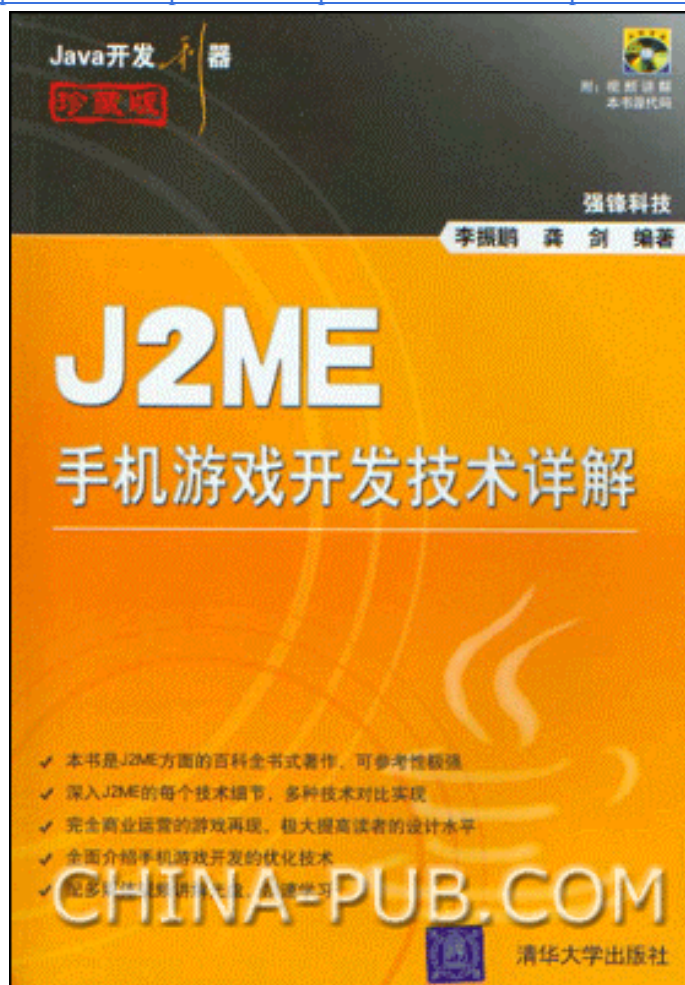


选自：J2ME 手机游戏开发技术详解

作者：李振鹏 龚剑

购书网址：<http://www.china-pub.com/computers/common/info.asp?id=29729>



第一篇 起步篇

第一章 J2ME 概述

1.1 J2ME 体系结构

- 1.1.1 JAVA 技术的版本
- 1.1.2 J2ME 的三层体系结构
- 1.1.3 J2ME 配置(Configuration)
- 1.1.4 K 虚拟机(KVM)
- 1.1.5 J2ME 简表(Profiles)

- 1.1.6 J2ME 规范 (Specification)
- 1.2 有限连接设备配置表 (CLDC)
 - 1.2.1 CLDC 概览
 - 1.2.2 CLDC 中使用的 J2SE 类
 - 1.2.3 CLDC 的字符集和系统属性
 - 1.2.4 CLDC 专用类
 - 1.2.5 CLDC1.1 的新特性
 - 1.2.6 CLDC 安全体系结构
- 1.3 MIDP 简表
 - 1.3.1 设备需求
 - 1.3.2 MIDP 的总体体系结构
 - 1.3.3 MIDP 类库
 - 1.3.4 MIDP2.0 的新特性
 - 1.3.5 MIDP2.0 的安全机制
 - 1.3.6 MIDP 的未来方向
- 1.4 本章小结

第二章 用 J2ME WTK 开发无线应用

- 2.1 J2ME WTK 的安裝配置
 - 2.1.1 WTK 简介
 - 2.1.2 安装前的准备工作
 - 2.1.3 WTK2.2 的安装
 - 2.1.4 WTK2.2 的目录结构
- 2.2 使用 KToolbar 部署应用
 - 2.2.1 启动 KToolbar 工具包
 - 2.2.2 创建新工程
 - 2.2.3 打开一个已有工程
 - 2.2.4 第一个 MIDP 程序“HelloWorld”
 - 2.2.5 编译并运行程序
 - 2.2.6 MIDP 程序打包以及混淆
 - 2.2.7 部署多个 MIDlet 组成的套件
 - 2.2.8 OTA 方式部署应用
- 2.3 MIDlet 应用程序模型
 - 2.3.1 MIDlet 套件
 - 2.3.2 清单文件 (MANIFEST) 和描述文件 (JAD)
 - 2.3.3 应用管理软件
 - 2.3.4 从 JAD 文件中读取属性
 - 2.3.5 MIDlet 的生命周期
 - 2.3.6 编写 MIDlet 应用程序
 - 2.3.7 MIDlet 的开发流程

- 2.4 设置 WTK 的工程属性和开发环境
 - 2.4.1 设置 JAD/清单属性
 - 2.4.2 设置自己的工程目录
 - 2.4.3 使用第三方类库
 - 2.4.4 设置 WTK 的版本控制
 - 2.4.5 设置调试 (Debug) 环境
 - 2.4.6 设置默认模拟器
 - 2.4.7 添加新的模拟器
 - 2.4.8 为程序添加图标
- 2.5 WTK 的模拟器使用
 - 2.5.1 WTK 自带的模拟器
 - 2.5.2 模拟器中的文本输入
 - 2.5.3 用模拟器运行本地配置应用程序
 - 2.5.4 设置模拟器的网络参数
 - 2.5.5 设置模拟器的永久存储和堆栈
 - 2.5.6 设置模拟器的执行性能
 - 2.5.7 监视程序的执行情况
- 2.6 本章小结

第三章 J2ME 应用开发环境的选择和配置

- 3.1 JBuilder2005 开发环境搭建
 - 3.1.1 搭建 J2ME 开发平台
 - 3.1.2 开发 Demo 程序
 - 3.1.3 打包和混淆应用程序
 - 3.1.4 打包和混淆出现的问题
 - 3.1.5 使用 Nokia 开发包开发应用程序
 - 3.1.6 使用 SonyEricsson 开发包开发应用程序
 - 3.1.7 使用 Motorola 开发包开发应用程序
- 3.2 Eclipse 开发环境的搭建
 - 3.2.1 搭建 J2ME 开发平台
 - 3.2.2 开发 Demo 程序
 - 3.2.3 打包和混淆应用程序
- 3.3 本章小结

第二篇 基础篇

第四章 高级用户界面

- 4.1 高级用户界面设计概述
 - 4.1.1 MIDP 用户界面概述
 - 4.1.2 用户界面 API 的分类
 - 4.1.3 高级用户界面的类层次
- 4.2 用户界面的管理
 - 4.2.1 Display 类
 - 4.1.2 Display 对象的管理
 - 4.2.3 可显示 (Displayable) 类
 - 4.2.4 当前显示对象 (Current Displayable)
 - 4.2.5 设置当前显示对象
 - 4.2.7 获取背景光和振动器信息
 - 4.2.8 获取颜色信息
 - 4.2.9 获取样式信息
 - 4.2.10 序列化
- 4.3 高级事件
 - 4.3.1 用户界面事件概述
 - 4.3.2 命令 (Command) 类和命令类型
 - 4.3.3 Command 对象与用户界面的映射
 - 4.3.4 CommandListener 侦听
 - 4.3.5 用 Command 跳转屏幕
- 4.4 高级屏幕类
 - 4.4.1 Screen 类
 - 4.4.2 用信息条 (Ticker 类) 实现滚动文字
 - 4.4.3 用 Alert 类实现提醒功能
 - 4.4.4 AlertType 类
 - 4.4.5 Choise 接口
 - 4.4.6 List
 - 4.4.7 用 TextBox 类输入和编辑文本
- 4.5 屏幕表单 (Form 类) 及其组件
 - 4.5.1 Form 类
 - 4.5.2 Item 类以及对 Item 状态的侦听
 - 4.5.3 用 ChoiceGroup 类实现选项
 - 4.5.4 用 CustomItem 自定义组件

- 4.5.5 用 DateField 类输入日期
- 4.5.6 Gauge (图形标尺) 类
- 4.5.7 用 ImageItem 类在 Form 中显示图片
- 4.5.8 Spacer
- 4.5.9 StringItem
- 4.5.10 用 TextField 类输入文字
- 4.6 本章小结

第五章 低级用户界面

- 5.1 Canvas 类
 - 5.1.1 Canvas 类概述
 - 5.1.2 绘制屏幕
 - 5.1.3 重绘屏幕和强制重绘
 - 5.1.4 显示和隐藏事件
 - 5.1.5 按键事件
 - 5.1.6 游戏动作
 - 5.1.7 指针事件
 - 5.1.8 Command 事件
- 5.2 Graphics 类
 - 5.2.1 Graphics 类概述
 - 5.2.2 颜色模型与 Alpha 透明
 - 5.2.3 绘制直线
 - 5.2.4 绘制和填充矩形
 - 5.2.5 绘制和填充弧形
 - 5.2.6 绘制和填充圆角矩形
 - 5.2.7 填充三角形
 - 5.2.8 画笔风格
 - 5.2.9 设置剪裁区域
 - 5.2.10 坐标系统变换
- 5.3 字体的使用
 - 5.3.1 Font 类概述
 - 5.3.2 字体属性
 - 5.3.3 创建字体
 - 5.3.4 文本的高度和宽度
 - 5.3.5 基线
 - 5.3.6 锚点
 - 5.3.7 绘制字符串
- 5.4 图像的绘制
 - 5.4.1 Image 类概述
 - 5.4.2 不变图像的创建

- 5.4.3 不变图像的绘制
- 5.4.4 ARGB 图像的创建与绘制
- 5.4.5 PNG 图像格式
- 5.4.6 制作 PNG 透明背景
- 5.4.7 可变图像的创建
- 5.4.8 图像双缓冲技术
- 5.4.9 图片处理的注意点
- 5.5 本章小结

第六章 记录存储系统

- 6.1 记录存储系统概述
 - 6.1.1 记录存储的概念
 - 6.1.2 记录存储 API
- 6.2 记录存储的管理
 - 6.2.1 创建记录存储
 - 6.2.2 关闭和删除记录存储
 - 6.2.3 增加记录存储
 - 6.2.4 查询记录存储
 - 6.2.5 修改记录存储
- 6.3 面对记录的高级操作
 - 6.3.1 记录枚举接口
 - 6.3.2 记录过滤接口
 - 6.3.2 记录比较接口
 - 6.3.3 记录监听接口
- 6.4 存储记录的格式问题
 - 6.4.1 二进制格式和其他格式的转换
 - 6.4.2 几个关于数据转换的问题
- 6.5 游戏中的数据存储
 - 6.5.1 游戏中记录存储的方式
 - 6.5.2 游戏记录存储类的使用
 - 6.5.3 与传统的数据存取方式的对比
- 6.6 本章小结

第七章 MIDP 网络编程

- 7.1 通用连接框架
 - 7.1.1 概述
 - 7.1.2 通用连接框架 Generic Connection Framework
 - 7.1.3 GCF 的层次结构
 - 7.1.4 GCF 的使用
- 7.3 MIDP 的 Http 连接

- 7.3.1 HTTP 协议简介
- 7.3.2 HTTP 连接状态
- 7.3.3 建立 HTTP 连接
- 7.3.4 设置 HTTP 请求头
- 7.3.5 使用 HTTP 连接
- 7.3.6 关闭 HTTP 连接
- 7.3.7 使用 http 协议下载文件
- 7.4 Socket 连接简介
 - 7.4.1 套结字 (Socket) 连接简介
 - 7.4.2 使用 Socket 下载文件
- 7.5 Datagram 连接简介
 - 7.5.1 Datagram 连接简介
 - 7.5.2 Datagram 使用实例
- 7.6 Push 技术概述
 - 7.6.1 Push 技术的分类
 - 7.6.2 静态注册与基于 inbound 网络连接的 Push
 - 7.6.3 动态注册与基于计时器的 Push
 - 7.6.4 使用 Push 应注意的问题
- 7.7 本章小结

第八章 声音的播放和处理

- 8.1 MMAPI 概述
 - 8.1.1 MMAPI 的概念
 - 8.1.2 MMAPI 的结构
- 8.2 播放器的创建和管理
 - 8.2.1 MMAPI 所支持的声音格式
 - 8.2.2 Player 接口和状态
- 8.3 播放器的使用
 - 8.3.1 播放简单音调
 - 8.3.2 播放音乐文件
 - 8.3.3 控制播放音量
 - 8.3.4 设置播放次数和循环播放
 - 8.3.6 播放器监听接口
- 8.4 游戏中声音的播放
 - 8.4.1 以独占方式播放声音
 - 8.4.2 以线程方式播放声音
- 8.5 本章小结

第三篇 进阶篇

第九章 MIDP2.0 游戏开发

- 9.1 MIDP2.0 游戏开发概述
 - 9.1.1 MIDP2.0 对游戏的支持
 - 9.1.2 Game API 概览
- 9.2 游戏的屏幕：GameCanvas 类
 - 9.2.1 GameCanvas 概述
 - 9.2.2 绘制双缓冲区
 - 9.2.3 实现游戏主循环
 - 9.2.4 获取键盘状态
 - 9.2.5 Canvas 绘制实例
- 9.3 图层的使用：Layer 类
- 9.4 游戏背景：TiledLayer 类
 - 9.4.1 TiledLayer 类概览
 - 9.4.2 图像贴图(Tile)
 - 9.4.3 单元格 (Cell)
 - 9.4.4 静态背景的制作
 - 9.4.5 动态背景的制作
- 9.5 游戏精灵：Sprite 类
 - 9.5.1 Sprite 类概览
 - 9.5.2 帧 (Frame)
 - 9.5.3 帧序列 (FrameSequence)
 - 9.5.4 旋转 (Transform)
 - 9.5.5 设置参考点 (reference pixel)
 - 9.5.6 精灵的绘制
 - 9.5.7 碰撞检测 (例子)
- 9.6 图层管理：LayerManager 类
 - 9.6.1 LayerManager 类概览
 - 9.6.2 层的索引和操作 (示意图)
 - 9.6.3 LayerManager 的绘制
 - 9.6.4 设置图层的位置
 - 9.6.5 设置显示窗口 (示意图)
 - 9.6.6 双重背景的制作
 - 9.6.7 实例：GameAPI 实现 RPG 游戏
- 9.7 本章小结

第十章 游戏主要编程技术

- 10.1 游戏的线程
 - 10.1.1 线程的创建
 - 10.1.2 线程的启动与停止
 - 10.1.3 线程的休眠与唤醒
 - 10.1.4 线程的优先级
 - 10.1.5 线程的同步问题
- 10.2 游戏的中断和恢复
- 10.3 Vector：用 Vector 实现的子弹
- 10.4 随机数字：游戏的人工智能
- 10.5 定时器 Timer：预定时间内完成任务
 - 10.5.1 只运行一次的定时器
 - 10.5.2 可以重复运用的定时器
- 10.7 游戏闪屏设计
 - 10.7.1 Alert 实现的闪屏
 - 10.7.2 Canvas 实现的闪屏
- 10.8 游戏菜单设计
 - 10.8.1 高级界面实现的菜单
 - 10.8.2 低级界面实现的菜单
- 10.9 本章小结

第十一章 3D 游戏开发入门

- 11.1 J2ME3D 概览
 - 11.1.1 J2ME 3D 游戏概述
 - 11.1.2 M3G 包描述
 - 11.1.3 手机 3D 图像开发技术
 - 11.1.4 M3G 技术所能实现的功能
 - 11.1.5 3D 动画的两种开发模式
 - 11.1.6 场景结构(Scene Graphs)
 - 11.1.7 类描述
- 11.2. 用点、线、面构造复杂形体
 - 11.2.1 基本原则
 - 11.2.2 顶点数组 (VertexArray)
 - 11.2.3 位置数组 (PositionArray)
 - 11.2.4 法向量数组 (NormalArray)
 - 11.2.5 颜色数组 (ColorsArray)
 - 11.2.6 纹理数组 (TextureArray)
 - 11.2.7 顶点缓冲 (VertexBuffer)
 - 11.2.8 索引缓冲 (IndexBuffer)

- 11.2.9 网格多面体 (Mesh)
- 11.2.10 实例：构造一个金字塔
- 11.2.11 顶点变形多面体 (MorphingMesh)
- 11.2.12 实例：变形多面体
- 11.2.13 骨骼变形多面体 (SkinnedMesh)
- 11.2.14 实例：活动的手臂
- 11.3 多面体的外观属性
 - 11.3.1 Apperance 概述
 - 11.3.2 深度和颜色属性 (CompositingMode)
 - 11.3.3 多边形属性 (PolygonMode)
 - 11.3.4 雾化属性 (Fog)
 - 11.3.5 材质属性 (Material)
 - 11.3.6 2 维图像 (Image2D)
 - 11.3.7 纹理映射 (Texture)
 - 11.3.8 实例：添加多重纹理
- 11.4 光源
 - 11.4.1 光照概述
 - 11.4.2 光照的类型
 - 11.4.3 光照的衰减
 - 11.4.4 光源的选择和颜色
 - 11.4.5 创建聚光源
 - 11.4.7 实例：比较光照效果
 - 11.4.8 实例：滚动的光源
- 11.5 构建三维世界
 - 11.5.1 变换矩阵 (Transform)
 - 11.5.2 变换类 (Transformable)
 - 11.5.3 节点 (Node)
 - 11.5.4 抽象三维物体 Object3D
 - 11.5.5 节点组 (Group)
 - 11.5.6 世界 (World)
 - 11.5.7 背景 (Background)
 - 11.5.8 摄影机 (Camera)
 - 11.5.9 精灵 (Sprite3D)
 - 11.5.10 实例：活动的圣诞老人
 - 11.5.11 场景绘制 (Graphics3D)
 - 11.5.12 实例：滚动的骰子
- 11.4. 动画制作
 - 11.4.1 类结构
 - 11.4.2 关键帧序列 (KeyframeSequence)
 - 11.4.3 动画轨迹 (AnimationTrack)

- 11.4.4 动画控制器 (AnimationController)
- 11.4.5 实例：滚动的立方体
- 11.5 使用 M3G 文件创建 3D 场景
 - 11.5.1 3D Max 制作 M3G 文件
 - 11.5.2 制作指南
 - 11.5.3 浏览 M3G 文件
 - 11.5.4 加载 M3G 文件
 - 11.5.5 实例：遍历世界
- 11.6 本章小结

第十二章 J2ME 游戏开发技巧

- 12.1 目标设备的约束
 - 12.1.1 受限的处理器
 - 12.1.2 受限的存储器
 - 12.1.3 联网能力和带宽
 - 12.1.4 输入输出的多样性
 - 12.1.5 外形和重量
 - 12.1.6 移动设备的操作系统
- 12.2 针对 J2ME 游戏开发的技巧及优化技术
 - 12.2.1 优化的基本知识
 - 12.2.2 使用 JBuilder 查找程序的性能瓶颈
 - 12.2.3 优化 J2ME 程序设计和编码，提高程序性能的方法。
 - 12.2.4 优化内存的使用
 - 12.2.5 缩减应用程序体积
 - 12.2.6 网络程序优化
 - 12.2.7 延迟掩盖技术
 - 12.2.8 程序设计的权衡
- 12.3 本章小结

第四篇 实战篇

第十三章 手机游戏开发的总体理念

- 13.1 游戏开发的思想
- 13.2 从游戏玩家角度开发的游戏
- 13.3 从开发者角度开发的游戏
 - 13.3.1 从游戏载体划分的游戏

- 13.3.2 从游戏实现角度划分的游戏
- 13.4 J2ME 手机游戏的版本和移植性
 - 13.4.1 J2ME 的版本介绍
 - 13.4.2 J2ME 的扩展标准
 - 13.4.3 J2ME 的不同厂家实现的差别介绍
 - 13.4.4 J2ME 程序在不同手机之间的移植
- 13.5 本章小结

第十四章 单屏游戏的设计与实现

- 14.1 游戏的策划以及架构
 - 14.1.1 单屏幕手机游戏概述
 - 14.1.2 手机游戏策划概论
 - 14.1.3 逃亡者游戏的策划
 - 14.1.4 逃亡者游戏的准备工作
 - 14.1.5 程序的类结构
 - 14.1.6 游戏的流程图
- 14.2 游戏的实现
 - 14.2.1 主类 escapeeMIDlet 的实现
 - 14.2.2 游戏闪屏 SplashScreen 类的实现
 - 14.2.3 游戏菜单 MenuList 类的实现
 - 14.2.4 高分屏幕 HighScoreScreen 类的实现
 - 14.2.5 简介屏幕 InstructionsScreen 类的实现
 - 14.2.6 子弹 Bullets 类的实现
 - 14.2.7 逃亡飞机 Escapee 类的实现
 - 14.2.8 游戏画布 escapeeCanvas 类的实现
 - 14.2.9 结束屏幕 GameOverScreen 类的实现
 - 14.2.10 声音效果 SoundEffects 类的实现
- 14.3 游戏的优化和改进
 - 14.3.1 逻辑层和表现层的分离
 - 14.3.2 玩家的定制
 - 14.3.3 制作游戏的试玩版
 - 14.3.4 游戏程序的注册
- 14.4 本章小结

第十五章 滚屏游戏设计——经典超级玛丽游戏

- 15.1 滚屏游戏的策划和准备工作
 - 15.1.1 游戏的策划
 - 15.1.2 游戏的准备工作
- 15.2 滚屏游戏的技术
 - 15.2.1 游戏中用到的主要技术

- 15.2.2 滚屏游戏的驱动方式
- 15.2.3 进度条技术
- 15.3 滚屏游戏的架构
 - 15.3.1 程序的总体架构
 - 15.3.2 程序的类结构
- 15.4 滚屏游戏的实现
 - 15.4.1 地图绘制的实现
 - 15.4.2 碰撞检测
 - 15.4.3 键控处理的实现
 - 15.4.4 滚屏的实现
 - 15.4.5 主要人物（玛丽）的实现
 - 15.4.6 乌龟的实现
 - 15.4.7 菜单的实现
 - 15.4.8 游戏的主 Canvas 类的实现
 - 15.4.9 游戏主 Manager 类的实现
- 15.5 滚屏游戏的优化和注意点
 - 15.5.1 游戏中屏幕绘制技术的优化
 - 15.5.2 游戏中新对象的产生和垃圾收集
 - 15.5.3 游戏优化中时间空间的权衡
 - 15.5.4 游戏中关于来电挂起和恢复处理的注意点
- 15.6 本章小结

第十六章 手机网络游戏设计——网络对战扑克游戏

- 16.1 网络游戏客户端的设计实现
 - 16.1.1 程序的设计
 - 16.1.2 程序的架构
 - 16.1.3 程序的实现
 - 16.1.4 程序的优化
- 16.2 网络游戏服务器的设计实现
 - 16.2.1 程序的设计
 - 16.2.2 程序的架构
 - 16.2.3 程序的实现
- 16.3 本章总结

第十七章 3D 迷宫游戏设计与实现

- 17.1 迷宫游戏的策划和准备工作
 - 17.1.1 3D 游戏开发概述
 - 17.1.2 游戏的策划
 - 17.1.3 游戏的准备工作
- 17.2 迷宫游戏的架构

- 17.2.1 游戏线程
- 17.2.3 游戏的类结构
- 17.2.2 游戏的场景架构
- 17.2.4 游戏的流程
- 17.2.5 深度优先算法生成迷宫
- 17.3 迷宫游戏的实现
 - 17.3.1 主类 Maze3DMIDlet 类
 - 17.3.2 菜单列表 MenuList 类的实现
 - 17.3.3 平面 Plane 类的实现
 - 17.3.4 迷宫地图 Maze 类的实现
 - 17.3.5 游戏画布 MazeCanvas 类的实现
 - 17.3.6 设备属性 Graphics3Dproperties 类的实现
 - 17.3.7 错误处理 ErrorScreen 类的实现
- 17.4 3D 手机游戏的相关探讨
 - 17.4.1 迷宫游戏的改善
 - 17.4.2 3D 手机游戏设计概要
 - 17.4.3 3D 绘制性能和游戏引擎
 - 17.4.4 3D 手机游戏的分类
- 17.4 本章小结

第 14 章 单屏游戏的设计与实现

本章实现了一个单屏幕飞机游戏，讲解了 2D 游戏的策划和架构设计，这些设计侧重在单屏手机游戏上。之后对程序设计以及游戏商用化进行了更深层次的讨论，包括游戏定制、版权保护和游戏框架等内容。通过本章的学习希望读者领悟到：

- 2D 游戏的基本设计流程
- 如何让游戏更具有吸引力
- 如何保护游戏版权以及可能有的方法
- 尽可能让玩家可以定制游戏
- 益智类游戏需要关注逻辑层和表现层的分离

14.1 游戏的策划以及架构

本节以一个小游戏为例，讲解了单屏幕游戏的策划、图片制作、类结构和流程设计。这些步骤讲解得比较简单，但却基本能够代表了一个游戏在策划方面的所有工作。在实际设计中，这些步骤还需要更加细致和具体化。例如图片的设计和优化、交互界面的人性化都是游

戏成功的重要因素。

14.1.1 单屏幕手机游戏概述

单屏幕游戏是指那些游戏画面不进行滚动（或者不进行大范围滚动）的游戏，它和滚屏游戏相比照不需要游戏的滚屏技术，但是单屏游戏更加侧重于游戏的可玩性和简单性。

在此将单屏游戏分为两类：其中一类体现在操作简便，持续时间短，通常需要玩家具有较好的操控能力和反应速度。这类游戏的代表例子为泡泡龙、俄罗斯方块、祖玛游戏等，此外类似的还有单屏幕的射击游戏。这类游戏的设计比滚屏游戏要简单，但其难点是可能逻辑判断更为复杂而且对美工的要求更高。如图 14-1 和图 14-2 分别是特别具有代表性的泡泡龙游戏和俄罗斯方块游戏，虽然游戏非常简单，但非常容易让玩家痴迷其中。



图 14-1 一款泡泡龙游戏的游戏画面



图 14-2 两款方块游戏画面

另一类单屏幕游戏是属于益智类的，虽然游戏界面比较简单，也不需要很复杂的游戏特效，但是其游戏逻辑通常使用到人工智能等算法，如何控制游戏难度和优化算法是该类游戏的难点之一。这类游戏包括棋类游戏、牌类游戏和智力测试，例如五子棋、麻将、拼图和扫雷游戏等。如图 14-3 所示为益智类单屏游戏的画面。



图 14-3 益智类游戏

14.1.2 手机游戏策划概论

是什么使得一个手机游戏令人上瘾?为了使一个游戏令人上瘾,它必须有一个使人玩下去的动力。以下是一些因素:想完成这个游戏,想战胜其他的对手,想掌握游戏的操作方法和交互界面,想在游戏的世界中探险并且获得高的得分或等价物。

1. 想完成手机游戏的动力

想完成这个游戏经常基于想看到游戏最终的结果或仅仅只是想完成它。在仅仅只是想完成游戏的例子中,这些游戏只是被看作是一个挑战。去持续地玩一个明显很难的游戏并直到完成为止,这可能是一种满足自信心的行为。然而,这并不能产生一种最好的结果,有些玩家乐于寻找挑战并去战胜它,而其他的玩家可能会觉得它太难了以至于抛在一边。人们都喜欢去赢得胜利,如果你能提供一个游戏,它挑战这些玩家,而且还最终让玩家赢得胜利,那么这个游戏会和高兴的玩家一起愉快地结束。游戏的难度虽然只是一个设计上的选择,但作为一个体贴玩家的设计者必需要充分地考虑到这一因素所带来的结果。

另一方面,一个玩家结束了一个游戏,来看游戏的结局是怎样的,这就是一个故事的推动力,甚至在象超级玛丽奥兄弟《SuperMarioBrothers》这样简单故事的游戏,也有一个结局。许多玩家想知道在不断地战斗并最终救出公主后会发生什么。一旦他们完成了整个故事,又看到了最终的结局,那么多半玩家会终止游戏的进行,不会再去碰它了。

2. 竞争的动力

和其他人竞争是一个有力的因素并且能够保持游戏的活力,能够让人难以致信地在很长一段时间内流行。一个两个人或更多玩家能够很好地互相竞争的游戏能够玩相当长的时间,远远超过了它在手机上的期待生存期。竞争是游戏的基石之一。它允许人们在游戏规则——这一公共标准的监督下互相交互,而且确实把游戏的主动权交到了每个玩家的手中而不是在设计者的手中。如果你设计的游戏规则比较自由、能动,那就允许了玩家创立他们自己的游戏方式,在游戏中移动和使用计谋,就像你以前从未考虑过的。

3. 提高操作技巧的动力

游戏中的技巧或控制也是非常重要的。运动模拟游戏尤为突出地表现了这一点。因为这

类游戏的主要目的是模拟独特的运动控制。玩家经常重复地玩这类游戏来提高自己的操作技巧。

举个例子，在赛车游戏中，在笔直向前的道路上做一个简单的拐弯很容易就能做到，但是为了更好地完成这个动作以赢得时间，你必须能够感觉到道路的情况，以及当你的轮胎打滑时所采取的相应动作。让玩家在你的游戏中学习如何控制的要点是在你给玩家的反馈中。一个轮胎摩擦的声音提示你轮子开始打滑了，当你穿过一个急转弯时发动机的轰鸣声会发生变化等。设计者应提供给玩家游戏操作手册或帮助功能，使玩家能够通过图片、文字、动画来理解游戏的功能和如何操作，这将使玩家逐渐地熟悉这种方式，并且给他们一个理由来试着掌握它。如果玩家不能够获得足够的反馈信息，那就可能没有更好的方式来掌握它。如果一个初玩者和一个老练的玩家在做一些动作时只有相同的选择，那就没有理由尝试着成为一个老练的玩家。

4. 渴望探险的动力

在计算机游戏开始时探险就已经包含在其中了。事实上早期的一些游戏只包括探险。冒险《Adventure》是一个文字类的游戏，在其中玩家可以在广阔的区域中游荡，查找并搜集有趣的物品，使用它们来解开几个谜题，通过这些谜题会发现更广阔的区域需要去探险。当代的探险游戏——冒险岛《Myst》也是用类似内容作为它的基础，这个游戏的流行并不说明许多人在这块有趣地方探险的推动力是去发现神秘岛中奇怪的事件，但有好几个游戏是以这点作为宣传的资本。

隐藏的内容也是许多游戏吸引人的因素之一，超级玛丽奥兄弟的一个吸引点就是去找出隐藏的情节。Shigeru Miyamoto（任天堂的设计小组成员之一，曾制作过超级玛丽奥兄弟，Zelda，Metroid，etc）认为视频游戏 40% 的内容应该是隐藏的，对玛丽兄弟游戏来说这一点做得非常好。

5. 获得高分的动力

渴望获得高分的情况主要分成二类，一般来说在游戏中尝试获得高分或其他等价物的玩家希望在竞争中超过其他玩家的得分记录或想完全地掌握这个游戏。有许多游戏的目地，只是简单地为了赢得一个较高的分数。它起源于古老的撞球游戏，在当前这个更先进的网络时代，这一规律仍然生效，并且广受欢迎。

另一种情况已经超越了赢得游戏本身。在超级玛丽奥兄弟中，当你赢了之后，你可能会在增加了难度后继续去玩。用一个硬币能通关几次或一条命能冲多少关，积多少分，这已经变成了衡量玩家水平高低的标准。老玩家会因水平高而自豪，它甚至会引来其他玩家的尊敬。

设计你的游戏，使你的玩家甚至在游戏获胜后仍愿意继续玩下去。并通过难度的增加，更具有挑战性来激励玩家玩下去。这是你能增加到你游戏中去的要素。

14.1.3 逃亡者游戏的策划

为了简化游戏的策划，选取了一款 PC 小游戏作为范例移植到手机上，它的中文名为《是男人就撑过 30 秒》，相信你一定玩过原版本的这个游戏，虽然看似简单，但绝对有挑战性！这是总结了无数日本纵版飞行射击游戏中的武器而研究出的特殊训练软件，专门用来训练“战

斗机”飞行员。

在漆黑一片的宇宙中，停着一架小飞船，突然四面八方出现很多黄色的小点向小飞船聚集过来，小飞船凭借速度优势和飞行技术从黄点之间的夹缝中飞过，之后又落入新的包围圈中，直至被黄点击毁。坚持的时间越长，就说明水平越高（起码是控制键盘上方向键的水平），持续不同的时间会得到不同的评价。如图 14-4 所示是它原来的游戏画面。



图 14-4 《特训：是男人就撑过 30 秒》游戏画面

这游戏是日本人设计出来的，不得不承认，设计者对现代社会的本质认识得很深刻，将激烈的社会竞争微缩在一个小小的游戏上。从中体会到三点人生启示：第一，玩这个游戏的窍门——不停地绕圈子，人活在这个世界上也是一样，必须始终保持活力，跟上时代的脚步，静止、停步不前就意味着死亡；第二，成功的关键在于坚持、在于坚定的意志，面对挑战和困难，比别人坚持的时间更长，成功的机率就越大，受到来自社会的赞誉就越高；第三，要想成功，仅仅有坚定的意志仍然不够，四面八方都是敌人，如何在运动的物体中找到一条安全的出路，这还需要具备冷静敏锐的头脑。

这个游戏的目的是提供一个指导范例，而不是完全的商业化游戏产品。希望读者能开发出具有更好的图形、音响效果和更具可玩性的游戏。

14.1.4 逃亡者游戏的准备工作

游戏的准备工作主要是准备游戏所需要的图片和声音，并且说明它们在何处用到。这里用到了 7 个图片文件，如图 14-5 所示。它们的大小和用途如表 14-1 所示。

注意：对于游戏中使用的大图，最好用 FireWork 或 PhotShop 等图像软件进行优化，例如在优化之前闪屏图片为 32k，优化之后大小变为 6k，缩小为原来的五分之一。



图 14-5 游戏中使用到的图片

表 14-1 图片清单

图片名	大小 (字节)	象素(宽×高)	用途
back_water.png	557	32 × 32	背景图片
bullet.png	230	6 × 6	子弹图片
Escapee.png	504	69 × 21	飞机图片
explosion.png	1131	128 × 32	爆炸图片
gameover.png	447	84 × 13	游戏结束图片
splash.png	6,158	146 × 156	闪屏图片
logo.png	474	13 × 13	游戏标志

对游戏来说，最有用的功能是播放短的声音文件产生音响效果。MIDP 2.0 Media API 没有规定必须支持的声音文件格式，但对 WAV（8 kHz mono PCM）和 MIDI 文件格式的支持是合理的期望。游戏中用到的声音如表 14-2 所示。

表 14-2 声音清单

声音文件名	大小 (字节)	声音格式	用途
Blast.wav	10544	wav	子弹和飞机碰撞时发出
gameover.mid	103	mid	游戏结束，不是最好成绩时发出
highscore.mid	115	mid	游戏结束，是最好成绩时发出

通过 MIDlet JAR 文件中的资源文件播放声音样本的代码如下所示：

```
try {
    InputStream is = getClass().getResourceAsStream("/dog.wav");    //以输入流的形式获取声音资源
    Player p = Manager.createPlayer(is, "audio/x-wav");             //创建 Player 对象
    p.prefetch();                                                    //声音的预读去
    p.start();                                                        //播放声音
} catch (IOException ex)
{
    //捕获 IO 异常
}
catch (MediaException ex)
{
    //捕获声音文件异常
}
```

可通过调用 `start` 重复使用同一个播放器。但当播放器已经在进行播放时，这样做不起作用，因此可以使用如下代码来确保播放器从头开始播放：

```
p.stop(); //停止声音
p.setMediaTime(0L);
p.start(); //从头从新开始播放
```

游戏中的类 `SoundEffects` 使用了这种方法。此外，MIDP2.0 在类 `Display` 中加入了方法 `vibrate(int)`，这使得游戏可以激活电话的振动功能（如果电话具有该功能）。其参数为用毫秒表示的振动持续时间——没有办法控制振动频率。当产生新的游戏纪录时，游戏使用该方法产生振动。

在类 `Display` 中加入了方法 `flashBacklight(int)` 之后，游戏可以使电话的背景灯闪烁（如果电话支持该功能）。其参数为用毫秒表示的闪烁持续时间——没有办法控制闪烁频率。当第一次显示游戏结束屏幕时，游戏使用该方法（由 `GameOverScreen` 调用）使背景灯闪烁。

14.1.5 程序的类结构

程序中一个有 10 个类，其中 `MIDlet` 主类负责各个屏幕的切换，它们是闪屏屏幕、菜单、介绍屏幕、高分屏幕、游戏屏幕，游戏结束屏幕。游戏中使用到的类为 `SoundEffects`（音效）、`Bullets`（子弹）、`Escapee`（逃亡小飞机）。程序的类结构如图 14-6 所示。

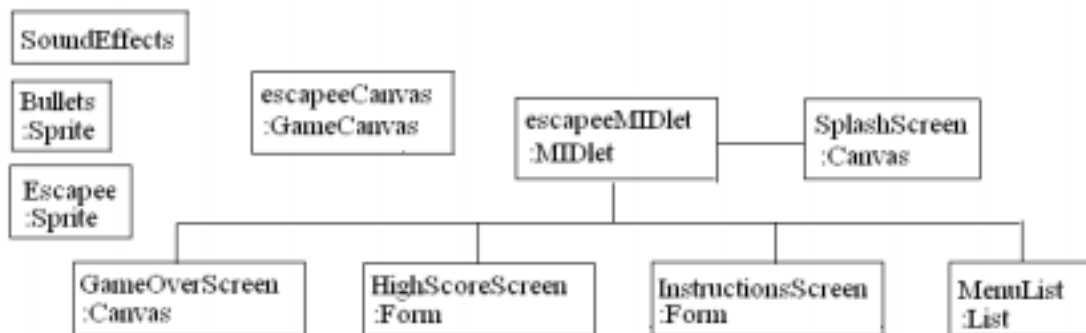


图 14-6 类结构

14.1.6 游戏的流程图

在进入游戏之前先显示闪屏图片，当用户按下键盘或等待 3 秒后，进入游戏菜单。初始情况下，游戏菜单有三个选项，它们分别是开始游戏，游戏说明和高分记录。选择开始新游戏则进入游戏，在游戏中如果按下非游戏键盘则中断游戏返回菜单，此时菜单中增加了一个继续游戏的选项，可以返回游戏也可以重新开始新的游戏。当游戏结束时则进入游戏结束屏幕，屏幕上显示了玩家的成绩和等级，以及游戏的最好成绩，如果当前成绩是最好成绩，则手机震动并播放音乐庆祝成功。在菜单中选择游戏说明或者高分纪录，则进入相应的屏幕，它们都能用“后退”软键返回菜单。菜单中的退出选项用于退出程序。游戏的流程如图 14-7 所示。

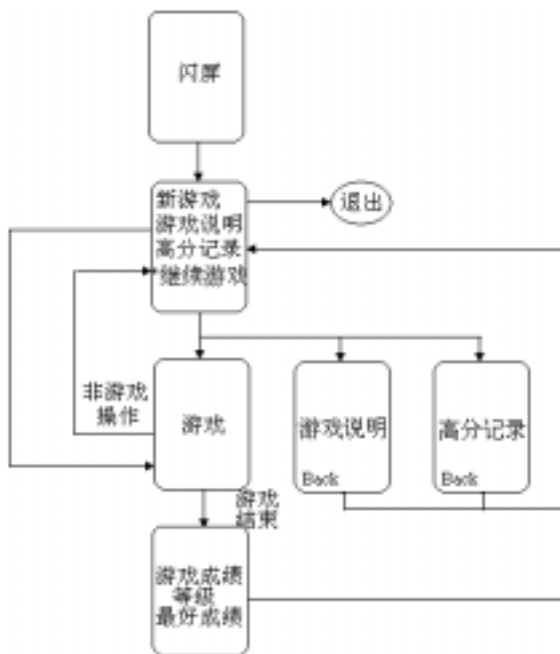


图 14-7 游戏的流程图

14.2 游戏的实现

逃亡者游戏一共实现了几个类包括用于关于游戏外部的闪屏类、菜单类、高分屏幕类、简介屏幕类、结束屏幕类，以及用于游戏本身的游戏画布类、子弹类、逃亡飞机类和声音效果类。

14.2.1 主类 escapeeMIDlet 的实现

MIDlet 被用作一个状态机来管理各种屏幕以及它们之间的转换。例如，当显示 splash 屏幕时，该类和类 SplashScreen（通过方法 splashScreenPainted 和 splashScreenDone）共同在方法 init 中完成背景初始化。而使用方法 readRecordStore 和 writeRecordStore 在一个名为“BESTTIME”的记录存储区中保存最高得分。下面将讲解它的具体实现。

1. 实现闪屏

游戏的最开始将会出现一幅和游戏相关的图片，它会停留一小段时间，然后才进入游戏菜单。游戏闪屏使用得当将会增加整个游戏的视觉效果，而且虽然画面在此期间停留不动，后台程序却是忙碌的，一般在此期间完成基本的初始化工作。

在 startApp 方法中，将游戏的显示权交给闪屏。注意前面说过，startApp 方法可能会在程序受到外界中断的情况下多次调用，所以还应当看是否处于这种情况，如果中断之前的显示内容为游戏画布，则重新启动画布的线程，显示权仍然交给画布。

```

public void startApp()
{
    Displayable current = Display.getDisplay(this).getCurrent();    //获得关于显示的设备上下文
    if (current == null)
    {
        Display.getDisplay(this).setCurrent(new SplashScreen(this)); //第一次将显示闪屏
    }
    else
    {
        if (current == myCanvas)
        {
            myCanvas.start();    //重新开始游戏线程
        }
        Display.getDisplay(this).setCurrent(current);    //显示游戏画布内容
    }
}

```

当闪屏绘制完毕时，将会调用 `splashScreenPainted` 方法，表示程序可以利用剩余的间歇来完成一些工作，这里启动了 MIDlet 实现的一个线程，它会自动去调用 `run` 方法。在 `run` 方法中将会进行了一些初始化工作，这包括读取游戏记录、初始化游戏声音、初始化游戏菜单、初始化游戏画布，当所有工作都完成时，将一个 `initDone` 标志位设置为 `true`，表明已经初始化完毕（用来避免重复初始化）。当闪屏结束后，将会调用 `splashScreenDone` 方法，在这个方法里将检测初始化是否完毕，如果还没有则继续初始化，如果已经完毕则显示游戏菜单。关于闪屏实现的代码如下：

```

void splashScreenPainted()
{
    new Thread(this).start();    //启动幕后的初始化线程
}
public void run()
{
    init();    //调用初始化方法
}
private synchronized void init()
{
    if (!initDone)    //如果还没有初始化完毕
    {
        readRecordStore();    //读取游戏记录
        SoundEffects.getInstance();    //获得声音效果类的唯一实例
        menuList = new MenuList(this);    //初始化菜单
        myCanvas = new escapeeCanvas(this);    //初始化游戏画布
        initDone = true;    //标志位为真表示初始化完毕
    }
}
void splashScreenDone()
{
    init();    //检测是否初始化完毕，如果没有继续初始化
    Display.getDisplay(this).setCurrent(menuList);    //在屏幕上显示游戏菜单
}

```

2. 屏幕切换

前面说过 MIDlet 主类起到一个状态机的作用，负责各个屏幕之间的切换。它所实现的切换功能如下。

从菜单中选择开始游戏，首先初始化 Canvas 的游戏数据，显示游戏画布，然后启动游戏线程：

```
void menuListNewGame()
{
    myCanvas.init();                //初始化游戏数据
    Display.getDisplay(this).setCurrent(myCanvas); //显示游戏画布
    myCanvas.start();                //启动游戏线程
}
```

从菜单中选择显示高分纪录，注意每次都动态生成一个高分纪录屏幕，使用完后就会销毁以避免在不使用的时候占用堆内存：

```
void menuListHighScore()
{
    Display.getDisplay(this).setCurrent(new HighScoreScreen(this)); //显示高分纪录屏幕
}
```

从菜单中选择游戏说明，和选择显示高分纪录类似：

```
void menuListInstructions()
{
    Display.getDisplay(this).setCurrent(new InstructionsScreen(this)); //显示游戏介绍屏幕
}
```

游戏如果中断，从菜单中选择继续游戏，显示游戏画布并重启游戏线程：

```
void menuListContinue()
{
    Display.getDisplay(this).setCurrent(myCanvas); //显示游戏画面
    myCanvas.start(); //启动游戏线程
}
```

从菜单中选择退出，调用 quit 方法：

```
void menuListQuit()
{
    quit(); //退出游戏
}
```

从游戏介绍退回菜单，将显示权交还游戏菜单：

```
void instructionsBack()
{
    Display.getDisplay(this).setCurrent(menuList); //显示游戏菜单
}
```

从高分榜退回菜单，将显示权交还游戏菜单：

```
void highScoreBack()
{
    Display.getDisplay(this).setCurrent(menuList); //显示游戏菜单
}
```

从游戏中切换到菜单，需要中止游戏线程并且告知菜单游戏正在进行中，以便让菜单添加继续游戏的选项：

```
void GameCanvasMenu()
```

```

{
    myCanvas.stop();                //暂停游戏线程
    menuList.setGameActive(true);    //告知菜单游戏已经开始（暂停状态）
    Display.getDisplay(this).setCurrent(menuList);    //显示游戏菜单
}

```

显示游戏结束画面，需要告诉 GameOverScreen 类当前的最好成绩以及子弹的数目，同时告诉菜单游戏已经结束：

```

void GameCanvasGameOver(long time,int BULLETS_NUM)
{
    myCanvas.stop();                //中止游戏线程
    menuList.setGameActive(false);   //告知游戏已经结束
    Display.getDisplay(this).setCurrent(new GameOverScreen(this, time, BULLETS_NUM)); //结束画面
}

```

从游戏结束画面返回，在屏幕上显示菜单：

```

void gameOverDone()
{
    Display.getDisplay(this).setCurrent(menuList);    //显示游戏菜单
}

```

3. 最佳纪录

读取存储游戏纪录，以及检查是否存在最佳纪录的工作也是在这里进行的。读取游戏存储记录的功能在 readRecordStore 方法中实现。在 MIDlet 类的开始指定了 rms 记录的名称，用静态字符串常量表示：

```
private static final String RS_NAME = "BESTTIME";
```

读取方法的代码如下，hasBestTime 标志位用来表明当前是否有最佳成绩。注意如果该记录不存在也不用创建它，而是在写入记录的时候加以创建，此外 rms 的数据记录是从第一条开始的：

```

private void readRecordStore()
{
    hasBestTime = false;            //读取记录还没有获得最好成绩
    RecordStore rs = null;
    ByteArrayInputStream bais = null;
    DataInputStream dis = null;
    try
    {
        rs = RecordStore.openRecordStore(RS_NAME, false);    //记录名为 RS_NAME 的字符串
        byte[] data = rs.getRecord(1);                        //读取第一条记录
        bais = new ByteArrayInputStream(data);
        dis = new DataInputStream(bais);
        bestTime = dis.readLong();                            //读取长整型值
        hasBestTime = true;                                    //以及从记录中获得最佳成绩
    }
    catch (IOException ex)
    {
        // 捕获 IO 异常
    }
    catch (RecordStoreException ex)
    {
    }
}

```



```

} // 捕获读取存储记录异常
finally
{
    if (dis != null)
    {
        try
        {
            dis.close(); //关闭流
        }
        catch (IOException ex)
        {
            // 捕获 IO 异常
        }
    }
    if (bais != null)
    {
        try
        {
            bais.close(); //关闭流
        }
        catch (IOException ex)
        {
            //捕获 IO 异常
        }
    }
    if (rs != null)
    {
        try
        {
            rs.closeRecordStore(); //关闭存储
        }
        catch (RecordStoreException ex)
        {
            //捕获存储异常
        }
    }
}
}

```

除了读取存储之外，如果游戏产生了一个最新成绩，还需要保存到数据存储中去。保存方法如下：

```

private void writeRecordStore()
{
    RecordStore rs = null;
    ByteArrayOutputStream baos = null;
    DataOutputStream dos = null;
    try
    {
        rs = RecordStore.openRecordStore(RS_NAME, true); //带开数据存储如果不存在则创建
        baos = new ByteArrayOutputStream(); //输出字节流
    }
}

```

```

        dos = new DataOutputStream(baos);           //输出数据流
        dos.writeLong(bestTime);                     //写入最佳成绩
        byte[] data = baos.toByteArray();
        if (rs.getNumRecords() == 0)                 //如果里面没有记录
        {
            rs.addRecord(data, 0, data.length);       //添加记录
        }
        else
        {
            rs.setRecord(1, data, 0, data.length);    //如果存在记录，则将第一条设置为当前最佳成绩
        }
    }
    catch (IOException ex)
    {
        // 捕获 IO 异常
    }
    catch (RecordStoreException ex)
    {
        // 捕获数据存储异常
    }
    finally
    {
        if (dos != null)
        {
            try
            {
                dos.close();                         //关闭流
            }
            catch (IOException ex)
            {
                //捕获 IO 异常
            }
        }
        if (baos != null)
        {
            try
            {
                baos.close();                         //关闭流
            }
            catch (IOException ex)
            {
                //捕获 IO 异常
            }
        }
        if (rs != null)
        {
            try
            {
                rs.closeRecordStore();               //关闭数据存储
            }
            catch (RecordStoreException ex)

```

```

        {
            //捕获数据存储异常
        }
    }
}

```

在读取游戏或者从玩家前面的游戏记录中获取到成绩之后，立刻写入 rms 存储。通过比较获得最佳成绩并保存的实现代码如下：

```

boolean checkBestTime(long time)
{
    if (!hasBestTime || (time > bestTime))    //如果还没有最好成绩或者当前成绩比最好成绩长的时候
    {
        hasBestTime = true;                //设置标志位为真，表示有最好成绩
        bestTime = time;                    //上面两种情况下，把最好成绩 bestTime 替换成当前成绩 time
        writeRecordStore();                //保存最好成绩
        return true;                        //返回 true 表明对最好成绩进行了更新操作
    }
    else
    {
        return false;                      //返回 false 表明没有最新的成绩纪录产生
    }
}

```

在生成最好成绩之后（无论是读取存储或者玩家在游戏中动态产生的），就需要让游戏结束屏幕或者高分榜知道这个最好成绩：

```

long getBestTime()
{
    return hasBestTime ? bestTime : -1;
}

```

4. 停止、销毁、退出程序

一般来说，游戏程序的停止、销毁、退出工作主要集中在对游戏画布的处理上，暂停程序如果当前显示内容为游戏画布（表明游戏进行中）则应当调用 Canvas 的 stop 方法暂停游戏线程。注意 Canvas 的暂停需要对当前的游戏状态做一个简单记录，以便将来恢复，Canvas 的暂停方法在后面会讲到。游戏的暂停、退出代码如下：

```

public void pauseApp()
{
    Displayable current = Display.getDisplay(this).getCurrent();    //获得当前设备的显示上下文
    if (current == myCanvas)    //如果屏幕上的显示内容为游戏画布
    {
        myCanvas.stop();    //中止游戏线程
    }
}

public void destroyApp(boolean unconditional)
{
    if (myCanvas != null)
    {
        myCanvas.stop();    //在销毁程序之前先停止游戏线程
    }
}

```

```

    }
    private void quit()
    {
        destroyApp(false);
        notifyDestroyed();           //告知系统已经销毁程序，可以完全退出
    }

```

5. 加载图片资源

MIDlet 中实现了根据字符串类型的文件名来加载图片资源的通用方法，其他各个类都可以使用这个方法来加载图片，也可以不这样做，但是这样使得程序更加模块化和便于理解：

```

static Image createImage(String filename)
{
    Image image = null;
    try
    {
        image = Image.createImage(filename);           //根据文件名创建 Image 对象
    }
    catch (java.io.IOException ex)
    {
        //捕获 IO 异常
    }
    return image;           //返回图片对象 image
}

```

6. 游戏特效

因为只有 MIDlet 类能够访问 Display 对象，因此通常将震动、背景光等特效放置在这个类里面，其代码如下，mills 代表持续的时间。

```

void vibrate(int millis)
{
    Display.getDisplay(this).vibrate(millis);           //手机震动，持续时间为 millis
}
void flashBacklight(int millis)
{
    Display.getDisplay(this).flashBacklight(millis);           //手机背景光闪烁，持续时间为 millis
}

```

14.2.2 游戏闪屏 SplashScreen 类的实现

Splash 屏幕在屏幕中央显示图像，如图 14-8 所示，画面为一架战斗机，以及游戏的名称和版本信息。在图像边缘显示有围绕的红线（这样在所有屏幕大小中都有较好视觉效果）。在第一次绘制屏幕后，它将图像释放作为垃圾回收（把图片设为 null）并回调 MIDlet 进行初始化工作。三秒钟后或第一次按键后，它再次回调 MIDlet 显示游戏菜单。通过这种方法，MIDlet 可在显示 splash 屏幕的同时进行初始化。



图 14-8 游戏闪屏画面

1. 构造函数

SplashScreen 类是一个 Canvas 类的子类，它的构造函数先获得一个对主类的引用，以便于回调主类的方法，然后加载闪屏图片 splash.png，并启动闪屏线程。

```
SplashScreen( escapeeMIDlet midlet)
{
    this.midlet = midlet;
    setFullScreenMode(true);
    splashImage = escapeeMIDlet.createImage("/splash.png");
    new Thread(this).start();
}
```

2. 绘制屏幕

闪屏屏幕的绘制是在 paint 方法中完成，当显示闪屏时会自动调用 paint 方法。注意下面是绘制具有轮廓的文本的有用诀窍：首先用轮廓颜色把它绘制四次，分别向上、下、左、右偏移，然后在它的正常位置用文本颜色对其绘制。在游戏主屏幕中使用该方法要小心，因为文本绘制可能很慢，而该方法使速度减慢为原来的五分之一。

```
public void paint(Graphics g)
{
    int CanvasWidth = getWidth();           //获得画布宽度
    int CanvasHeight = getHeight();          //获得画布高度
    g.setColor(0x00FFFFFF);                 //画笔颜色设置为白色
    g.fillRect(0, 0, CanvasWidth, CanvasHeight); //填充整个屏幕
    g.setColor(0x00FF0000);                 //画笔颜色设置为红色
    g.drawRect(1, 1, CanvasWidth-3, CanvasHeight-3); //在屏幕边缘绘制一个矩形边框
    if (splashImage != null)
    {
        g.drawImage(splashImage, CanvasWidth/2, CanvasHeight/2,
                    Graphics.VCENTER | Graphics.HCENTER); //在屏幕中央绘制闪屏图片
        splashImage = null;                  //将图片对象设置为 null，以便垃圾回收
    }
    g.setFont(Font.getFont(Font.FACE_PROPORTIONAL,Font.STYLE_BOLD, Font.SIZE_LARGE));
    int centerX = CanvasWidth / 2;
    int centerY = CanvasHeight / 2+60;
    g.setColor(0x00FFFFFF);                 //将画笔颜色设置为白色
}
```

```

        drawText(g, centerX, centerY - 1); //分别绘制四次，相差 1 个像素，呈现文本带有背景的感觉
        drawText(g, centerX, centerY + 1);
        drawText(g, centerX - 1, centerY);
        drawText(g, centerX + 1, centerY);
        g.setColor(0x00000000); //将画笔颜色设置为黑色
        drawText(g, centerX, centerY); //绘制文本
        midlet.splashScreenPainted(); //回调 midlet 的方法，通知程序在幕后做初始化
    }

    private void drawText(Graphics g, int centerX, int centerY)
    {
        int fontHeight = g.getFont().getHeight();
        int textHeight = 2 * fontHeight; //文本总的高度
        int topY = centerY - textHeight / 2; //topY 用于定位文本的合适位置
        g.drawString("逃亡者游戏", centerX, topY, Graphics.HCENTER | Graphics.TOP);
        g.drawString("版本: " + midlet.getAppProperty("MIDlet-Version"),
            centerX, topY + fontHeight, Graphics.HCENTER | Graphics.TOP);
    }

```

3. 闪屏线程

闪屏线程的主要工作为接受玩家键盘事件退出或者等待 3 秒退出，代码如下：

```

public void run()
{
    synchronized(this)
    {
        try
        {
            wait(3000L); //等待 3 秒
        }
        catch (InterruptedException e)
        {
            //捕获线程中断异常
        }
        dismiss();
    }
}

```

4. 结束线程

有两种可能会结束闪屏线程，一为在 run 方法中线程等待 3 秒后调用，二为玩家按下键盘，触发键盘事件，调用 keyPressed 方法，在这个方法中结束闪屏线程。

```

public synchronized void keyPressed(int keyCode)
{
    dismiss();
}

```

dismiss 方法如下，它将回调 midlet 的 splashScreenDone 方法，显示游戏菜单：

```

private void dismiss()
{
    if (!dismissed)
    {

```

```

        dismissed = true;
        midlet.splashScreenDone();
    }
}

```

14.2.3 游戏菜单 MenuList 类的实现

菜单列表是在 splash 屏幕之后, 或者游戏结束时, 或者游戏中用户按下一个非游戏键时 (从而暂停了游戏) 显示的屏幕。如果游戏被暂停, 在菜单列表的顶部还有一个额外的菜单项 “继续游戏”。如图 14-9 所示。



图 14-9 游戏菜单

1. 构造函数

MenuList 类是 List 类的子类, 并且实现了 CommandListener 接口, 它的构造函数为, 首先它实现 List 类的构造函数, 然后添加了三个基本选项以及 Command:

```

MenuList(escapeeMIDlet midlet)
{
    super("逃亡者", List.IMPLICIT);           //实现父类 List 的构造方法, 指定 List 名和选择方式
    this.midlet = midlet;
    append("开始新游戏", null); //添加新游戏选项
    append("高分记录", null); //添加高分纪录选项
    append("游戏说明", null); //添加游戏说明选项
    exitCommand = new Command("退出", Command.EXIT, 1); //定义并添加退出软键
    addCommand(exitCommand);
    setCommandListener(this);                //绑定侦听器侦听选项以及软键
}

```

2. 继续游戏选项

当游戏中间被玩家中止时, 游戏菜单中将会多一个 “继续游戏” 的选项, 这个选项添加与否取决于外部传入的布尔值和 gameActive 值, 前者为外部通知菜单游戏是否运行, 后者和继续游戏选项是否存在对应 (初始情况下为 false)。

```

void setGameActive(boolean active)
{
    if (active && !gameActive)                //如果游戏正在运行, 且不存在 “继续游戏” 选项

```

```

    {
        gameActive = true;
        insert(0, "继续游戏", null);           //在选项顶部插入继续游戏选项
    }
    else if (!active && gameActive)           //如果游戏不在运行，但存在“继续游戏”选项
    {
        gameActive = false;
        delete(0);                             //删除继续游戏选项
    }
}

```

3. 处理事件

游戏菜单的事件处理包括两种，一种是 Command 触发的事件，这里为退出程序，另一种是菜单选项的处理，例如开始游戏、显示高分纪录等，处理方式根据选择不同回调 midlet 的相应方法，让 midlet 来负责屏幕的切换。

```

public void commandAction(Command c, Displayable d)
{
    if (c == List.SELECT_COMMAND)
    {
        int index = getSelectedIndex();           //获得选择项的序号
        if (index != -1) // should never be -1
        {
            if (!gameActive)//如果不存在继续游戏选项，则序号往后移位，以便和 case 选项对应
            {
                index++;
            }
            switch (index)
            {
                case 0:                             //继续游戏
                    midlet.menuListContinue();
                    break;
                case 1:                             //新游戏
                    midlet.menuListNewGame();
                    break;
                case 2:                             //高分纪录
                    midlet.menuListHighScore();
                    break;
                case 3:                             //游戏介绍
                    midlet.menuListInstructions();
                    break;
                default:
                    //按照逻辑不可能出现这种情况
                    break;
            }
        }
    }
    else if (c == exitCommand)                   //如果按下退出软键
    {
        midlet.menuListQuit();                   //退出程序
    }
}

```



```

    }
}

```

14.2.4 高分屏幕 HighScoreScreen 类的实现

最高得分屏幕用来显示当前最高得分（到目前为止的最好成绩）。如图 14-10 所示。

HighScoreScreen 类是 Form 的子类，并且实现了 CommandListener 接口，其构造函数在 Form 上添加了当前最好成绩的字符串组件 StringItem 以及添加了返回菜单的软键。

```

HighScoreScreen(escapeeMIDlet midlet)
{
    super("最好成绩");//实现父类的构造函数，指定 Form 的名称
    this.midlet = midlet;
    long bestTime = midlet.getBestTime();           //从 midlet 中获取最好成绩
    float BestTime_Second = (float)bestTime/1000;   //将成绩转化为以秒为单位（原来是毫秒）
    String text = (bestTime == -1) ? "目前还没有，期待您的表现哦！" //判断是否存在这个成绩
                                                         : (Float.toString(BestTime_Second) + "s");
    append(new StringItem("最长坚持时间：", text));
    backCommand = new Command("后退", Command.BACK, 1); //添加后退软键
    addCommand(backCommand);
    setCommandListener(this);                          //绑定侦听器
}

```

当玩家按下返回软键后，将回调 midlet 类的高分 highScoreBack，显示游戏菜单。

```

public void commandAction(Command c, Displayable d)
{
    midlet.highScoreBack();
}

```

14.2.5 简介屏幕 InstructionsScreen 类的实现

说明屏幕向用户显示说明文本，如图 14-11 所示。InstructionsScreen 类是 Form 类的子类，并且实现了 CommandListener 接口。



图 14-10 显示最高纪录

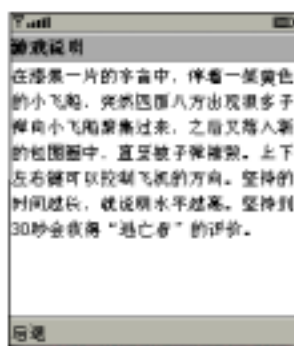


图 14-11 游戏说明屏幕

首先定义关于游戏的文字说明：

```

private static final String instructions =
    "在漆黑一片的宇宙中，停着一架白色的小飞船，" +

```

"突然四面八方出现很多子弹向小飞船聚集过来，" +
 "之后又落入新的包围圈中，直至被子弹摧毁。"+
 "上下左右键可以控制飞机的方向。"+
 "坚持的时间越长，就说明水平越高。"+
 "坚持到 30 秒将获得“逃亡者”的评价。";

然后在构造函数中将游戏说明以 StringItem 组件的形式添加到介绍屏幕上，此外还添加了返回软键和绑定了侦听器：

```
InstructionsScreen(escapeeMIDlet midlet)
{
    super("游戏说明");                //实现父类的构造函数
    this.midlet = midlet;
    append(new StringItem(null, instructions))    ;//用 instructions 创建 StringItem 对象并添加到屏幕上
    backCommand = new Command("后退", Command.BACK, 1);
    addCommand(backCommand);            //添加后退按键
    setCommandListener(this);          //绑定侦听器
}
```

当玩家按下后退软键，就会回调 midlet 的 instructionsBack 方法显示游戏菜单。

14.2.6 子弹 Bullets 类的实现

飞机类游戏中子弹是必不可少的，它们数量很多且充斥着整个屏幕，这些随机或者有着一定 AI 的小物体，实现起来不是总那么容易，有时候不得不考虑很多和效能有关的问题。例如如果游戏中使用了 50 发子弹，可以使用 50 个 Sprite 对象来实现这些子弹，但这样占用大量的内存空间和对象堆栈，甚至影响游戏的正常速度。这种情况下编写了一个 Sprite 类的子类 Bullets 类，这个类的关键部分是一个类似索引的数组，数组中包含并管理各个“虚拟”子弹的位置、速度等信息，而根据这些信息在屏幕的不同位置绘制，这样在玩家看来就有众多的子弹在屏幕上移动。如图 14-12 所示。

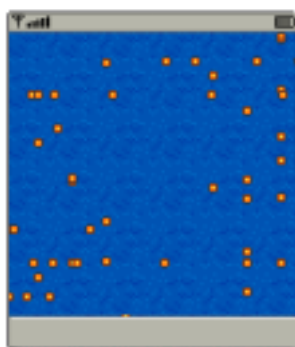


图 14-12 在屏幕上移动的子弹

在编写 Bullets 类之前，将一些可能用到的属性定义为全局变量或常量，这些变量或常量如下，在此将它们都设置为 private 类型，如果外界需要对它们进行设置或查询则必须使用 set 或 get 方法：

```
private int[][] bullets;                //子弹数组
private int BULLETS_NUM;              //子弹数目
```

```
private Random random;           //用来生成随机数
private int frameWidth,frameHeight; //帧的宽度和高度
private static final int ALIVE = 1; //代表子弹处于激活状态
private static final int DEFAULT_SPEED = 3; //默认速度，放置某个方向速度为 0
private int CanvasWidth,CanvasHeight; //屏幕宽度和高度
```

1. 构造函数

由于 Bullets 类继承了 Sprite 类,因此在 Bullets 的构造函数中必须实现 Sprite 类的构造函数,也就是调用 super 语句。构造函数还用来获取必要参数例如帧的宽度和高度,以及定位精灵的参考点等简单的操作。构造函数的代码如下:

```
public Bullets(Image image,int frameWidth,int frameHeight) {
    super(image,frameWidth,frameHeight);
    this.frameWidth = frameWidth;
    this.frameHeight = frameHeight;
    defineReferencePixel(frameWidth/2 , frameHeight/2 );
    random=new Random();
}
```

2. 设置参数

为了让 Bullets 类更加面向对象,这里还定义了两个 public 方法,外部类可以调用这两个方法来设置子弹的数目,以及画布的宽度和高度,这两个方法非常简单,实际应用中,也可以将 Bullets 类的这些变量属性设置为 public,而由外部直接设置。这两个方法代码如下:

```
public void setBulletsNum(int BULLETS_NUM){
    this.BULLETS_NUM = BULLETS_NUM;
}
public void setCanvasSize(int CanvasWidth,int CanvasHeight){
    this.CanvasWidth = CanvasWidth;
    this.CanvasHeight = CanvasHeight;
}
```

3. 初始化子弹的状态

首先用一个二维数组来记录子弹的状态,前面的 i 代表第 i 个子弹,第二维数组用来表示该子弹的当前状态,这些状态如下:

- bullets[i][0] : 表示子弹的类型
- bullets[i][1] : 表示子弹的 x 坐标
- bullets[i][2] : 表示子弹的 y 坐标
- bullets[i][3] : 表示子弹的 x 方向速度
- bullets[i][4] : 表示子弹的 y 方向速度
- bullets[i][5] : 表示子弹的存活状态

子弹的类型有上、下、左、右四种,分别表示子弹飞入屏幕前的四种位置,这里用四个静态整型常量表示:

- BULLET_TYPE_LEF: 静态整型常量,值为 0,代表从屏幕左方飞入
- BULLET_TYPE_RIGHT: 静态整型常量,值为 1,代表从屏幕右方飞入
- BULLET_TYPE_TOP: 静态整型常量,值为 2,代表从屏幕上方飞入

- BULLET_TYPE_BOTTOM：静态整型常量，值为 3，代表从屏幕下方飞入

这四种类型其实决定了子弹的初始位置和移动方向，例如从上方飞入的子弹，其起始坐标 y 分量应该为负或者一个比较小的、靠近屏幕上方的值（按照 Canvas 的坐标系统，左上方为坐标原点，屏幕处于第一象限）；因为子弹向下飞入屏幕，所以子弹 y 方向的速度应该为正值，其他类型也可以得到类似的结论，根据这些特性，就可以按照一定规则随机取子弹起始坐标和速度了。下面的代码将初始化一个子弹的所有初始属性：

```
bullets[i][0] = (random.nextInt() & 0x7fffffff) % 4; //采用随机值取子弹的类型，范围为[0, 3]（包含端点）
bullets[i][5] = ALIVE; //将所有的子弹都设置为活动的
switch (bullets[i][0]) {
    case BULLET_TYPE_LEFT: //子弹类型为从屏幕左方飞入
        bullets[i][1] = -frameWidth; //子弹的 x 坐标为较小的负数（屏幕左边）
        bullets[i][2] = (random.nextInt() & 0x7fffffff) % CanvasHeight; //随机取子弹的 y 坐标（0 ~ CanvasHeight）
        bullets[i][3] = (random.nextInt() & 0x7fffffff) % 3 + 1; //随机取子弹 x 方向的速度，范围为[1, 3]
        bullets[i][4] = (random.nextInt()) % 3; //随机取子弹 y 方向的速度，范围为[- 2,2]
        break;
    case BULLET_TYPE_RIGHT: //子弹类型为从屏幕右方飞入
        bullets[i][1] = CanvasWidth + frameWidth; //子弹的 x 坐标为略大于屏幕宽度的正数（屏幕右边）
        bullets[i][2] = (random.nextInt() & 0x7fffffff) % CanvasHeight; //随机取子弹的 y 坐标（0 ~ CanvasHeight）
        bullets[i][3] = ((random.nextInt() & 0x7fffffff) % 3 + 1) * -1;
        //随机取子弹 x 方向的速度，范围为[- 3, - 1]之间
        bullets[i][4] = (random.nextInt()) % 3; //随机取子弹 y 方向的速度，范围为[- 2,2]
        break;
    case BULLET_TYPE_TOP: //子弹类型为从屏幕上方向飞入
        bullets[i][1] = (random.nextInt() & 0x7fffffff) % CanvasWidth;
        //随机取子弹的 x 坐标为[0, CanvasWidth]之间的正数
        bullets[i][2] = -frameHeight; //子弹的 y 坐标为一个较小的负数（处于屏幕上方）
        bullets[i][3] = (random.nextInt()) % 3; //随机取子弹 x 方向上的速度范围为[- 2, 2]之间
        bullets[i][4] = (random.nextInt() & 0x7fffffff) % 3 + 1; //随机取子弹 y 方向的速度，范围为[1,3]之间
        break;
    case BULLET_TYPE_BOTTOM: //子弹类型为从屏幕下方飞入
        bullets[i][1] = (random.nextInt() & 0x7fffffff) % CanvasWidth;
        //随机取子弹的 x 坐标为[0, CanvasWidth]之间的正数
        bullets[i][2] = CanvasHeight + frameHeight;
        //子弹的 y 坐标为一个略大于屏幕高度的正数（处于屏幕下方）
        bullets[i][3] = (random.nextInt()) % 3; //随机取子弹 x 方向上的速度范围为[- 2, 2]之间
        bullets[i][4] = ((random.nextInt() & 0x7fffffff) % 3 + 1) * -1;
        //随机取子弹 y 方向的速度，范围为[- 3, - 1]之间
        break;
}
```

使用上面的方法可以随机设置子弹的位置和速度，使得这些子弹具有人工智能的色彩，但是有些情况下，某个方向的速度会出现 0 的情况，也就是子弹沿着 x 方向或者 y 方向移动，影响了子弹随机运动的视觉效果，为了避免这种情况的发生，将检测子弹某方向速度出现 0 的情况，并且给该方向设置一个默认速度，修正子弹运行轨迹的代码如下：

```
if(bullets[i][3]==0) //如果子弹的 x 方向上的速度为 0
{
    if(bullets[i][0] == BULLET_TYPE_RIGHT||bullets[i][0] == BULLET_TYPE_BOTTOM)
        bullets[i][3] = -DEFAULT_SPEED; //根据子弹类型设置 x 方向上速度为默认值
    else bullets[i][3] = DEFAULT_SPEED;
```

```

    }
    if(bullets[i][4]==0)                //如果子弹的 y 方向上的速度为 0
    {
        if(bullets[i][0] == BULLET_TYPE_RIGHT||bullets[i][0] == BULLET_TYPE_BOTTOM)
            bullets[i][4] = -DEFAULT_SPEED;    //根据子弹类型设置 y 方向上速度为默认值
        else bullets[i][4] = DEFAULT_SPEED;
    }

```

上面已经初始化了一颗子弹的属性，对于屏幕上的多个（数目为 BULLETS_NUM）子弹，可以使用遍历数组的方法来一一设定：

```

bullets=new int[BULLETS_NUM][6];        //根据子弹数目创建子弹数组
for (int i = 0; i < bullets.length; i++) {
    initBullet(i);                      //遍历数组，初始化所有子弹的类型、初始位置、速度
}

```

4. 子弹的移动

游戏中的子弹的运动看似杂乱无章，事实上，当子弹初始化完毕后，它的运动轨迹就已经确定下来了，子弹每帧的位置都是根据初始位置和速度计算出来的，实现的代码非常简单就是当前帧的 x、y 轴坐标分别加上相应 x、y 方向的速度，就得到下一帧的 x、y 坐标，也就是下一帧子弹的位置。还注意到，不要让子弹超出屏幕，当触及屏幕边界时，就将某个方向速度的值取反（也即该方向反方向运动），从视觉上就造成子弹碰撞屏幕边界并且反弹的效果。子弹运动的代码如下：

```

private void tick(int i){
    bullets[i][1]+=bullets[i][3];        //x 方向上根据初始位置和速度进行移动
    bullets[i][2]+=bullets[i][4];        //y 方向上根据初始位置和速度进行移动
    if(bullets[i][1]<-5 || bullets[i][1]>CanvasWidth+5){ //当超出屏幕的左右边界时
        bullets[i][3]*=-1; //x 方向的速度取反
    }
    if(bullets[i][2]<-5 || bullets[i][2]>CanvasHeight+5){ //当超出屏幕的上下边界时
        bullets[i][4]*=-1; //y 方向的速度取反
    }
}

```

和子弹的初始化一样，上面的代码完成了第 i 颗子弹的运动，想要所有的子弹运动起来，只要遍历子弹数组，对所有子弹做相同处理即可：

```

public void tick(){
    for (int i = 0; i < bullets.length; i++) {
        tick(i);                      //遍历子弹数组，移动所有子弹
    }
}

```

5. 子弹的绘制

子弹的绘制格外简单，只要将子弹用 setPosition 方法放置到指定位置，然后调用父类 Sprite 类的 paint 方法即可，这样将子弹设置到不同的位置，然后多次绘制，屏幕上就出现了多个子弹的图形，玩家感觉到的就好像有很多子弹在移动，而从幕后程序的角度来看，这些子弹仅仅只有一个子弹对象，这也是子弹类定义了复杂的数组，来标识这些“虚拟”子弹状态的用意所在。下面代码将第 i 颗子弹设置到当前位置，然后加以绘制：

```
private void draw(Graphics g,int i){
    setPosition(bullets[i][1],bullets[i][2]);    //根据数组内容，设置子弹的位置
    paint(g);                                    //绘制子弹
}
```

多个子弹的绘制只需遍历子弹数组，从数组中获得子弹的绘制位置，然后绘制的方法同样是调用 Sprite 类的 paint 方法，这里添了一句代码检测子弹是否处于激活状态，如果不是，则跳过不进行绘制，事实上这句代码在这个游戏中没有实际意义，但是有可能在别的使用到子弹的程序中用到，例如子弹移动到屏幕之外或和物体发生碰撞爆炸，此时应该将子弹的状态设置为非激活状态。绘制所有子弹的代码如下：

```
public void draw(Graphics g) {
    for (int i = 0; i < bullets.length; i++) {    //遍历子弹数组，绘制每颗子弹
        if(bullets[i][5] != ALIVE){              //如果子弹处于非激活状态，则跳过
            continue;
        }
        draw(g,i);                               //绘制第 i 颗子弹
    }
}
```

6. 碰撞检测

在游戏的运行中，始终要检测子弹和逃亡飞机是否发生碰撞，这里使用 Sprite 类的 collidesWith 方法来检测精灵与精灵之间的碰撞。飞机为不规则形体，因此这里使用的是像素级碰撞检测。注意在检测之前应当把子弹移动到相应位置。

技巧：可以对碰撞过程进行改进，例如以飞机为中心划分一个区域，然后读取子弹数组，只有位置值在这个区域内的子弹，才调用 setPosition 方法和 collidesWith 方法检测碰撞。

```
public boolean collidesWith(Sprite s){
    for (int i = 0; i < bullets.length; i++) {    //遍历子弹数组
        if(bullets[i][5] != ALIVE){              //如果子弹处于非激活状态则进行下一轮循环
            continue;
        }
        setPosition(bullets[i][1],bullets[i][2]); //将子弹设置到子弹数组所指定位置
        if(collidesWith(s,true)){                //和逃亡飞机进行像素级的碰撞检测，如果发生碰撞则返回
            return true;
        }
    }
    return false;                                //如果未发生碰撞，则返回 false
}
```

14.2.7 逃亡飞机 Escapee 类的实现

逃亡飞机就是玩家所控制的对象，玩家通过键盘来控制它的上下左右移动。Escapee 类是精灵 Sprite 类的子类，因此它继承了 Sprite 类的很多方法，例如设置位置、设置帧、以及移动位置，所需要做的仅仅是计算位置和进行边界检测。

1. 定义变量

下面将用代码实现飞机的基本功能，在此之前先定义以下一些变量或者常量：

```
private boolean isAlive;           //飞机是否存活
private boolean isMove = false;    //是否移动
static final int SPEED = 3;        //静态常量,移动的速度
static final int UP = 0;           //静态常量,向上移动
static final int LEFT = 1;         //静态常量,向左移动
static final int DOWN = 2;         //静态常量,向下移动
static final int RIGHT = 3;        //静态常量,向右移动
private int frameWidth,frameHeight; //精灵帧的宽度和高度
private int CanvasWidth,CanvasHeight; //画布的宽度和高度
```

2. 初始化

飞机的初始化分为几个小方法，需要初始化的内容为游戏帧的宽度和高度、画布的宽度和高度、定义参考点、设置生命（是否存活）以及设置初始帧

```
public Escapee(Image image,int frameWidth,int frameHeight)
{
    super(image,frameWidth,frameHeight);    //实现 Sprite 精灵类的构造函数
    this.frameWidth = frameWidth;           //获得帧的宽度
    this.frameHeight = frameHeight;         //获得帧的高度
    defineReferencePixel(frameWidth/2 , frameHeight/2 ); //设置参考点在中心点
    reset();
}
public void reset()
{
    isAlive=true;                           //初始情况下飞机存活
    setFrame(0);                            //默认帧设置为 0
}
public void setCanvasSize(int CanvasWidth,int CanvasHeight) //设置屏幕属性
{
    this.CanvasWidth = CanvasWidth;         //获得屏幕的宽度
    this.CanvasHeight = CanvasHeight;       //获得屏幕的高度
}
```

对于飞机是否存活提供了两个方法供外部设置和查询：

```
public void setAlive(boolean isAlive) //设置飞机是否存活
{
    this.isAlive = isAlive;
}
public boolean isAlive() //查询飞机是否存活
{
    return isAlive;
}
```

3. 飞机的绘制

飞机的绘制非常简单，这里调用了 Sprite 类的 paint 方法，注意在绘制之前先检测飞机是否是活动的，这步检测实际上在本游戏中意义不大，但在很多类似的游戏都需要那么做。

```
public void draw(Graphics g)
```

```

{
    if(!isAlive)return;           //如果飞机不存活则不进行绘制
    paint(g);                     //绘制飞机
}

```

4. 飞机移动

逃亡飞机的基本动作就是移动，移动操作只是按照用户键控获得飞机移动的方向，然后根据移动的方向和当前的速度计算出下一帧的位置，然后调用精灵类的 setPosition 方法将飞机放置到那个位置上，也可以直接调用 move 方法向那个方向移动若干个像素。需要注意的是飞机的移动不可能超出屏幕边界，因此应该在各个移动方向上设置边界检测。

```

public void move(int direction)
{
    if(direction == UP){           //如果方向向上
        move(0,-SPEED);           //调用 move 方法向上移动坐标，每次移动幅度为 SPEED 个像素
        if(getY()<0)setPosition(getX(),0);           //如果到达上边界（y 坐标为 0）则停止不动
        setFrame(0);
    }
    if(direction == DOWN){        //如果方向向下
        move(0,SPEED);            //调用 move 方法向下移动坐标，每次移动幅度为 SPEED 个像素
        //如果到达下边界（y 坐标靠近 CanvasHeight）则停止不动
        if(getY())>CanvasHeight-frameHeight)setPosition(getX(),CanvasHeight-frameHeight);
        setFrame(0);
    }
    if(direction == LEFT){        //如果方向向左
        move(-SPEED,0);           //调用 move 方法向左移动坐标，每次移动幅度为 SPEED 个像素
        //如果到达左边界（x 坐标靠近 0）则停止不动
        if(getX())<0)setPosition(0,getY());
        setFrame(1);
    }
    if(direction == RIGHT){       //如果方向向右
        move(SPEED,0);            //调用 move 方法向右移动坐标，每次移动幅度为 SPEED 个像素
        //如果到达右边界（x 坐标靠近 CanvasWidth）则停止不动
        if(getX())>CanvasWidth-frameWidth)setPosition(CanvasWidth-frameWidth,getY());
        setFrame(2);
    }
    isMove = true;
}

```

5. 帧的设置

飞机可以前后左右四个方向飞行，前后飞行的时候使用的是第 0 帧，往左飞行使用第 1 帧，往右飞行使用第 2 帧，这样左右飞行时候机翼会相应倾斜，看起来更加真实，但是在左右移动之后应当将帧重新置为 0（否则即使不动也保持机翼倾斜的动作），这里用了 isMove 标志为，当发生键盘事件会将 isMove 设置为真，而在下一个游戏帧中会把 isMove 重置过来，相应飞机的动画帧也设置为 0。

```

public void tick(){
    if(!isMove)setFrame(0);       //设置游戏帧为 0
    if(isMove)isMove = false;
}

```



```
}
```

14.2.8 游戏画布 escapeeCanvas 类的实现

游戏画布负责处理游戏的大部分工作，包括检测用户输入、处理游戏逻辑和绘制游戏画面，所有这些工作都放置在游戏线程中完成。escapeeCanvas 是 GameCanvas 的子类并且实现了 Runnable

1. 游戏变量

游戏画布中定义了大量的变量来代表各个游戏成分，例如地图、精灵的状态和属性。这些变量如下

```
static final int BULLETS_NUM = 50;           //子弹数目
static final int UP = 0;                     //代表玩家按下了游戏上键
static final int LEFT = 1;                   //代表玩家按下了游戏左键
static final int DOWN = 2;                   //代表玩家按下了游戏下键
static final int RIGHT = 3;                  //代表玩家按下了游戏右键
private TiledLayer background;               //TiledLayer 类对象，代表游戏背景
private Escapee escapee;                    //Escapee 类的对象
private Bullets bullets;                     //Bullet 类的对象
private Sprite explosion;                    //爆炸的精灵对象
private int CanvasWidth = getWidth();        //获取屏幕的高度
private int CanvasHeight = getHeight();      //获取屏幕的高度
private Image image;                         //Image 对象，用来加载图片资源
private static final int MILLIS_PER_TICK = 50; //每个 tick 的毫秒时间
private final escapeeMIDlet midlet;
private boolean isGameOver = false;          //表明游戏是否结束
private boolean isCollidesWith = false;      //是否玩家飞机和子弹碰撞
private int explosionCnt = 4;                 //爆炸计数器
private final Graphics g;                     //Graphics 对象
private long gameDuration = 0;                //游戏的持续时间
private long startTime;                       //开始时间
private volatile Thread animationThread = null; //游戏线程
```

2. 构造函数

构造函数主要完成加载资源和初始化游戏数据的工作，获得 midlet 的引用，以及实现父类 GameCanvas 的构造函数等。其代码如下：

```
escapeeCanvas(escapeeMIDlet midlet)
{
    super(true);                             //抑制非游戏键盘
    this.midlet = midlet;
    setFullScreenMode(false);                 //非全屏模式
    g = getGraphics();
    load();                                   //加载游戏资源
    init();                                   //初始化游戏数据
}
```

3. 加载游戏资源

加载游戏资源包括初始化游戏的各个部分：地图、子弹、飞机、爆炸图片等。下面来一步完成这些初始化工作。

加载地图，这里的地图使用 TiledLayer 实现，为了增加游戏的移植性，TiledLayer 类的地图数组是根据屏幕的实际大小计算出来的，这样就不会出现屏幕某些部分出现空白的情况。加载图片并创建游戏地图的代码如下：

```
image = midlet.createImage("/back_water.png");           //加载背景图片
int backColumns = CanvasWidth/image.getWidth()+1;        //计算地图数组的列数
int backRows = CanvasHeight/image.getHeight()+1;         //计算地图数组的行数
background = new TiledLayer(backColumns,backRows,image,image.getWidth(),image.getHeight());
int x,y;
for (int i = 0; i < backColumns*backRows; i++)
{
    x=i%backColumns;
    y=i/backColumns;
    background.setCell(x,y,1);                           //填充地图数组，这里只有一张图片
}
```

飞机类 Escapee 是 Sprite 类的子类，这里加载飞机图片，并告知飞机帧的宽度和高度、屏幕的大小尺寸，并在开始时把飞机设置在屏幕的正中央。

```
image = midlet.createImage("/Escapee.png");              //加载图片
escapee = new Escapee(image,image.getWidth()/3,image.getHeight());
escapee.setCanvasSize(CanvasWidth,CanvasHeight);         //设置画布尺寸
```

加载加载子弹的图片，创建子弹对象，同时设置子弹数目，以及屏幕的大小尺寸。

```
image = midlet.createImage("/bullet.png");               //加载图片
bullets = new Bullets(image,image.getWidth(),image.getHeight());
bullets.setBulletsNum(BULLETS_NUM);
bullets.setCanvasSize(CanvasWidth,CanvasHeight);         //设置画布尺寸
```

此外还需要加载爆炸图片，并用此创建一个 Sprite 对象，用以演示子弹和飞机碰撞的爆炸效果。

```
image = midlet.createImage("/explosion.png");
explosion = new Sprite(image,image.getWidth()/4,image.getHeight());
```

4. 初始化游戏数据

需要初始化的游戏数据包括子弹的初始位置和速度、飞机的起始位置、爆炸的可见属性等。和加载资源不同的是，加载资源在构造函数里完成，仅仅加载一次，而初始化游戏数据则每次开始一个新游戏就被调用，相当于“清除”上一次游戏的痕迹（例如飞机的位置等属性已经改变）。初始化的代码如下：

```
bullets.initBullets();                                   //初始化子弹的方位和速度
escapee.setPosition(CanvasWidth/2,CanvasHeight/2);     //设置飞机的初始位置
escapee.setAlive(true);                                 //飞机初始情况下为可活动的
explosion.setVisible(false);                             //设置爆炸动画为不可见
explosion.setFrame(0);                                  //设置爆炸动画的初始帧
explosionCnt = 4;                                       //爆炸计数器
isCollidesWith = false;                                //布尔值为 false，表明没有碰撞发生
isGameOver = false;                                    //表示游戏没有结束
```

```
gameDuration = 0;
```

```
//游戏持续时间
```

5. 游戏线程

游戏的线程主要由三个部分组成：检测键盘输入（与玩家交互），更新游戏场景（处理游戏逻辑），绘制游戏画布。基本上所有的 2D 游戏都可以按照这三个模块来设计，三个模块相互联系又互相联系，按照模块的思想来实现游戏线程将更加面向对象和便于理解。如果程序运行快于预期速度会使玩家感觉到抖动，从而影响游戏效果，因此还需要检测每帧的运行时间，如果过快则等待一段时间，如果过慢则挂起当前线程允许其他线程执行。线程的模块如图 14-13 所示。

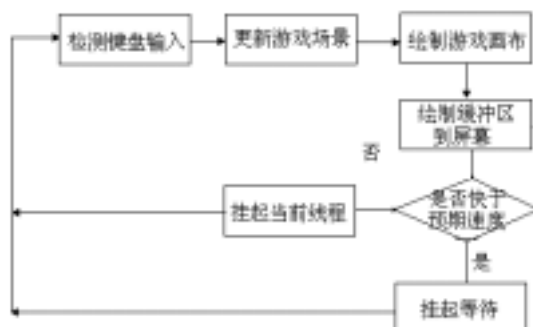


图 14-13 游戏线程模块图

游戏线程的代码如下：

```
public void run()
{
    Thread currentThread = Thread.currentThread();           //获取当前线程
    try
    {
        while (currentThread == animationThread)
        {
            long startTime = System.currentTimeMillis();      //获得游戏开始时间
            if (isShown())                                     //如果画布可见
            {
                input();                                       //用户输入
                tick();                                         //游戏逻辑
                draw();                                         //绘制画布
            }
            long timeTaken = System.currentTimeMillis() - startTime; //计算没次循环所花费的时间
            if (timeTaken < MILLIS_PER_TICK)
            {
                synchronized (this)
                {
                    wait(MILLIS_PER_TICK - timeTaken);        //等待
                }
            }
            else
            {

```

```

        currentThread.yield();
    }
}
catch (InterruptedException e){ }           //处理线程中断异常
}

```

启动线程的方法如下，用游戏画布构造了一个新的游戏线程，调用线程的 start 方法，并且因为 start 方法可能多次被调用（例如游戏暂时中止），所以要恢复某些中止前的游戏数据，这里是用中止线程之前保存的游戏持续时间（gameDuration）来恢复玩家的游戏开始时间 startTime，注意这里的 startTime 为“游戏”（忽略掉中间停止时间）开始的时间，而非线程开始时间，因为游戏需要记录的是玩家玩游戏的时间。

```

public synchronized void start()
{
    animationThread = new Thread(this);           //创建一个新的游戏线程
    animationThread.start();                       //开始线程
    startTime = System.currentTimeMillis() - gameDuration; //恢复游戏的开始时间
}

```

中止线程的方法如下，中止的方法是销毁画布的游戏线程，但需要注意的是销毁之前需要保存 gameDuration 时间，以便游戏重新开始时进行恢复。

```

public synchronized void stop()
{
    gameDuration = System.currentTimeMillis() - startTime; //保存游戏持续时间
    animationThread = null;                                //销毁线程
}

```

6. 游戏逻辑

游戏的逻辑比较简单，因为把各个游戏要素例如飞机、子弹都封装到各自的逻辑（tick 方法）中，只需要让飞机和子弹各自运动，然后检测它们之间是否有碰撞时间发生。飞机的运动根据玩家的键盘操作来控制，子弹的运动按照初始位置和速度来推算，一旦发生碰撞则发出爆炸声音并用爆炸精灵在碰撞点逐帧播放爆炸动画。游戏逻辑的实现如下所示：

```

private void tick()
{
    escapee.tick(); //飞机的运动
    if(!isCollidesWith)bullets.tick(); //如果未碰撞，子弹运动
    if(isCollidesWith&&(explosionCnt!=0)){//用 explosionCnt 来计数，当爆炸动画播放完，游戏结束
        explosion.setFrame((explosionCnt-1));
        explosionCnt--;
        if(explosionCnt == 0)isGameOver = true;
    }
    if(bullets.collidesWith(escapee)){ //如果子弹和飞机发生碰撞
        isCollidesWith = true; //置标志位，发生碰撞
        escapee.setAlive(false); //置标志位。飞机不存活
        SoundEffects.getInstance().startBlastSound(); //播放爆炸声音
        explosion.setPosition(escapee.getRefPixelX()-explosion.getWidth()/2, //定位爆炸点
            escapee.getRefPixelY()-explosion.getHeight()/2);
        explosion.setVisible(true); //爆炸精灵可见
    }
}

```

```

        if(isGameOver){                                     //如果游戏结束
            long time = (System.currentTimeMillis() - startTime); //记录玩家的持续时间
            midlet.GameCanvasGameOver(time,BULLETS_NUM);      //绘制游戏结束画面
        }
    }
}

```

7. 游戏事件

这个游戏的玩家操作比较简单，仅仅控制飞机的上下左右移动，相应地，游戏事件处理也比较简单，只是在玩家还存活的情况下，调用 escapee 对象的 move 方法，这个方法和 Sprite 类的 move 方法区别不大，主要是涉及到边界检测和动画帧的设置。

```

private void input() {
    int keyStates = getKeyStates();                //获取键盘状态
    if(escapee.isAlive()){                          //如果飞机处于存活状态
        if ((keyStates & LEFT_PRESSED) != 0) escapee.move(LEFT);    //左键按下，飞机左移
        else if ((keyStates & RIGHT_PRESSED) != 0) escapee.move(RIGHT); //右键按下，飞机右移
        else if ((keyStates & UP_PRESSED) != 0) escapee.move(UP);    //上键按下，飞机上移
        else if ((keyStates & DOWN_PRESSED) != 0) escapee.move(DOWN); //下键按下，飞机下移
    }
}

```

8. 非游戏事件

非游戏事件指的是玩家按下非游戏键盘，例如井号、星号，Clear 键和系统软键，这些按键通常具有负值的键码，因此按键按下将调用 keyPressed，如果键码小于 0 则停止线程并且显示游戏菜单。

```

public void keyPressed(int keyCode)
{
    if (keyCode < 0)                                //如果键码小于 0，代表按下了非游戏键盘
    {
        stop();                                    //停止游戏线程
        midlet.GameCanvasMenu();                  //显示游戏菜单
    }
}

```

9. 绘制游戏画布

游戏的绘制比较简单，因为游戏中采用的 GameAPI，大部分的对象都有各自的 paint 方法，直接调用就好了，其中 bullets 对象自定义了 draw 方法，实际上根据子弹的不同位置绘制了 50 次。爆炸精灵虽然也调用了 paint 方法，但注意它的可见属性是可变的，只有在子弹和飞机碰撞之后才能看到。此外当子弹和飞机碰撞演示爆炸效果的同时还将绘制一幅游戏结束的图片。如图 14-14 所示。

```

private void draw()
{
    background.paint(g);                            //绘制背景
    escapee.paint(g);                                //绘制飞机
    bullets.draw(g);                                 //绘制子弹
    explosion.paint(g);                              //绘制爆炸精灵
    if(isCollidesWith&&!isGameOver)                 //如果碰撞，绘制游戏结束图片
}

```

```

g.drawImage(image, CanvasWidth/2,CanvasHeight/2,g.VCENTER|g.HCENTER);
flushGraphics();           //将缓冲中的内容绘制到屏幕上
}

```

1.4.2.9 结束屏幕 GameOverScreen 类的实现

当飞机和子弹碰撞后将显示该屏幕。该屏幕显示游戏时间和等级，并指出这是否是新的最好时间，或者给游戏者显示出当前最好纪录。当它出现时，屏幕闪烁一秒钟，并播放两个短 MIDI 曲调之一。游戏结束画面如图 14-15 所示。

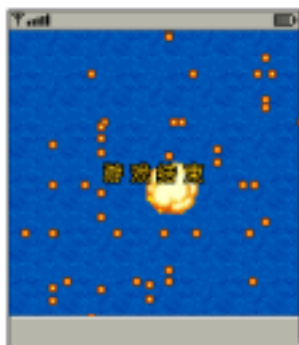


图 14-14 游戏结束画面

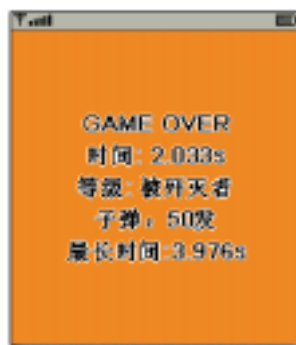


图 14-15 游戏结束画面

游戏结束画面的文本绘制方法和闪屏的一样，在此只给出等级判断和返回菜单的代码。游戏结束后，将先检测是否当前的最好成绩，如果是则播放高分音乐，如果不是则播放一般的游戏结束音乐：

```

if (midlet.checkBestTime(time))           //如果是最好成绩
{
    wasBestTime = true;
    bestTime = time;                      //将 bestTime 替换为当前成绩
    SoundEffects.getInstance().startHighScoreSound(); //播放高分音乐
    midlet.vibrate(1000);                 //手机震动
}
else
{
    wasBestTime = false;                  //如果不是最好成绩
    bestTime = midlet.getBestTime();       //获得最好成绩
    SoundEffects.getInstance().startGameOverSound(); //播放游戏结束音乐，不震动
}
midlet.flashBacklight(1000);              //闪烁背景灯，持续 1 秒
}

```

在得出游戏成绩后，还可以判断玩家的水平，这里只是比较粗略地给出等级划分，读者有兴趣可以对此加以改进，提高游戏的挑战性和趣味性：

```

String level;
if(Time_Second<5)
{
    level = "被歼灭者";                  //等级 1
}else if(Time_Second<15)

```

```

{
    level = "初级飞行员";           //等级 2
}else if(Time_Second<20)
{
    level = "中级飞行员";           //等级 3
}else if(Time_Second<25)
{
    level = "高级飞行员";           //等级 4
}else if(Time_Second<30)
{
    level = "逃亡者";               //等级 5
}else
{
    level = "胜利大逃亡！";         //最高等级
}

```

当玩家按下键盘时，将回调 midlet 的 gameOverDone 方法，显示游戏菜单。

```

public void keyPressed(int keyCode)
{
    midlet.gameOverDone();
}

```

14.2.10 声音效果 SoundEffects 类的实现

该类的主要功能为使用 MIDP 2.0 Media API 播放三种声响效果：碰撞爆炸，和游戏结束时的两个短 MIDI 曲调，在此写了两个通用的加载声音和播放声音的方法。

1. 构造函数和获取实例

与前面有所不同，使用 SoundEffects 对象将不采用 new 方法，而是使用静态访问 getInstance 来获取该类的唯一实例，SoundEffects 的构造方法是私有的，在构造方法里将加载游戏的爆炸声音。其代码如下：

```

private static SoundEffects instance;           //声明一个 SoundEffects 实例的全局变量
private SoundEffects()
{
    blastSoundPlayer = createPlayer("/blast.wav", "audio/x-wav"); //加载并准备播放爆炸声音
}
static SoundEffects getInstance()
{
    if (instance == null)                       //如果目前 SoundEffects 实例不存在
    {
        instance = new SoundEffects();         //构造一个新的实例，构造方法只调用一次
    }
    return instance;                           //返回 SoundEffects 类的唯一实例
}

```

2. 创建播放器

```

private Player createPlayer(String filename, String format)
{

```

```

    Player p = null;
    try
    {
        InputStream is = getClass().getResourceAsStream(filename); //以流的方式读取资源
        p = Manager.createPlayer(is, format);                      //根据流和文件格式创建播放器
        p.prefetch();
    }
    catch (IOException ioe)
    {
                                                                    // 捕获 IO 异常
    }
    catch (MediaException me)
    {
                                                                    //捕获媒体异常
    }
    return p;
                                                                    //返回 Player 对象
}

```

3. 播放声音

播放声音的代码如下，需要注意的是在播放之前都需要调用 stop 方法来中止声音（如果声音未播放则该方法无效）。

```

private void startPlayer(Player p)
{
    if (p != null)
    {
        try
        {
            //停止声音，并且重新从头开始重新播放
            p.stop();
            p.setMediaTime(0L);
            p.start();
        }
        catch (MediaException me)
        {
            //捕获媒体异常
        }
    }
}

```

使用 startPlayer 方法来播放游戏中的声音：

```

void startBlastSound()
{
    startPlayer(blastSoundPlayer); //播放子弹和飞机碰撞时发出的爆炸声音
}
void startGameOverSound()
{
    startPlayer(createPlayer("/gameover.mid", "audio/midi")); //播放游戏结束声音
}
void startHighScoreSound()
{
    startPlayer(createPlayer("/highscore.mid", "audio/midi")); //产生高分纪录时，播放高分声音
}

```



```
}
```

14.3 游戏的优化和改进

这里将对游戏进行更深层次的研究,涉及方面包括益智类游戏的程序框架、数字版权的保护以及如何定制游戏,在实际设计并实现手机游戏的过程中,所设计的方面还有很多,希望读者能更加细致的进行分析和思考。

14.3.1 逻辑层和表现层的分离

益智类游戏是移动设备比较擅长表现的游戏,相对于动作游戏的快节奏,益智类游戏的特点就是玩起来速度慢,比较幽雅,主要来培养玩家在某方面的智力和反应能力,此类游戏的代表非常多,比如牌类游戏,拼图类游戏,棋类游戏等等,总而言之,那种玩起来主要靠玩家动脑筋的游戏都可以被称为益智类游戏。

这类型的游戏以趣味性的思考为游戏的主轴,内容可以包罗万,思维模式也可朝物理性及逻辑性方向着眼,具代表性的是的“台湾十六张麻将”、“大富翁”、“强手棋”等,而棋盘式的思考方式著名的有“中国象棋”及“五子棋大师”等,这些游戏入手容易且不分男女老少皆喜欢的特性,使得益智型态的开发较有市场,成本也较低。

益智类游戏的开发需要注重的是逻辑层和表现层的分离。逻辑层处理所有的判断、推算,可能采用各种比较复杂的算法,而表现层则只管与玩家交互。逻辑层和表现层之间的接口应当越简单越好,有时候它们之间仅仅通信几个变量。

以光盘中五子棋程序为例,所有的绘制都放置在 FiveCanvas 类中,而所有的逻辑处理都放置在 FiveLogic 类中,游戏画布负责告知逻辑类玩家所走的棋路,然后由逻辑类运算得出计算机所走的棋路,并告知游戏画布让它进行相应的绘制。如图 14-16 所示为该游戏的画面。

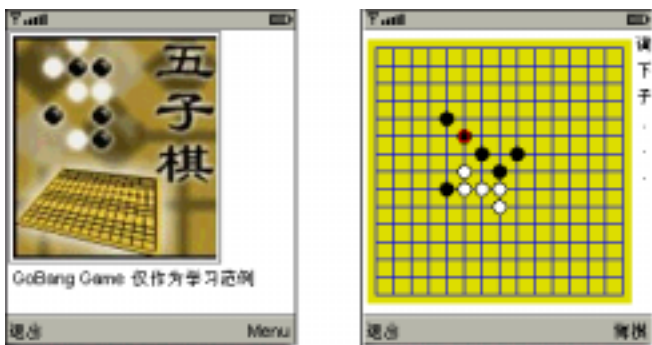


图 14-16 五子棋游戏的画面

MIDP 设备可能有不同的屏幕尺寸、不同的键盘布局以及不同的可用 API。在设计游戏时必须考虑这些不同点,以便获得移植性最好的游戏软件。

应用软件应该从系统获得屏幕的尺寸,避免在屏幕上绘画时对坐标使用硬代码。使用 Canvas 类的 getHeight 和 getWidth 方法能够完成上述任务。例如这里的棋盘实际尺寸是根据游戏画布的大小计算出来的,程序会计算出最适合当前游戏画布的棋盘大小和布局,如图

14-17 所示为四款不同大小的模拟器上的棋盘布局。此外一个良好的做法是让玩家既支持键盘操作也支持指针操作。

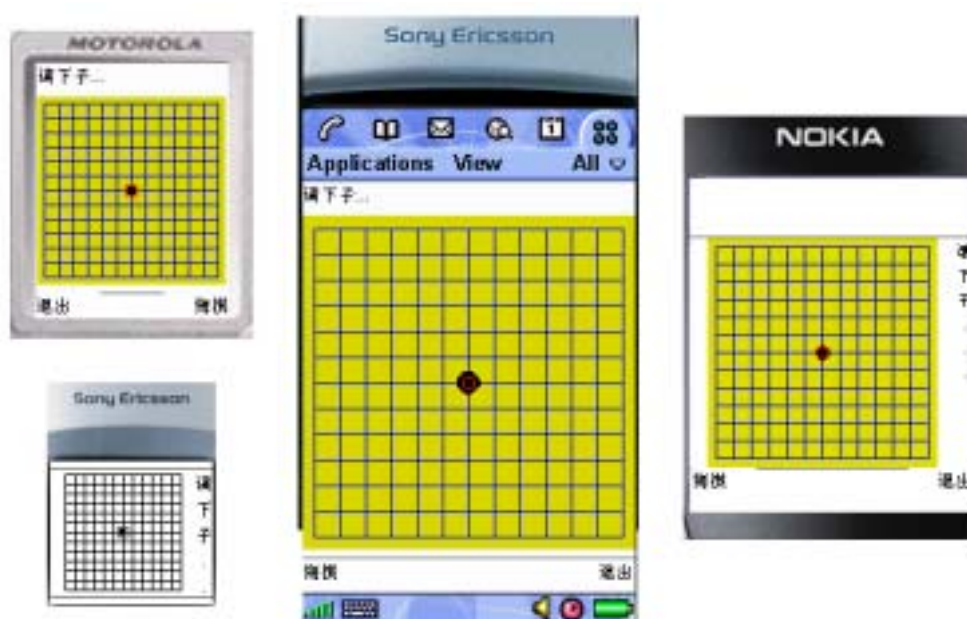


图 14-17 不同手机屏幕上的棋盘

14.3.2 玩家的定制

应当让游戏尽可能地可定制，如果玩家的水平不一，过于复杂或过于简单的游戏都会让游戏失去吸引力。从另一个角度来说，定制本身就是一种玩家和游戏之间的交互，无论结果是否合理，都会让玩家产生参与其中的感觉，从而增强对游戏的兴趣。

游戏所能定制的内容应该仔细设计，贴近玩家的需要而且又非常合理，这里给出了一个简单的例子，是关于对五子棋游戏的定制，这些定制方案也有可能在其他棋类游戏中使用到。

技巧：如果能把玩家定制结果保存起来，会让游戏更加人性化，而一个好的游戏往往体现在这些小细节里。

定义一个 Form 来添加这些游戏定制选项，在添加选项之前先读取 rms 记录，看玩家是否曾经进行了定制操作。这里有棋盘大小、电脑先行、游戏难度三种定制方案，实现定义了棋盘大小为 13×13 ，电脑先行，游戏难度等级为 1。

```
borderSize = 13;
computerFirst = true;
degree = 1;
```

然后调用 load 方法加载之前的定制内容，如果有则改变上面的默认值。Form 中使用 Gauge 组件来改变棋盘大小，使用复选 ChoiceGroup 组件来选择是否电脑先行，使用单选 ChoiceGroup 组件来设定游戏难度。其代码如下：

```
load();
frmOptions = new Form("游戏设置");
```

```

ggSize = new Gauge("棋盘大小: " + borderSize + " X " + borderSize, true, 10, borderSize - 9);
frmOptions.append(ggSize);
cgFirst = new ChoiceGroup(null, 2);
cgFirst.append("对方先行", null);
cgFirst.setSelectedIndex(0, computerFirst);
frmOptions.append(cgFirst);
cgDegree = new ChoiceGroup("难度等级:", 1);
cgDegree.append("简单", null);
cgDegree.append("中等", null);
cgDegree.append("较难", null);
cgDegree.setSelectedIndex(degree - 1, true);
frmOptions.append(cgDegree);
frmOptions.setItemStateListener(this);

```

此外，还添加了确定和取消 Command，如果侦听到确定软键被按下，则对当前的定制选项进行保存。

```

cmdOk = new Command("确定", 4, 2);
cmdCancel = new Command("取消", 3, 1);
frmOptions.addCommand(cmdOk);
frmOptions.addCommand(cmdCancel);
frmOptions.setCommandListener(this);

```

响应 Command 事件的方法如下：

```

public void commandAction(Command c, Displayable s)
{
    if(c == cmdOk)
    {
        borderSize = ggSize.getValue() + 9;
        if(borderSize > 19)
            borderSize = 19;
        if(borderSize < 9)
            borderSize = 9;
        computerFirst = cgFirst.isSelected(0);
        degree = cgDegree.getSelectedIndex() + 1;
        save();
        midFive.backHome();
    } else
        if(c == cmdCancel)
            midFive.backHome();
}

```

如果侦听到 Gauge 的内容发生变化，也相应改变旁边的说明文字：

```

public void itemStateChanged(Item item)
{
    if(item == ggSize)
    {
        int bs = ggSize.getValue() + 9;
        ggSize.setLabel("棋盘大小: " + bs + " X " + bs);
    }
}

```

记录的加载和保存代码如下：

```

private void load()

```

```

{
    try
    {
        RecordStore rs = RecordStore.openRecordStore("Options", false); //记录文件名为 Option
        if(rs.getNumRecords() > 0)
        {
            byte bs[] = rs.getRecord(1);
            if(bs.length >= 3)
            {
                borderSize = bs[0];
                if(borderSize < 9)
                    borderSize = 9;
                if(borderSize > 19)
                    borderSize = 19;
                computerFirst = bs[1] == 1;
                degree = bs[2];
                if(degree < 1)
                    degree = 1;
                if(degree > 3)
                    degree = 3;
            }
        }
        rs.closeRecordStore(); //关闭存储
    }
    catch(RecordStoreException _ex) { }
}

```

定制记录的保存，将玩家的定制选项保存到一个叫做 Option 的 rms 记录中：

```

private void save()
{
    try
    {
        RecordStore rs = RecordStore.openRecordStore("Options", true);
        byte bs[] = new byte[3];
        bs[0] = (byte)borderSize; //棋盘大小
        bs[1] = (byte)(computerFirst ? 1 : 0); //是否电脑先行
        bs[2] = (byte)degree; //难度等级
        if(rs.getNumRecords() > 0)
            rs.setRecord(1, bs, 0, 3); //设置存储内容
        else
            rs.addRecord(bs, 0, 3); //如不存在记录，则添加新的
        rs.closeRecordStore();
    }
    catch(RecordStoreException _ex) { }
}

```

编译、运行程序，其结果如图 14-18 所示。

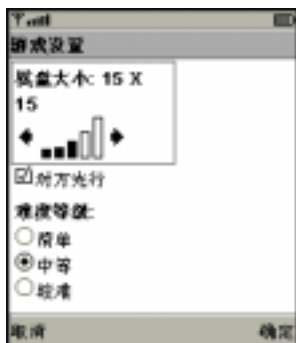


图 14-18 定制五子棋游戏

14.3.3 制作游戏的试玩版

要推广一个商用游戏，常用的做法是提供游戏的免费试玩，这就需要对程序进行一定的保护。虽然任何加密都有可能被专注的玩家破解，这里将讨论一种难度适中，对程序影响不大，但又不那么容易破解的方法来保护程序。

游戏的试玩版将是让潜在用户购买商用游戏的好办法，顾名思义，试玩版将限制游戏的某些功能或者限制游戏的使用时间。通常来说，试玩分为几种类型：

- 日期：指定一个试玩日期，如果当前日期超过试玩日期，则视为“超期”。虽然玩家可以修改手机的系统时间使游戏继续有效，但是所带来的麻烦也会让他们开始考虑下载游戏的正式版，此外，有的手持终端是根据所处的无线网络环境来更新时间的，更增加了玩家修改时间的麻烦。
- 天数：当用户第一次玩游戏，当前的日期将被记录，从所记录的日期到指定最大天数时视为“超期”。
- 次数：每次玩家启动游戏程序就会被记录，当超过一定的次数之后就视为“超期”。

可以注意到，后两者都使用了手机的记录保存功能，通常来说都会使用 RMS 系统，但是有的手机设备能够允许玩家删除或者清空这些存储内容，即使不是这样，有的玩家也能够轻松地获得程序使用的 RMS 或者删除 RMS 记录，甚至对这些记录进行读写。在这种情况下，可以使用网络服务功能，从服务器上读取超期数据。为了简单起见，这里还是使用本地的 RMS，毕竟大部分情况下玩家不会懂得如何删除 RMS 记录。

前面讲到，对超期与否的检测可能要用到超期日期，最大天数和最大启动次数。对于这些数据的初始值，一种方法是硬编码的形式嵌入到程序中，或者保存在某个文件中，但是当需要改动这些初始值时会非常麻烦，需要重新编译或者打包程序。一种可行的方法是把这些初始值放在 JAD 文件中，如果需要进行修改，只需要替换 JAD 文件即可，而不用经过复杂的打包等手续。在此自定义了 JAD 三条属性：

```
Expire-Date:20060101
```

```
Expire-Days:30
```

```
Expire-Plays:10
```

但是实际使用中观察到的超期初始值并非这样，而是：

```
Expire-Date:MjAwMzEwMTU=
```

显然这是因为 JAD 文件能够很容易地进行修改,对所采用的日期进行了一次加密,如果程序读取的数据无法解密或者为空,那么依然视程序已经超期。

在此虽然增加了超期限制但是无法保证玩家将程序转移到别的手机上,一种更高级的做法是使用 DRM (数字权限管理),即使被用户下载保存并散播给他人,没有得到数字节目授权中心的验证授权也无法播放,从而严密地保护了节目的版权。DRM 对手机游戏的保护体现在:

- 允许内容提供商定义游戏使用的规则(版权),用户必须按照这些规则进行内容的消费。
- 可以对一个游戏对象定义不同的版权,并制定不同的价格。如按照不同的使用次数、使用时间以及运行等不同操作定义不同的版权以供用户选择,为一系列新的商业模式的实现提供了可能,如按天出租游戏、按使用次数控制游戏的使用等。
- 通过对版权的控制,使内容的真正价值体现于版权,而非游戏对象本身。在这种情况下,内容可以在移动网内根据用户的喜好进行转发和传播,从而形成业务的流行,但内容的使用必须重新申请新的版权,从而既保证了业务的“传染性”,又保证了内容提供商的利益。
- 通过对版权的控制,DRM 使游戏对象的数字版权成为计费的来源。因此,不同的业务可以使用同一个 DRM 授权中心生成数字版权,并同时生成计费原始话单记录(CDR),从而简化了计费体系,堵塞了计费漏洞,保证了业务收入。

在试玩版的游戏中,可以限制游戏的某些功能,例如升级、纪录保存或者使用某种特殊的武器,这些可能使某些在体验过程中发生兴趣的玩家决定购买游戏的完全版本。

在光盘中有一个关于上述三种超期使用的例子。

14.3.4 游戏程序的注册

手机程序也可以像 PC 程序一样使用注册序列号来进行加密保护,虽然这种方法也容易被破解,但是如果加密算法较为复杂还是不那么容易的,例如这里利用程序的当前时间动态地生成序列号(加密算法的种子),并且计算结果保存在 rms 中。

```
serialNo = (int)(System.currentTimeMillis() % 0xf4240L);
```

如果玩家输入的数字与之算法匹配则表明程序被注册,同时开放游戏的某些受限操作。当然如果这个算法被人破译,就会产生类似 PC 游戏上的算号器。

```
import javax.microedition.lcdui.*;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
public class Reg implements CommandListener
{
    private FiveMIDlet midFive;
    private boolean registered = false;           //判断是否已经注册
    private static long nTriedMax = 0x989680L;
    private int nTriedLeft = 0x989680;
    private int serialNo;
    private Form frmReg;                          //注册表单
    private StringItem siInfo;                    //注册信息的字符串组件
    private TextField tfRegNo;
```

```

private Command cmdOk;
private Command cmdCancel;
public Reg(FiveMIDlet m)
{
    midFive = m;
    load();
    save();
}
public void play()
{
    if(!registered && nTriedLeft > 0)
        save();
}
public void register(Display display)
{
    frmReg = new Form("注册游戏"); //创建表单
    frmReg.append("您的序列号:[" + serialNo + "]\n" + "请把该号码告知作者并获得注册号");
    tfRegNo = new TextField("请输入注册号:", "", 8, 2); //用于输入注册号
    frmReg.append(tfRegNo);
    siInfo = new StringItem(null, null);
    frmReg.append(siInfo);
    cmdOk = new Command("确定", 4, 1); //创建确定软键
    cmdCancel = new Command("取消", 3, 2); //创建取消软键
    frmReg.addCommand(cmdOk); //添加确定软键
    frmReg.addCommand(cmdCancel); //添加取消软键
    frmReg.setCommandListener(this); //侦听软键事件
    display.setCurrent(frmReg); //显示注册表单
}
private void freeRes() //该方法用于释放资源，以便垃圾回收
{
    frmReg = null;
    cmdOk = null;
    cmdCancel = null;
    siInfo = null;
    tfRegNo = null;
}
public void commandAction(Command c, Displayable s)
{
    if(c == cmdOk)
    {
        String sRegNo = tfRegNo.getString();
        int regNo = 0;
        try
        {
            regNo = Integer.parseInt(sRegNo); //将用户输入转换为整数
        }
        catch(Exception _ex) { }
        if(isRegNoOk(regNo)) //如果注册成功，则保存并返回
        {
            registered = true;
            save();
        }
    }
}

```

```

        midFive.backHome();
        freeRes();
    } else
    {
        siInfo.setText("注册号不正确!请重新输入!");
    }
} else
if(c == cmdCancel) //如果用户按下取消软键，则返回
{
    midFive.backHome();
    freeRes();
}
}
private boolean isRegNoOk(int regNo) //如果 regNo 等于 serialNo，则注册成功
{
    return regNo == serialNo;
}
private void save()
{
    try //保存注册信息
    {
        byte bs[] = new byte[5];
        bs[0] = (byte)(registered ? 1 : 0);
        bs[1] = (byte)nTriedLeft;
        bs[2] = (byte)(serialNo >> 16 & 0xff);
        bs[3] = (byte)(serialNo >> 8 & 0xff);
        bs[4] = (byte)(serialNo & 0xff);
        RecordStore rs = RecordStore.openRecordStore("Reg", true);
        if(rs.getNumRecords() >= 1)
            rs.setRecord(1, bs, 0, 5);
        else
            rs.addRecord(bs, 0, 5);
        rs.closeRecordStore();
    }
    catch(RecordStoreException _ex) { }
}
private void load()
{
    try //读取序列号
    {
        RecordStore rs = RecordStore.openRecordStore("Reg", false);
        if(rs.getNumRecords() >= 1)
        {
            byte bs[] = rs.getRecord(1);
            if(bs.length >= 5)
            {
                registered = bs[0] == 1;
                nTriedLeft = bs[1];
                if(nTriedLeft > 10)
                    nTriedLeft = 10;
                int b1 = bs[2];
            }
        }
    }
}

```



```

        if(b1 < 0)
            b1 += 256;
        int b2 = bs[3];
        if(b2 < 0)
            b2 += 256;
        int b3 = bs[4];
        if(b3 < 0)
            b3 += 256;
        serialNo = (b1 << 16) + (b2 << 8) + b3;
    }
}
rs.closeRecordStore();
}
catch(RecordStoreException _ex) { }
}
}

```

编译、运行程序，其结果如图 14-19 所示。



图 14-19 程序的注册界面

14.4 本章小结

本章介绍了 2D 游戏的策划设计过程，以一个飞机游戏为范例，讲解了程序架构以及基本的 2D 简单动作类游戏的制作，其中子弹的数组管理、分数的记录和存储都是游戏中常用的技术。之后本章又对游戏制作进行了更多的讨论，这包括对益智类游戏架构的讨论、如何对程序版权进行保护的讨论、并提出了一些加强玩家定制功能的建议。单屏幕游戏在手机游戏中占有很大的比例，这类游戏的亮点在于一个好的创意，而这就期待读者来实现了。