p1.x, i))
            {
                // 有障碍
                return true;
            }
        }
        return false;
    }
```

18.6.8 一个转折点的连接

对于两个方块连接线上只有一个转折点的情况，程序需要先找到这个转折点。为了找到这个转折点，程序定义一个遍历两个通道并获取它们交点的方法。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 遍历两个通道，获取它们的交点
 * @param p1Chanel 第一个点的通道
 * @param p2Chanel 第二个点的通道
 * @return 两个通道有交点，返回交点，否则返回 null
 */
private Point getWrapPoint(List<Point> p1Chanel, List<Point> p2Chanel)
{
      for (int i = 0; i < p1Chanel.size(); i++)
      {
            Point temp1 = p1Chanel.get(i);
            for (int j = 0; j < p2Chanel.size(); j++)
            {
                  Point temp2 = p2Chanel.get(j);
                  if (temp1.equals(temp2))
                  {
                        // 如果两个 List 中有同一个元素，表明这两个通道有交点
                        return temp1;
                  }
            }
      }
      return null;
}
```

为了找出两个方块连接线上的连接点，程序同样需要分析 p1、p2 两个点的位置分布。根据前面的分析，我们知道 p2 要么位于 p1 的右上角，要么位于 p1 的右下角。

注意：

对于 p2 位于 p1 的左上角、左下角的情况，只要把 p1、p2 交换即可。

对于 p2 位于 p1 的右上角的情形，如图 18.11 所示。

从图 18.11 可以看出：当 p2 位于 p1 的右上角时，应该计算 p1 的左通道与 p2 的向下通道是否有交点，p1 的向上通道与 p2 的向左通道是否有交点。

对于 p2 位于 p1 的右下角的情形，如图 18.12 所示。

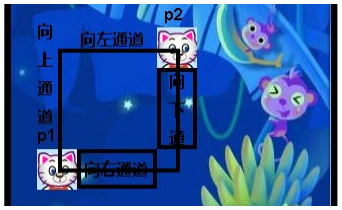


图 18.11　p2 位于 p1 的右上角

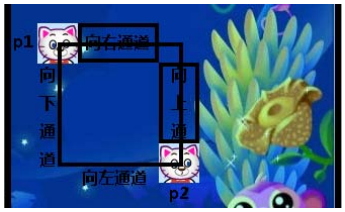


图 18.12　p2 位于 p1 的右下角

从图 18.12 可以看出：当 p2 位于 p1 的右上角时，应该计算 p1 的向右通道与 p2 的向上通道是否有交点，p1 的向下通道与 p2 的向左通道是否有交点。

考虑到 p1 与 p2 具有上面两种分布情形，程序提供了如下方法进行处理。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 获取两个不在同一行或者同一列的坐标点的直角连接点，即只有一个转折点
* @param point1 第一个点
```

```
     * @param point2 第二个点
     * @return 两个不在同一行或者同一列的坐标点的直角连接点
     */
    private Point getCornerPoint(Point point1, Point point2, int pieceWidth,
            int pieceHeight)
    {
            // 先判断这两个点的位置关系
            // point2 在 point1 的左上角, point2 在 point1 的左下角
            if (isLeftUp(point1, point2) || isLeftDown(point1, point2))
            {
                    // 参数换位, 重新调用本方法
                    return getCornerPoint(point2, point1, pieceWidth, pieceHeight);
            }
            // 获取 p1 向右, 向上, 向下的三个通道
            List<Point> point1RightChanel = getRightChanel(point1, point2.x,
                    pieceWidth);
            List<Point> point1UpChanel = getUpChanel(point1, point2.y, pieceHeight);
            List<Point> point1DownChanel = getDownChanel(point1, point2.y,
                    pieceHeight);
            // 获取 p2 向下, 向左, 向下的三个通道
            List<Point> point2DownChanel = getDownChanel(point2, point1.y,
                    pieceHeight);
            List<Point> point2LeftChanel = getLeftChanel(point2, point1.x,
                    pieceWidth);
            List<Point> point2UpChanel = getUpChanel(point2, point1.y, pieceHeight);
            if (isRightUp(point1, point2))
            {
                    // point2 在 point1 的右上角
                    // 获取 p1 向右和 p2 向下的交点
                    Point linkPoint1 = getWrapPoint(point1RightChanel, point2DownChanel);
                    // 获取 p1 向上和 p2 向左的交点
                    Point linkPoint2 = getWrapPoint(point1UpChanel, point2LeftChanel);
                    // 返回其中一个交点, 如果没有交点, 则返回 null
                    return (linkPoint1 == null) ? linkPoint2 : linkPoint1;
            }
            if (isRightDown(point1, point2))
            {
                    // point2 在 point1 的右下角
                    // 获取 p1 向下和 p2 向左的交点
                    Point linkPoint1 = getWrapPoint(point1DownChanel, point2LeftChanel);
                    // 获取 p1 向右和 p2 向下的交点
                    Point linkPoint2 = getWrapPoint(point1RightChanel, point2UpChanel);
                    return (linkPoint1 == null) ? linkPoint2 : linkPoint1;
            }
            return null;
    }
```

上面的两行粗体字代码分别处理了 p2 位于 p1 的右上、右下的两种情形。

上面的程序中用到了 isLeftUp、isLeftDown、isRightUp、isRightDown 四个方法来判断 p2 位于 p1 的左上、左下、右上、右下 4 种情形，这 4 个方法的实现比较简单，只要对它们的 *X*、*Y* 坐标进行简单判断即可。四个方法的代码如下：

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
    /**
     * 判断 point2 是否在 point1 的左上角
```

```
     * @param point1
     * @param point2
     * @return p2 位于 p1 的左上角时返回 true，否则返回 false
     */
    private boolean isLeftUp(Point point1, Point point2)
    {
         return (point2.x < point1.x && point2.y < point1.y);
    }
    /**
     * 判断 point2 是否在 point1 的左下角
    * @param point1
     * @param point2
     * @return p2 位于 p1 的左下角时返回 true，否则返回 false
     */
    private boolean isLeftDown(Point point1, Point point2)
    {
         return (point2.x < point1.x && point2.y > point1.y);
    }
    /**
     * 判断 point2 是否在 point1 的右上角
    * @param point1
     * @param point2
     * @return p2 位于 p1 的右上角时返回 true，否则返回 false
     */
    private boolean isRightUp(Point point1, Point point2)
    {
         return (point2.x > point1.x && point2.y < point1.y);
    }
    /**
     * 判断 point2 是否在 point1 的右下角
    * @param point1
     * @param point2
     * @return p2 位于 p1 的右下角时返回 true，否则返回 false
     */
    private boolean isRightDown(Point point1, Point point2)
    {
         return (point2.x > point1.x && point2.y > point1.y);
    }
```

18.6.9 两个转折点的连接

两个转折点的连接又是最复杂的一种连接情况，因为两个转折点又可分为如下几种情况。

- p1、p2 位于同一行，不能直接相连，就必须有两个转折点，分向上与向下两种连接情况。
- p1、p2 位于同一列，不能直接相连，也必须有两个转折点，分向左与向右两种连接情况。
- p2 在 p1 的右下角，有 6 种转折情况。
- p2 在 p1 的右上角，同样有 6 种转折情况。

注意：

对于 p2 位于 p1 的左上角、左下角的情况，程序同样只要把 p1、p2 的位置互换即可。

对于上面 4 种情况，同样需要分别进行处理。

1．同一行不能直接相连

p1、p2 位于同一行，但它们不能直接相连，因此必须有两个转折点，图 18.13 显示了这种相连的示意。

从图 18.13 可以看到，当 p1 与 p2 位于同一行不能直接相连时，这两个点既可在上面相连，也可在下面相连，这两种情况都代表它们可以相连，我们先把这两种情况都加入结果中，最后再去计算最近的距离。

实现时可以先构建一个 Map，Map 的 key 为第一个转折点，Map 的 value 为第二个转折点（每种连接情况最多只有两个连接点），如 Map 的 size()大于 1，说明这两个 Point 有多种连接途径，那么程序还需要计算路径最小的连接方式。

2．同一列不能直接相连

p1、p2 位于同一列，但它们不能直接相连，因此必须有两个转折点，图 18.14 显示了这种相连的示意。

图 18.13　同一行不能直接相连

图 18.14　同一列不能直接相连

从图 18.14 可以看到，当 p1 与 p2 位于同一列不能直接相连时，这两个点既可在左边相连，也可在右边相连，这两种情况都代表它们可以相连，我们先把这两种情况都加入结果中，最后再去计算最近的距离。

实现时可以先构建一个 Map，Map 的 key 为第一个转折点，Map 的 value 为第二个转折点（每种连接情况最多只有两个连接点），如 Map 的 size()大于 1，说明这两个 Point 有多种连接途径，那么程序还需要计算路径最小的连接方式。

3．p2 位于 p1 右下角的六种转折情况

p2 位于 p1 右下角时一共可能出现 6 种连接情况，图 18.15～图 18.20 分别绘制了这 6 种连接情况。

图 18.15　p2 位于 p1 右下角、两个转折点情况 1

图 18.16　p2 位于 p1 右下角、两个转折点情况 2

图 18.17　p2 位于 p1 右下角、两个转折点情况 3

图 18.18　p2 位于 p1 右下角、两个转折点情况 4

图 18.19　p2 位于 p1 右下角、两个转折点情况 5

图 18.20　p2 位于 p1 右下角、两个转折点情况 6

图 18.15～图 18.20 显示了 p2 位于 p1 右下角可能出现的 6 种连接情形；实际上 p2 还可能位于 p1 的右上角，其可能出现的 6 种连接情形也与此相似，此处不再详述。

接下来定义一个 getLinkPoints 方法对具有两个连接点的情况进行处理。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 获取两个转折点的情况
 *
 * @param point1
 * @param point2
 * @return Map 对象的每个 key-value 对代表一种连接方式，
 *   其中 key、value 分别代表第 1 个、第 2 个连接点
 */
private Map<Point, Point> getLinkPoints(Point point1, Point point2,
     int pieceWidth, int pieceHeight)
```

```
{
      Map<Point, Point> result = new HashMap<Point, Point>();
      // 获取以 point1 为中心的向上, 向右, 向下的通道
      List<Point> p1UpChanel = getUpChanel(point1, point2.y, pieceHeight);
      List<Point> p1RightChanel = getRightChanel(point1, point2.x, pieceWidth);
      List<Point> p1DownChanel = getDownChanel(point1, point2.y, pieceHeight);
      // 获取以 point2 为中心的向下, 向左, 向上的通道
      List<Point> p2DownChanel = getDownChanel(point2, point1.y, pieceHeight);
      List<Point> p2LeftChanel = getLeftChanel(point2, point1.x, pieceWidth);
      List<Point> p2UpChanel = getUpChanel(point2, point1.y, pieceHeight);
      // 获取 Board 的最大高度
      int heightMax = (this.config.getYSize() + 1) * pieceHeight
            + this.config.getBeginImageY();
      // 获取 Board 的最大宽度
      int widthMax = (this.config.getXSize() + 1) * pieceWidth
            + this.config.getBeginImageX();
      // 先确定两个点的关系
      // point2 在 point1 的左上角或者左下角
      if (isLeftUp(point1, point2) || isLeftDown(point1, point2))
      {
            // 参数换位, 调用本方法
            return getLinkPoints(point2, point1, pieceWidth, pieceHeight);
      }
      // p1、p2 位于同一行不能直接相连
      if (point1.y == point2.y)
      {
            // 在同一行
            // 向上遍历
            // 以 p1 的中心点向上遍历获取点集合
            p1UpChanel = getUpChanel(point1, 0, pieceHeight);
            // 以 p2 的中心点向上遍历获取点集合
            p2UpChanel = getUpChanel(point2, 0, pieceHeight);
            Map<Point, Point> upLinkPoints = getXLinkPoints(p1UpChanel,
                  p2UpChanel, pieceHeight);
            // 向下遍历, 不超过 Board(有方块的地方)的边框
            // 以 p1 中心点向下遍历获取点集合
            p1DownChanel = getDownChanel(point1, heightMax, pieceHeight);
            // 以 p2 中心点向下遍历获取点集合
            p2DownChanel = getDownChanel(point2, heightMax, pieceHeight);
            Map<Point, Point> downLinkPoints = getXLinkPoints(p1DownChanel,
                  p2DownChanel, pieceHeight);
            result.putAll(upLinkPoints);
            result.putAll(downLinkPoints);
      }
      // p1、p2 位于同一列不能直接相连
      if (point1.x == point2.x)
      {
            // 在同一列
            // 向左遍历
            // 以 p1 的中心点向左遍历获取点集合
            List<Point> p1LeftChanel = getLeftChanel(point1, 0, pieceWidth);
            // 以 p2 的中心点向左遍历获取点集合
            p2LeftChanel = getLeftChanel(point2, 0, pieceWidth);
            Map<Point, Point> leftLinkPoints = getYLinkPoints(p1LeftChanel,
                  p2LeftChanel, pieceWidth);
            // 向右遍历, 不得超过 Board 的边框（有方块的地方）
```

```
            // 以 p1 的中心点向右遍历获取点集合
            p1RightChanel = getRightChanel(point1, widthMax, pieceWidth);
            // 以 p2 的中心点向右遍历获取点集合
            List<Point> p2RightChanel = getRightChanel(point2, widthMax,
                pieceWidth);
            Map<Point, Point> rightLinkPoints = getYLinkPoints(p1RightChanel,
                p2RightChanel, pieceWidth);
            result.putAll(leftLinkPoints);
            result.putAll(rightLinkPoints);
        }
        // point2 位于 point1 的右上角
        if (isRightUp(point1, point2))
        {
            // 获取 point1 向上遍历, point2 向下遍历时横向可以连接的点
            Map<Point, Point> upDownLinkPoints = getXLinkPoints(p1UpChanel,
                p2DownChanel, pieceWidth);
            // 获取 point1 向右遍历, point2 向左遍历时纵向可以连接的点
            Map<Point, Point> rightLeftLinkPoints = getYLinkPoints(
                p1RightChanel, p2LeftChanel, pieceHeight);
            // 获取以 p1 为中心的向上通道
            p1UpChanel = getUpChanel(point1, 0, pieceHeight);
            // 获取以 p2 为中心的向上通道
            p2UpChanel = getUpChanel(point2, 0, pieceHeight);
            // 获取 point1 向上遍历, point2 向上遍历时横向可以连接的点
            Map<Point, Point> upUpLinkPoints = getXLinkPoints(p1UpChanel,
                p2UpChanel, pieceWidth);
            // 获取以 p1 为中心的向下通道
            p1DownChanel = getDownChanel(point1, heightMax, pieceHeight);
            // 获取以 p2 为中心的向下通道
            p2DownChanel = getDownChanel(point2, heightMax, pieceHeight);
            // 获取 point1 向下遍历, point2 向下遍历时横向可以连接的点
            Map<Point, Point> downDownLinkPoints = getXLinkPoints(p1DownChanel,
                p2DownChanel, pieceWidth);
            // 获取以 p1 为中心的向右通道
            p1RightChanel = getRightChanel(point1, widthMax, pieceWidth);
            // 获取以 p2 为中心的向右通道
            List<Point> p2RightChanel = getRightChanel(point2, widthMax,
                pieceWidth);
            // 获取 point1 向右遍历, point2 向右遍历时纵向可以连接的点
            Map<Point, Point> rightRightLinkPoints = getYLinkPoints(
                p1RightChanel, p2RightChanel, pieceHeight);
            // 获取以 p1 为中心的向左通道
            List<Point> p1LeftChanel = getLeftChanel(point1, 0, pieceWidth);
            // 获取以 p2 为中心的向左通道
            p2LeftChanel = getLeftChanel(point2, 0, pieceWidth);
            // 获取 point1 向左遍历, point2 向右遍历时纵向可以连接的点
            Map<Point, Point> leftLeftLinkPoints = getYLinkPoints(p1LeftChanel,
                p2LeftChanel, pieceHeight);
            result.putAll(upDownLinkPoints);
            result.putAll(rightLeftLinkPoints);
            result.putAll(upUpLinkPoints);
            result.putAll(downDownLinkPoints);
            result.putAll(rightRightLinkPoints);
            result.putAll(leftLeftLinkPoints);
        }
        // point2 位于 point1 的右下角
```

```
        if (isRightDown(point1, point2))
        {
            // 获取 point1 向下遍历，point2 向上遍历时横向可连接的点
            Map<Point, Point> downUpLinkPoints = getXLinkPoints(p1DownChanel,
                p2UpChanel, pieceWidth);
            // 获取 point1 向右遍历，point2 向左遍历时纵向可连接的点
            Map<Point, Point> rightLeftLinkPoints = getYLinkPoints(
                p1RightChanel, p2LeftChanel, pieceHeight);
            // 获取以 p1 为中心的向上通道
            p1UpChanel = getUpChanel(point1, 0, pieceHeight);
            // 获取以 p2 为中心的向上通道
            p2UpChanel = getUpChanel(point2, 0, pieceHeight);
            // 获取 point1 向上遍历，point2 向上遍历时横向可连接的点
            Map<Point, Point> upUpLinkPoints = getXLinkPoints(p1UpChanel,
                p2UpChanel, pieceWidth);
            // 获取以 p1 为中心的向下通道
            p1DownChanel = getDownChanel(point1, heightMax, pieceHeight);
            // 获取以 p2 为中心的向下通道
            p2DownChanel = getDownChanel(point2, heightMax, pieceHeight);
            // 获取 point1 向下遍历，point2 向下遍历时横向可连接的点
            Map<Point, Point> downDownLinkPoints = getXLinkPoints(p1DownChanel,
                p2DownChanel, pieceWidth);
            // 获取以 p1 为中心的向左通道
            List<Point> p1LeftChanel = getLeftChanel(point1, 0, pieceWidth);
            // 获取以 p2 为中心的向左通道
            p2LeftChanel = getLeftChanel(point2, 0, pieceWidth);
            // 获取 point1 向左遍历，point2 向左遍历时纵向可连接的点
            Map<Point, Point> leftLeftLinkPoints = getYLinkPoints(p1LeftChanel,
                p2LeftChanel, pieceHeight);
            // 获取以 p1 为中心的向右通道
            p1RightChanel = getRightChanel(point1, widthMax, pieceWidth);
            // 获取以 p2 为中心的向右通道
            List<Point> p2RightChanel = getRightChanel(point2, widthMax,
                pieceWidth);
            // 获取 point1 向右遍历，point2 向右遍历时纵向可以连接的点
            Map<Point, Point> rightRightLinkPoints = getYLinkPoints(
                p1RightChanel, p2RightChanel, pieceHeight);
            result.putAll(downUpLinkPoints);
            result.putAll(rightLeftLinkPoints);
            result.putAll(upUpLinkPoints);
            result.putAll(downDownLinkPoints);
            result.putAll(leftLeftLinkPoints);
            result.putAll(rightRightLinkPoints);
        }
        return result;
    }
```

上面的程序中粗体字代码分别调用了 getYLinkPoints、getXLinkPoints 方法来收集各种可能出现的连接路径，getYLinkPoints、getXLinkPoints 两种方法的代码如下。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
    /**
     * 遍历两个集合，先判断第一个集合的元素的 X 坐标与另一个集合中的元素 X 坐标相同(纵向)，
     * 如果相同，即在同一列，再判断是否有障碍，没有则加到结果的 Map 中去
    * @param p1Chanel
```

```
     * @param p2Chanel
     * @param pieceHeight
     * @return
     */
    private Map<Point, Point> getYLinkPoints(List<Point> p1Chanel,
          List<Point> p2Chanel, int pieceHeight)
    {
          Map<Point, Point> result = new HashMap<Point, Point>();
          for (int i = 0; i < p1Chanel.size(); i++)
          {
                Point temp1 = p1Chanel.get(i);
                for (int j = 0; j < p2Chanel.size(); j++)
                {
                      Point temp2 = p2Chanel.get(j);
                      // 如果 X 坐标相同(在同一列)
                      if (temp1.x == temp2.x)
                      {
                            // 没有障碍，放到 map 中去
                            if (!isYBlock(temp1, temp2, pieceHeight))
                            {
                                  result.put(temp1, temp2);
                            }
                      }
                }
          }
          return result;
    }
    /**
     * 遍历两个集合，先判断第一个集合的元素的 Y 坐标与另一个集合中的元素 Y 坐标相同(横向)，
     * 如果相同，即在同一行，再判断是否有障碍，没有 则加到结果的 map 中去
    * @param p1Chanel
     * @param p2Chanel
     * @param pieceWidth
     * @return 存放可以横向直线连接的连接点的键值对
     */
    private Map<Point, Point> getXLinkPoints(List<Point> p1Chanel,
          List<Point> p2Chanel, int pieceWidth)
    {
          Map<Point, Point> result = new HashMap<Point, Point>();
          for (int i = 0; i < p1Chanel.size(); i++)
          {
                // 从第一通道中取一个点
                Point temp1 = p1Chanel.get(i);
                // 再遍历第二个通道，看下第二通道中是否有点可以与 temp1 横向相连
                for (int j = 0; j < p2Chanel.size(); j++)
                {
                      Point temp2 = p2Chanel.get(j);
                      // 如果 y 坐标相同(在同一行)，再判断它们之间是否有直接障碍
                      if (temp1.y == temp2.y)
                      {
                            if (!isXBlock(temp1, temp2, pieceWidth))
                            {
                                  // 没有障碍则直接加到结果的 map 中
                                  result.put(temp1, temp2);
                            }
                      }
```

```
            }
        }
        return result;
    }
```

经过上面的处理之后，getLinkPoints(Point point1, Point point2,int pieceWidth, int pieceHeight)方法可以找出 point1、point2 两个点之间的所有可能的连接情况，该方法返回一个 Map 对象，Map 中每个 key-value 对代表一种连接情况，其中 key 代表第一个连接点，value 代表第二个连接点。

当 point1、point2 之间有多种连接情况时，程序还需要找出所有连接情况中的最短路径，link(Piece p1, Piece p2)方法中④号粗体字代码下调用了 getShortcut(p1Point, p2Point, turns,getDistance(p1Point, p2Point));方法进行处理，下面进行详细分析。

➤➤18.6.10 找出最短距离

为了找出所有连接情况中的最短路径，程序实现可分为两步。

① 遍历转折点 Map 中的所有 key-value 对，与原来选择的两个点构成一个 LinkInfo。每个 LinkInfo 代表一条完整的连接路径，并将这些 LinkInfo 收集成一个 List 集合

② 遍历第一步得到的 List<LinkInfo>集合，计算每个 LinkInfo 中连接全部连接点的总距离，选与最短距离相差最小的 LinkInfo 返回即可。

下面的方法实现了上面的思路。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
    /**
     * 获取 p1 和 p2 之间最短的连接信息
    * @param p1
     * @param p2
     * @param turns 放转折点的 map
     * @param shortDistance 两点之间的最短距离
     * @return p1 和 p2 之间最短的连接信息
     */
    private LinkInfo getShortcut(Point p1, Point p2, Map<Point, Point> turns,
            int shortDistance)
    {
            List<LinkInfo> infos = new ArrayList<LinkInfo>();
            // 遍历结果 Map,
            for (Point point1 : turns.keySet())
            {
                    Point point2 = turns.get(point1);
                    // 将转折点与选择点封装成 LinkInfo 对象，放到 List 集合中
                    infos.add(new LinkInfo(p1, point1, point2, p2));
            }
            return getShortcut(infos, shortDistance);
    }
    /**
     * 从 infos 中获取连接线最短的那个 LinkInfo 对象
    * @param infos
     * @return 连接线最短的那个 LinkInfo 对象
     */
    private LinkInfo getShortcut(List<LinkInfo> infos, int shortDistance)
```

```
{
      int temp1 = 0;
      LinkInfo result = null;
      for (int i = 0; i < infos.size(); i++)
      {
            LinkInfo info = infos.get(i);
            // 计算出几个点的总距离
            int distance = countAll(info.getLinkPoints());
            // 将循环第一个的差距用 temp1 保存
            if (i == 0)
            {
                  temp1 = distance - shortDistance;
                  result = info;
            }
            // 如果下一次循环的值比 temp1 的还小，则用当前的值作为 temp1
            if (distance - shortDistance < temp1)
            {
                  temp1 = distance - shortDistance;
                  result = info;
            }
      }
      return result;
}
/**
 * 计算 List<Point>中所有点的距离总和
* @param points 需要计算的连接点
 * @return 所有点的距离的总和
 */
private int countAll(List<Point> points)
{
      int result = 0;
      for (int i = 0; i < points.size() - 1; i++)
      {
            // 获取第 i 个点
            Point point1 = points.get(i);
            // 获取第 i + 1 个点
            Point point2 = points.get(i + 1);
            // 计算第 i 个点与第 i + 1 个点的距离，并添加到总距离中
            result += getDistance(point1, point2);
      }
      return result;
}
/**
 * 获取两个 LinkPoint 之间的最短距离
* @param p1 第一个点
 * @param p2 第二个点
 * @return 两个点的距离距离总和
 */
private int getDistance(Point p1, Point p2)
{
      int xDistance = Math.abs(p1.x - p2.x);
      int yDistance = Math.abs(p1.y - p2.y);
      return xDistance + yDistance;
}
```

至此，连连看游戏中两个方块可能相连的所有情况都处理完成了，应用程序即可调用

GameServiceImpl 所提供的 link(Piece p1, Piece p2)方法来判断两个方块是否可以相连了，这个过程也是编写该游戏最烦琐的地方。

通过对连连看游戏的分析与开发，读者应该发现编写一个游戏并没有想象的那么难，但也不像想象的那么简单。开发者需要冷静、条理化的思维，先分析游戏中所有可能出现的情况，然后在程序中对所有情况进行判断并进行相应的处理。

本程序的 GameService 组件的实现思路基本来自于《疯狂 Java 实战演义》第 7 章的实现思路，笔者无法保证这种实现方式为最优算法。这种算法实现起来有些烦琐，但它的条理十分清晰，非常适合初、中级程序员学习。

18.7 本章小结

本章开发了一个非常常见的单机休闲类游戏：Android 版的连连看，开发这个流行的小游戏难度适中，而且能充分激发学习热情，对于 Android 学习者来说是一个不错的选择。学习本章需要重点掌握单机游戏的界面分析与数据建模的能力：游戏玩家眼中看到的是游戏界面；但开发者眼中看到的应该是数据模型。除此之外，单机游戏通常总会有一个比较美观的界面，因此通常都需要通过自定义 View 来实现游戏主界面。连连看游戏中需要判断两个方块（图片）是否可以相连，这需要开发者对两个方块的位置分门别类地进行处理，并针对不同的情况提供相应的实现，这也是开发单机游戏需要重点掌握的能力。

物联网•云计算•移动开发

《物联网：技术、应用、标准和商业模式》

周洪波 著
定价：35.00元
内容简介：本书全面、客观、公正、系统地描述了物联网理念和产业兴起的历史渊源、相关技术及其共性、应用和业务模式等内容，是作者多年研发实战经验的总结。本书不仅适合作为高校物联网相关专业教学参考书，也适合其他对物联网有兴趣的读者阅读。

《让云触手可及：微软云计算实践指南》

赵立威 方国伟 主编
定价：38.00元
内容简介：本书由微软专家集体奉献。从实践的角度阐述了企业在选择、采用云计算时应考虑的要点和必需的准备；运用案例对微软云计算平台策略、开发实践做了清晰的介绍。对于身处向云计算时代转型的IT业界具有重要的参考价值。

《虚拟化与云计算》

《虚拟化与云计算》小组著
定价：45.00元
内容简介：本书系统阐述了当今信息产业界最受关注的两项新技术——虚拟化与云计算。本书以在数据中心采用服务器虚拟化技术构建云计算平台为主题，全面地勾画出虚拟化与云计算的产生背景、发展现状和关键技术等。

《Android核心技术与实例详解》

吴亚峰 索依娜 等编著 百纳科技 审校
定价：69.00元（含DVD光盘1张）
内容简介：本书以Android应用程序的开发为主题，并结合真实的案例向读者详细介绍了Android的基本组件的使用及应用程序开发的整个流程。本书附赠DVD光盘1张，其中包含了大容量的手把手教学视频、电子教案（PPT）、实例源代码等。

《Android应用开发详解》

郭宏志 编著
定价：59.80元（含CD光盘1张）
内容简介：本书分为三个部分，共18章，由浅入深地详细介绍了Android的每个开发细节。本书基础翔实，实例丰富，案例真实。从基础到案例覆盖了Android应用开发的三大领域：基础应用、网络应用和游戏应用。读者所需要学习的，正是本书描述的。

《Android系统原理及开发要点详解》

韩超，梁泉 著
定价：58.00元
内容简介：本书按照Android系统的框架和各个子系统的主线，重点介绍开发Android应用程序和构建硬件抽象层。其内容涵盖了Android应用程序开发和Android系统移植构建手机系统两大方面。

《It's Android Time——Google Android创赢路线与产品开发实战》

eoeAndroid开发者社区 编著
定价：69.00元
内容简介：本书对Android相关的产品定义和方向进行了详细的调查和分析，以实例的形式循序渐进地引导大家进一步了解Android的知识。本书深入Android底层讲述如何进行底层开发，同时会站在更高的层面和方向上看待和剖析Android及其开发相关的内容。

《人人都玩开心网：Ext JS+Android+SSH整合开发Web与移动SNS》

李宁 编著
定价：65.00元（含光盘1张）
内容简介：本书的主旨为，以开心网为例实现Web版和Android版的SNS应用。本书分为四篇，前三篇主要实现了Web版的开心网系统。其中重点介绍了Ext JS技术，包括Ext JS的核心组件、对话框、表单组件、布局、数据校验、表格、菜单、树组件等技术。

《OPhone应用开发权威指南》

黄晓庆 主编
詹建飞 田 淼 吴 博 吕志虎 编著
定价：59.00元（含光盘1张）
内容简介：本书系统地介绍了OPhone平台的体系结构、应用程序开发流程和调试技巧、OPhone应用程序开发中涉及的主要模块。本书适合有一定Java编程基础，希望从Symbian、Java ME或者Windows Mobile等平台过渡到OPhone及Android平台的软件开发人员阅读。

《iPhone SDK 3开发指南》

【美】Bill Dudney Chris Adamson 著
李亮 杨武 张永强 苟振兴 译
定价：65.00元
内容简介：本书循序渐进地讲述了基于iPhone SDK的应用程序开发的各个方面，涉及从编码到调试到性能优化的各个步骤。特别值得称道的是本书涵盖了最新的iPhone SDK 3.0的有关内容，既适合作为了解iPhone。

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

Broadview®
www.broadview.com.cn

《疯狂 Android 讲义》读者交流区

尊敬的读者：

感谢您选择我们出版的图书，您的支持与信任是我们持续上升的动力。为了使您能通过本书更透彻地了解相关领域，更深入的学习相关技术，我们将特别为您提供一系列后续的服务，包括：

1. 提供本书的修订和升级内容、相关配套资料；
2. 本书作者的见面会信息或网络视频的沟通活动；
3. 相关领域的培训优惠等。

您可以任意选择以下四种方式之一与我们联系，我们都将记录和保存您的信息，并给您提供不定期的信息反馈。

1．在线提交

登录**www.broadview.com.cn/13576**，填写本书的读者调查表。

2．电子邮件

您可以发邮件至**jsj@phei.com.cn**或**editor@broadview.com.cn**。

3．读者电话

您可以直接拨打我们的读者服务电话：**010-88254369**。

4．信件

您可以写信至如下地址：北京万寿路**173**信箱博文视点，邮编：**100036**。

您还可以告诉我们更多有关您个人的情况，及您对本书的意见、评论等，内容可以包括：

（1）您的姓名、职业、您关注的领域、您的电话、E-mail地址或通信地址；

（2）您了解新书信息的途径、影响您购买图书的因素；

（3）您对本书的意见、您读过的同领域的图书、您还希望增加的图书、您希望参加的培训等。

如果您在后期想停止接收后续资讯，只需编写邮件“退订+需退订的邮箱地址”发送至邮箱：market@broadview.com.cn 即可取消服务。

同时，我们非常欢迎您为本书撰写书评，将您的切身感受变成文字与广大书友共享。我们将挑选特别优秀的作品转载在我们的网站（**www.broadview.com.cn**）上，或推荐至**CSDN.NET**等专业网站上发表，被发表的书评的作者将获得价值**50**元的博文视点图书奖励。

更多信息，请关注博文视点官方微博：**http://t.sina.com.cn/broadviewbj**。

我们期待您的消息！

博文视点愿与所有爱书的人一起，共同学习，共同进步！

通信地址：北京万寿路 173 信箱　博文视点（100036）　　电话：010-51260888

E-mail：jsj@phei.com.cn，editor@broadview.com.cn

www.phei.com.cn
www.broadview.com.cn

反侵权盗版声明

Broadview
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

疯狂源自梦想 技术成就辉煌

看得懂 学得会 做得出

1. 知识全面，覆盖面广

本书深入阐述了Android应用开发的Activity、Service、BroadcastReceiver与ContentProvider四大组件，并详细介绍了Android全部图形界面组件的功能和用法，Android各种资源的管理与用法，Android图形、图像处理，事件处理，Android输入/输出处理，视频/音频等多媒体开发，OpenGL-ES开发，网络通信，传感器和GPS开发等内容，全面覆盖Android官方指南，在某些内容上更加具体、深入。

2. 内容实际，实用性强

本书并不局限于枯燥的理论介绍，而是采用了“项目驱动”的方式来讲授知识点，全书包括近百个实例，几乎每个知识点都可找到对应的参考实例。本书最后还提供了“疯狂连连看”、“电子拍卖系统Android客户端”两个应用，具有极高的参考价值。

3. 讲解详细，上手容易

本书保持了“疯狂Java体系”的一贯风格：操作步骤详细、编程思路清晰，语言平实。只要读者有一定的Java编程基础，阅读本书将可以很轻松地上手Android应用开发；学习完本书最后的两个案例后，读者即可完全满足实际企业中Android应用开发的要求。

阅读此书有任何技术问题，都可以登录如下站点获得解决：
疯狂Java联盟：http://www.crazyit.org

上架建议：移动开发>Android

ISBN 978-7-121-13576-7

9 787121 135767 >

定价：89.00元（含光盘1张）

策划编辑：张月萍
责任编辑：高洪霞
封面设计：李 玲