

第 1 章

电脑游戏开发概述

电脑游戏 (Personal computer games, Computer games 或 PC games) 是指在电子计算机上运行的游戏软件。这种软件是一种具有娱乐功能的电脑软件。电脑游戏产业与电脑硬件、电脑软件、互联网的发展联系紧密。电脑游戏为游戏参与者提供了一个虚拟的空间,从一定程度上使游戏参与者可以摆脱现实世界,在另一个世界中扮演真实世界中无法扮演的角色。同时电脑多媒体技术的发展,使游戏带给了人们很多新的体验和享受。

1.1 电脑游戏的发展历史

电脑游戏的出现与 1960 年电子计算机进入美国大学校园关系密切。当时的环境培养出了一批电脑编程高手。1962 年一位叫斯蒂夫·拉塞尔的大学生在美国 DEC 公司生产的 PDP-1 型电子计算机上编制的《宇宙战争》(Space War) 是当时很有名的电脑游戏。一般认为,斯蒂夫·拉塞尔是电脑游戏的发明人。从 1970 年开始,随着电子计算机技术的发展,其成本越来越低。1971 年,被誉为“电子游戏之父”的诺兰·布什内尔发明了第一台商业化电子游戏机。不久他创办了世界上第一家电子游戏公司——雅达利公司 (ATARI)。1970 年,随着苹果电脑的问世,电脑游戏才真正开始了商业化的道路。此时,电脑游戏的图形效果还非常简陋,但是游戏的类型化已经开始出现了。

从 1980 年开始,多媒体技术也开始成熟,电脑游戏则成为这些技术进步的先行者。1985 年 9 月 13 日,日本任天堂公司发售了一款真正的游戏巨作——超级马里奥 (Super Mario),这款游戏讲述了一个意大利管子工打败魔王拯救世界迎娶公主的故事。任天堂公司凭借这台游戏机确立了自己游戏界霸主的地位。尽管这台游戏机也叫作 Computer,但是它却彻底抛弃了计算机的一部分特征,专心于游戏机平台的营造,电脑游戏和游戏机游戏从这时开始分道扬镳。

Windows 操作系统的出现给电脑游戏的制作带来了一场革命,游戏开始向注重感官刺激的 3D 方向发展。1992 年,3D Realms 公司发行了《德军总部 3D》,不久之后 Id Software 公司的《Doom》诞生,成为第一个被授权引用的商品化的引擎。1996 年,一款划时代的游戏作品诞生了,那就是被称为雷神之锤的《Quake》,与之前的第一人称射击游戏不同,这是一款真正意义上的 3D 游戏,它带给玩家一个比以往任何时候都要真实的 3D 虚拟世界。《Quake》不仅代表计算机游戏正式迈进了 3D 门槛,更是带来了电子竞技运动的新概念。

从 20 世纪 90 年代开始,即时战略类游戏成为 PC 上最引人入胜的游戏类型。从最早的由 Westwood 公司开发的《C&C》和《红色警戒》,到脱胎于《文明》的《帝国时代》,再到集大成

者《星际争霸》，即时战略游戏的发展随着各种游戏概念的提出和创新，达到了其发展史上的一个高峰。与此同时，动作游戏（ACT 游戏）在 3D 技术不断进步的条件下也获得了新生，后来的《古墓丽影》系列、《波斯王子》系列等均成为计算机游戏的经典作品。随着 3D 技术的广泛应用，动作游戏的规则也发生了微妙变化，原来的动作游戏只能在 2D 平面上运行，因此一些真实的动作无法表现出来，而 3D 技术的引入则带来了新的游戏规则，人物除了可以前后左右自由平移外，还可以通过自己的视角来观察，并能创造出新的动作。在这次 3D 革命中，唯一改变不大的游戏类型是角色扮演游戏（RPG）。尽管很多游戏机上的 RPG 已经实现了完全的 3D，但实质上这些 3D 游戏除了效果上得到提升之外，并没有给游戏带来本质上的改变和提升。

进入 20 世纪 90 年代后期，随着电脑软、硬件技术的进步以及因特网的广泛使用为电脑游戏的发展带来了强大的动力。进入 21 世纪，网络游戏成为电脑游戏的一个新的发展方向。

网络游戏英文名称为 Online Game，又称“在线游戏”，简称“网游”。不同于单机版游戏，网络游戏的玩家必须通过互联网的连接来进行多人游戏。网络游戏一般指由多名玩家通过计算机网络在虚拟的环境下对人物角色及场景按照一定的规则进行操作以达到娱乐和互动目的的游戏。

网络游戏可以分为大型角色扮演类网络游戏（如神话 2、大话西游、大唐豪侠、梦幻西游等）、休闲（对战）网络游戏（如疯狂赛车 II、CS、魔兽争霸、彩虹岛、泡泡堂等）和棋牌网络游戏（联众世界棋牌游戏、QQ 棋牌游戏）3 种类型。大型角色扮演类网络游戏，使所有用户都存在于一个大型的虚拟世界中，用户可以拥有不同特点的角色来体验虚拟生活，游戏本身是持续发展的。休闲（对战）网络游戏，大多是采用平台竞技方式进行的，游戏以“局”的形式存在，每局游戏参与的用户数量相对较少。棋牌类网络游戏，与平台对战网络游戏类似，该种游戏也以平台为基础，区别在于棋牌类游戏往往从平台自身下载，无需单机游戏支持，内容也多以棋牌等小型互动游戏为主。

根据游戏提供形式的不同，网络游戏分为客户端网络游戏和网页游戏两种类型。客户端网络游戏，是指需要在电脑上安装游戏客户端软件才能运行的游戏。这种类型的游戏是由公司所架设的服务器来提供游戏，而玩家则是由公司所提供的客户端来连上公司服务器以进行游戏，而现在称之为网络游戏的大部分都属于此类型。此类游戏的特征是大多数玩家都会有一个专属于自己的角色（虚拟身份），而一切角色资料以及游戏资讯均记录在服务器端。此类游戏大部分来自欧美以及亚洲地区，这种类型的游戏有 World of Warcraft（魔兽世界）（美国）、穿越火线（韩国）、EVE（冰岛）、战地（Battlefield）（瑞典）、最终幻想 14（日本）、天堂 2（韩国）、梦幻西游（中国）等。

网页游戏又称 Web 游戏，是指用户可以直接通过互联网的浏览器玩的网络游戏，它不需要安装任何客户端软件。只需用 IE 浏览器打开网页，10 秒钟即可进入游戏，不存在硬件配置不够的问题，最重要的是关闭或者切换极其方便，尤其适合上班族。网页游戏的类型及题材也非常丰富，典型的类型有角色扮演（功夫派）、战争策略（七雄争霸）、社区养成（洛克王国）、模拟经营（范特西篮球经理）、休闲竞技（弹弹堂）等。

1.2 电脑游戏的类型

目前常见的电脑游戏的类型有动作游戏、传统益智游戏、体育游戏、策略游戏、休闲游戏和角色扮演类游戏等。不同类型的游戏有着自身的特点，每一种类型的游戏都有一定的支持人群。开发出一款良好的游戏，了解各种类型游戏的基本特点是很有必要的。

游戏厂商在制作游戏时，首先要定位游戏类型，例如 SLG（策略类）游戏通常锻炼玩家的智力和策略，玩家通过与电脑进行较量，取得各式各样的胜利。RPG（角色扮演类）游戏通常是指玩家通过控制游戏中的主角进行升级、成长来完成游戏，体验厮杀的快感和故事情节的跌宕。下面介绍几种常见的游戏类型。

① RPG（角色扮演类）游戏

角色扮演类游戏(Role Playing Game, RPG)分为两种：动作角色扮演游戏(Action Role Playing Game, ARPG)和模拟角色扮演游戏（ Simulation Role Playing Game, SRPG ）。

标准 RPG 类游戏的主要特征是主角在与怪物战斗时进入特定的战斗画面，PC 上著名的游戏有《仙剑奇侠传》系列，还有北京捷通华声公司开发的《神雕侠侣》（如图 1-1 所示）和上海乐游公司开发的《封神榜》（如图 1-2 所示）。



图 1-1 《神雕侠侣》截图



图 1-2 《封神榜》截图

ARPG 类游戏的主要特征是战斗的画面都在地图上进行，而且还能体验标准 RPG 类游戏中精彩的剧情，例如北京掌中米格公司开发的《热血三国之猛将风云》（如图 1-3 所示）。

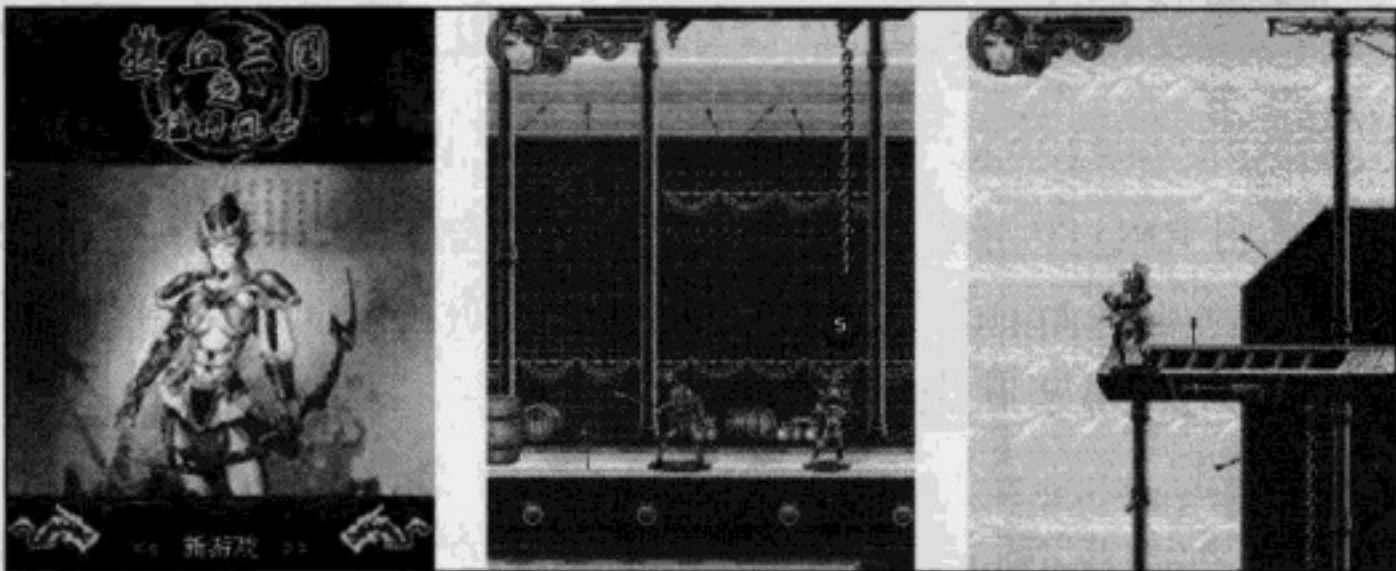


图 1-3 《热血三国之猛将风云》截图

SRPG 类游戏非常类似于策略类游戏，主要区别是 SRPG 类游戏拥有完善的剧情。



国内大部分手机玩家都喜欢玩 RPG 类型的游戏，因此如今国内此类型游戏比较多，也比较成熟。

② SLG（策略类）游戏

SLG（Simulation Game）类游戏主要是玩家通过思考执行命令去执行游戏，此类游戏不但可以锻炼玩家的智力，而且也可以体会其中的乐趣。PC 上比较流行的单机版经典游戏有《魔兽争霸》以及 In-Fusio 公司模仿其制作的《魔兽（白金攻防版）》（如图 1-4 所示）。



图 1-4 《魔兽（白金攻防版）》截图

③ AVG（冒险类）游戏

AVG（Adventure Game）大部分都在关卡设定以及游戏节奏上加大投入，游戏时间比较短，并强化其游戏动作的快感和乐趣。大多数 AVG 类型的游戏采用横屏卷轴式方法，例如北京华娱无线公司开发的《野人岛》（如图 1-5 所示）。

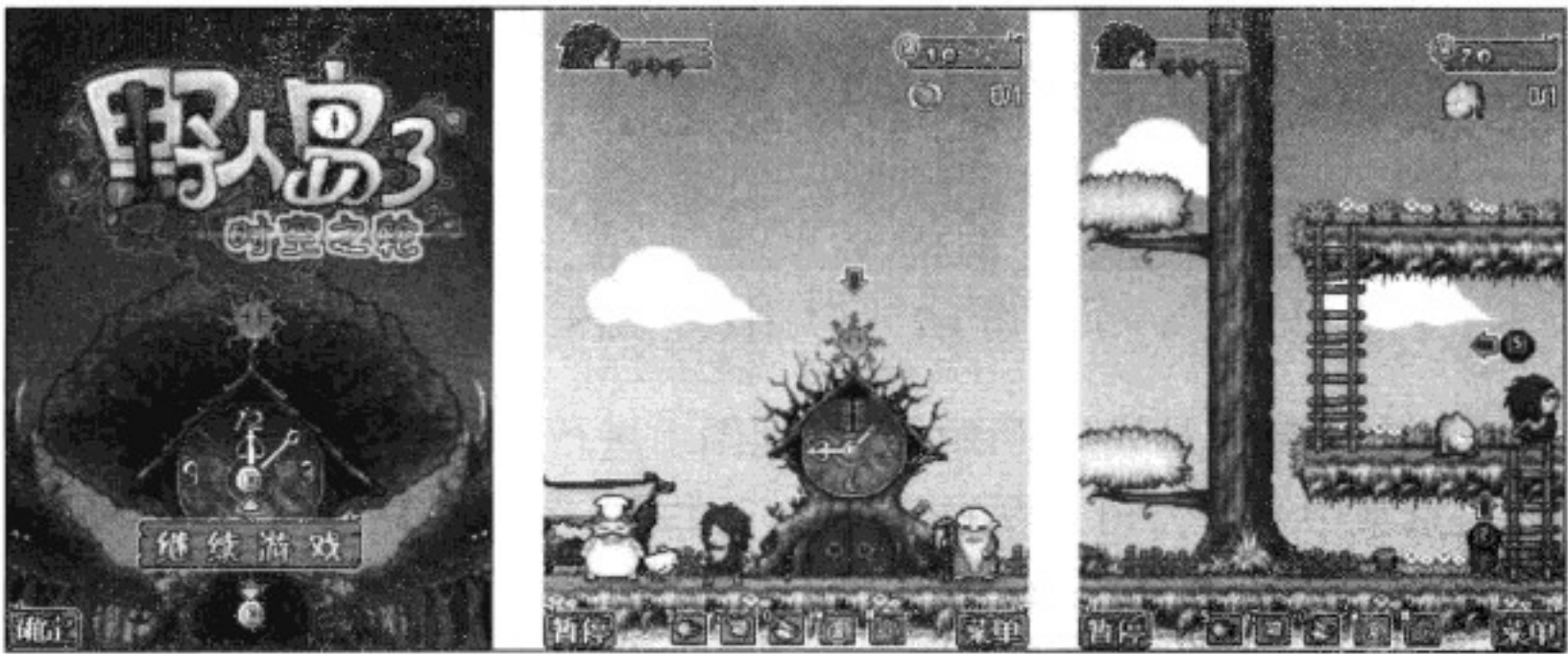


图 1-5 《野人岛》截图

有些 AVG 类游戏采用 RPG 类游戏的风格，带领玩家去逐次破解其中的谜团，这种游戏一般都在剧情上加大投入，使玩家沉醉在曲折的剧情当中，例如北京捷通华声公司开发的一款根据电影改编的《古宅魅影》游戏就属于此类型，如图 1-6 所示。

④ PUZ（益智类）游戏

PUZ（Puzzle Game）类游戏不同于其他类型的游戏，它没有选材的局限和核心的游戏系统，一般比较轻松、可爱，而且简单易玩。例如，从手机游戏开始兴起时就风靡一时的《贪吃蛇》、《俄罗斯方块》等经典的益智类游戏，这类游戏需要的美术造诣极低，而且算法简单，玩家操作极易

上手；同时游戏可重复挑战，对玩家有很强的吸引力。PUZ 类游戏的代表为北京掌趣公司根据 PC 大富翁改编的一款益智休闲类游戏《大富翁-奋斗》，如图 1-7 所示。

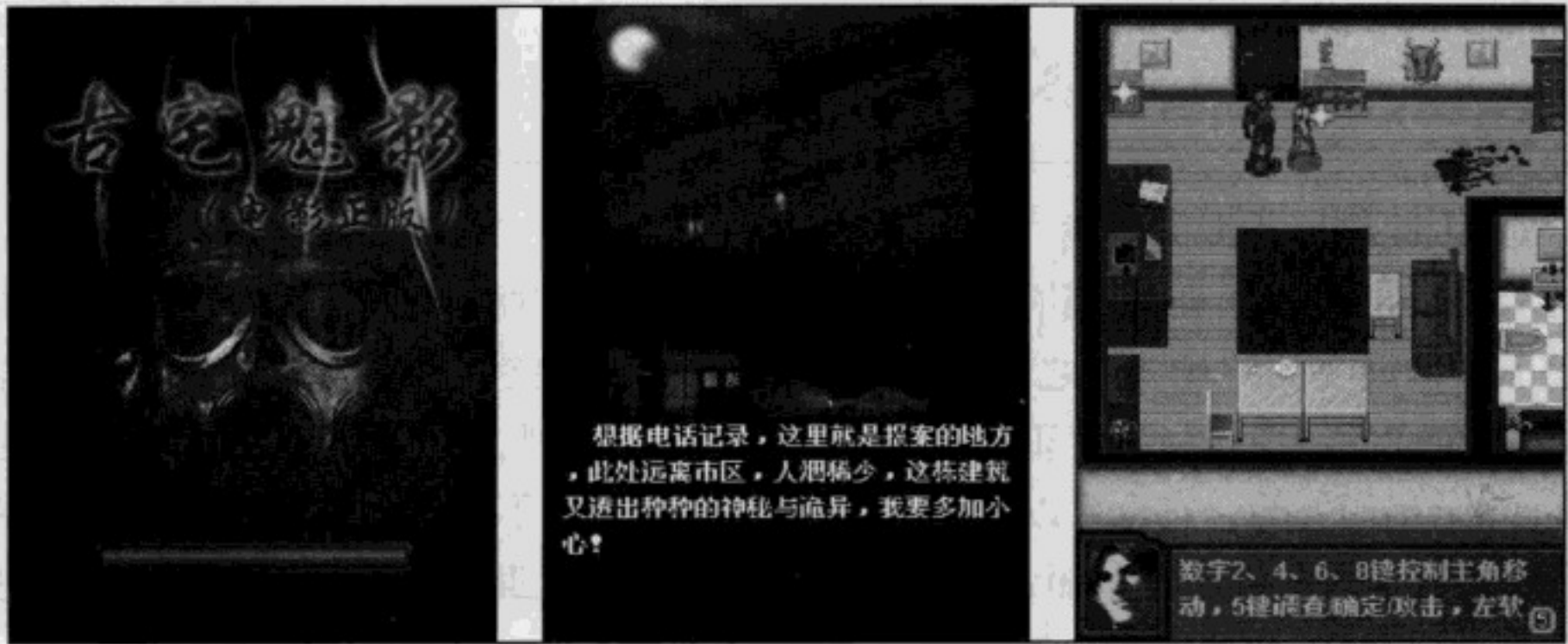


图 1-6 《古宅魅影》截图

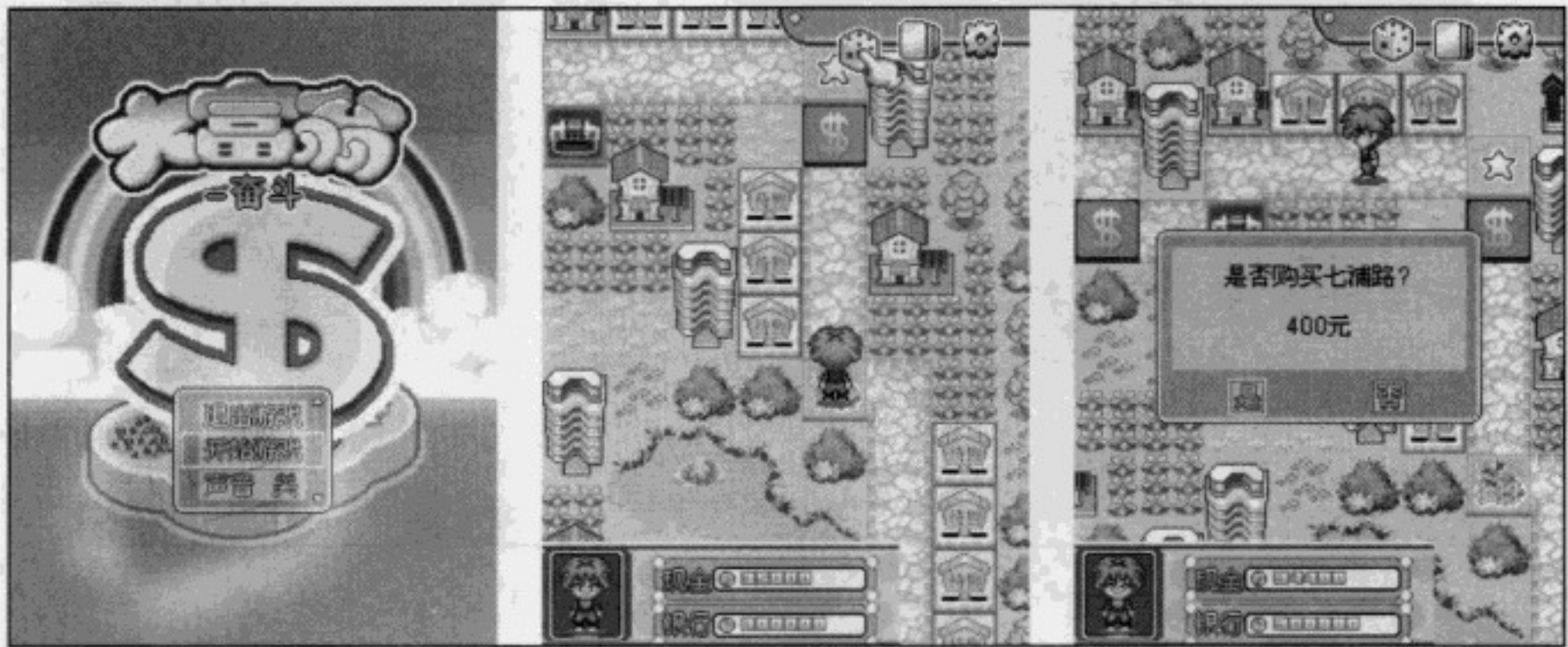


图 1-7 《大富翁-奋斗》截图

⑤ STG（射击类）游戏

STG（Shooting Game）类游戏主要是指依靠远程武器与敌人进行对抗的游戏，一般玩家所说的是指飞行射击类的游戏。PC 上著名的《雷电》就属于此类型，《雷电 II》就是根据《雷电》更改而成的手机飞行射击类游戏，其拥有绚丽的射击画面，如图 1-8 所示。

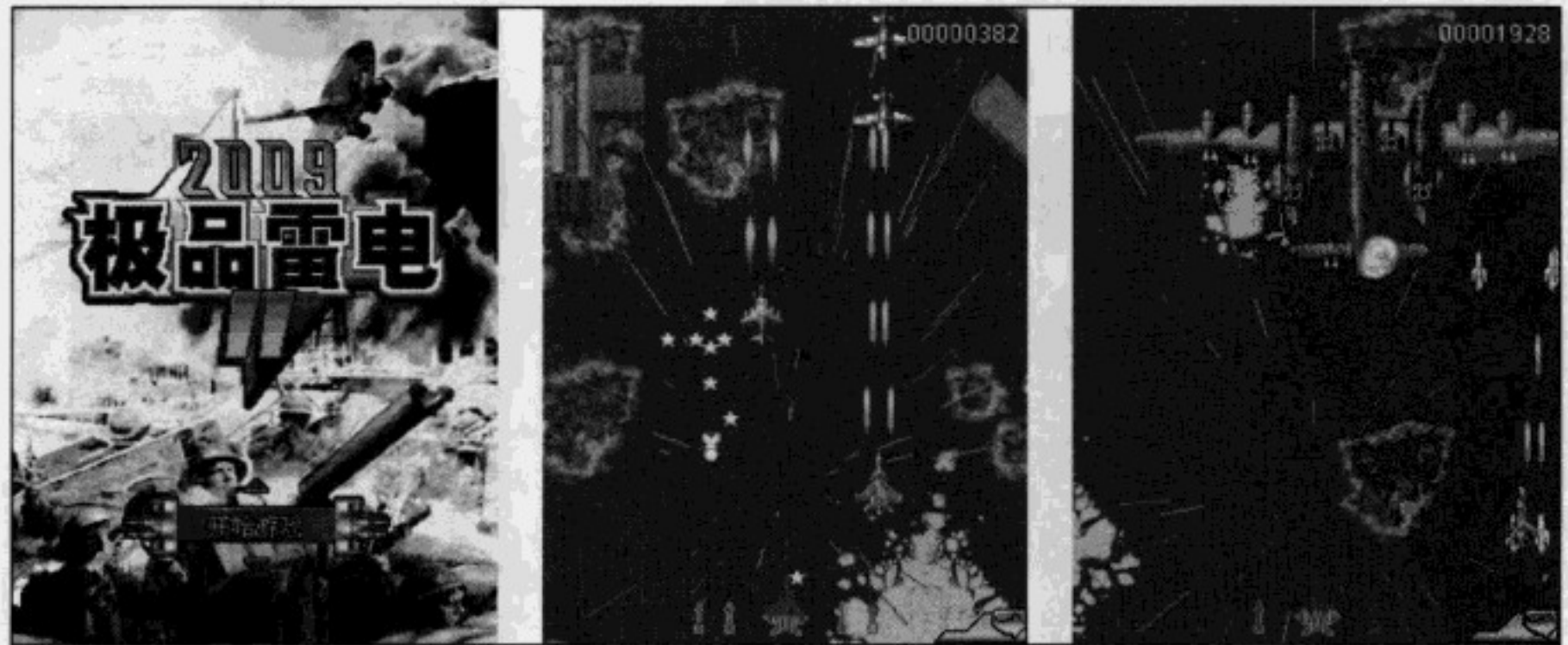


图 1-8 《雷电 II》截图

STG 类游戏还包括枪支光线射击类游戏，例如 PC 上的《VR 战警》，玩家通过控制枪支的光线进行射击。



射击类游戏主要考虑物体碰撞的问题。

⑥ ACT（动作过关类）游戏

ACT（Action Game）类游戏讲究的是打斗的快感以及绚丽的画面体验。目前，在国内，ACT 类游戏是下载量最大的游戏类型之一，其打斗的场景以及快速的节奏深受手机玩家的喜爱。

ACT 类游戏的经典作品有育碧公司的《细胞分裂》系列、《波斯王子》系列以及胜天堂公司的《刺客-六国相印》等，它也可以和 RPG 类游戏一样加入不同的剧情和关卡，丰富游戏的趣味性，但是一般其通过时间都比较短。北京华娱无线公司根据四大名著之一《西游记》制作的手机游戏《真西游记》，就增加了不少趣味性，如图 1-9 所示。



图 1-9 《真西游记》截图

ACT 类游戏的重点在于动作与打斗的体验上，玩家操作主角进行厮杀、过关以及历险等，此类游戏一般考验的是玩家的反应速度，著名的 PC 游戏《魂斗罗》就属于该类型。

⑦ RAC（赛车类）游戏

RAC（Race Game）类游戏使玩家体验赛车时所产生的速度快感，著名的 PC 游戏《极品飞车》就属于该类型。但对于手机平台有限的性能，制作一款很好的赛车类游戏是相当不易的，而且国内此类游戏的下载量也不是很大，一般在欧美比较流行。GameLoft 公司的《街头赛车 3D》可以说是一款不错的赛车类游戏，如图 1-10 所示。

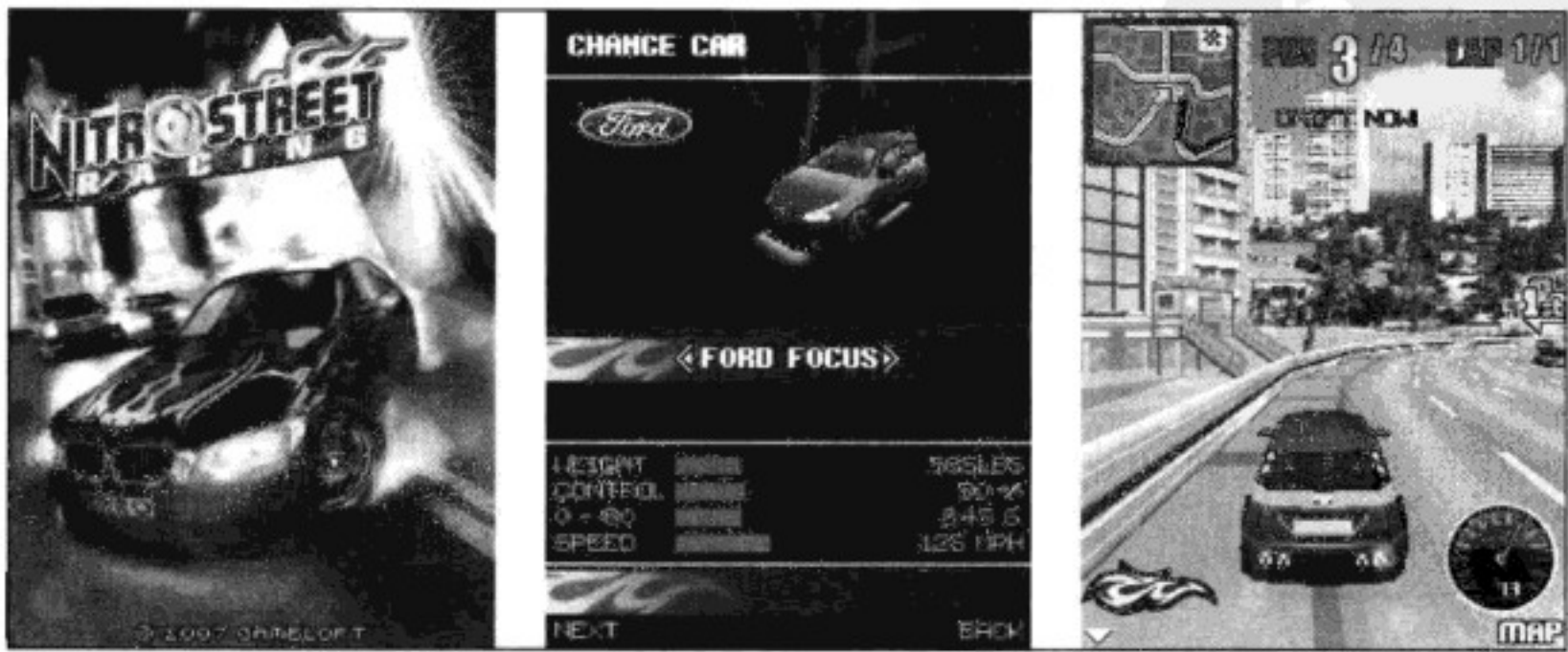


图 1-10 《街头赛车 3D》截图

目前体育游戏由于比较耐玩，发展得很快，但是这些体育游戏不够真实，因为在图形表现、规则使用以及体育运动本身的多样性方面，与实际的体育运动无法相比。但是现在可以幸运地发现，很多体育游戏提供了人工智能还不错的运动游戏，但是粗糙的动画效果和缓慢而僵硬的人物还是很难使人兴奋。随着设备性能的提高，各种类型的体育游戏，例如保龄球、网球、篮球等都有了很大的提高，可玩性也大幅度提升。

同时从统计数据来看，传统益智游戏占据了绝大多数。传统益智游戏是年轻一族十分热衷的娱乐项目，耐玩性也很强，是游戏开发的一个重点。

1.3 电脑游戏的策划

1.3.1 游戏策划的基本概念

策划主要担任游戏的整体规划工作（包含游戏运行的所有细节和开发进度等），如同建筑工程中施工前要有建筑蓝图一样，策划的工作就是用程序和美工能够理解的方式，撰写游戏设计文档，对游戏的整体模式进行叙述。游戏中的所有部分都在策划的工作范围之内。因此，做策划的人员最好对程序开发和美术设计的工作有所了解，这是做游戏策划所要具备的基本素质之一。

美术设计人员需要制作各种美术素材，使游戏里的各种东西得以呈现在我们面前，而程序开发人员则要把策划人员设计的游戏规则用各种代码加以实现，使美术设计人员制作的素材按照策划人员制订的游戏规则在游戏里互动。

游戏策划具体分为以下几种。

1. 关卡策划

职责：需要掌握绘图工具来设计关卡。配合数值策划设定数值、配合剧情策划进行设计，甚至系统设计等方面。需要跟进程序，进行任务系统方面的实现；提出任务编辑器、场景编辑器等方面的需求；提交美术资源的需求；跟进美术资源的制作等。此外，还要架构整个任务系统，进行场景架构以及编写任务等。

2. 数值策划

职责：进行数值的平衡和制定，游戏中各种公式的设计，以及整个经济系统的搭建、整个战斗系统的设计等。根据公司和项目不同，可能还包括同战斗系统和数值密切相关的，例如职业系统、技能系统、装备系统、精炼打造系统等系统的设计。此外，需要关卡策划的辅助进行怪物数值的制定，需要系统策划配合进行系统中各种数值的设定等。

3. 剧情策划

职责：主要负责游戏的背景、世界观、剧情的扩展，任务的设计，任务对白的撰写等方面。另外，在剧情设计中，剧情策划还需要同关卡策划紧密配合。因为关卡策划在架构世界的时候，就是依托于剧情策划设计的世界观和背景的；而剧情策划又会根据关卡策划设计的世界，设计相关的剧情。

1.3.2 游戏设计的基本内容

1. 游戏的类型

设计者首先需要确定游戏类型。这是一款第一人称射击游戏？还是继承《马里奥》灵感的 3D

平台游戏？或者是基于《万智牌》的纸牌游戏？设计者需要考虑的是要创造一款什么样的游戏，它与其他游戏类型有什么不同，它的游戏玩法是怎样的，外观和游戏感觉又是怎样的——是黑暗的、现实的、还是模拟现实生活或突出幻想世界等。

2. 市场定位

尽管这主要是市场营销团队的工作，但是对于所有致力于游戏开发的人来说这点非常重要，因为只有明确了市场定位，他们才能掌握自己创造的游戏是面向哪些用户。如果游戏是面向儿童，那么游戏中就需要尽可能地减少暴力和性元素。如果游戏是针对男性或女性玩家，那么就需要设计者在创造任何游戏元素（例如游戏帮助：包括游戏玩法、游戏图像以及音乐）时始终考虑对象的爱好特点。

除此之外设计者还将在这里提及游戏所面向的平台，以及为何这一平台适合他们的游戏，列出同一类型中最成功的游戏、游戏的潜在竞争对手、如何才能取得竞争优势，并根据市场研究做出游戏潜在销量的推断。

3. 游戏角色

设计者需要确定游戏中的所有角色，从主角开始，必须包括角色的外观、他们的年龄、体重、个性、背景等。除了主角，还不能漏掉所有的非玩家角色，包括玩家将在游戏中遭遇到的怪物、好友和敌人等。

4. 故事情节

在这一部分的设计中，设计者需要设计游戏故事情节，即以线性结构模式告诉玩家他们将在游戏中经历什么。其中还包含故事的阐述模式（是否有文本、画外音、过场动画或者所有这些方法相结合），以及背景故事或次要情节（即未依附主要故事情节但是却伴随着它而发展的内容）的细节等。

5. 游戏玩法

游戏玩法是设计中最重要的一大环节。设计者需要基于游戏的不同部分完整地绘制出游戏控制，明确玩家在小规模和整体规模下的成功和失败标准、具体 AI 行为模式、武器或升级能力、菜单或任何隐藏目标等内容。总之，这一部分必须详细明确玩家所控制的角色所经历的所有内容以及受 AI 控制的非玩家角色的反应。

1.4 电脑游戏的程序开发工具

当前市面上的游戏种类很多，游戏程序设计工具也五花八门，现将其分门别类进行说明。

1. C/C++程序设计语言

大中型游戏大多使用 C/C++作为程序设计语言开发。C/C++是所有程序设计人员公认的功能强大的程序设计语言，也是运行时速度比较快的语言。比较著名的 C/C++语言开发工具是微软公司的 Visual C++.NET 产品，如图 1-11 所示。

2. Java 程序设计语言

Java 程序设计语言具有跨平台的优点，程序的移植性好，因此 Java 语言非常适合进行游戏制作。对于大型网络游戏来说，使用 Java 语言设计则不具备速度优势。比较著名的 Java 语言的开发工具有 IBM 公司的 Eclipse，如图 1-12 所示，以及 Sun 公司的 NetBeans 等。

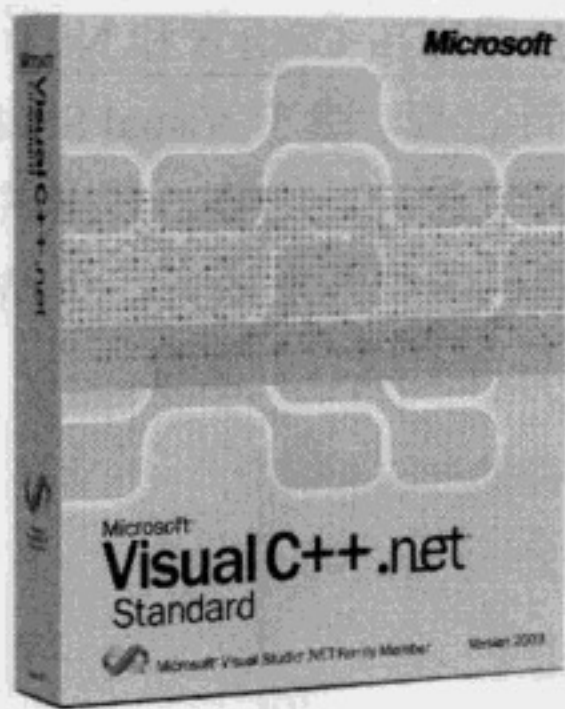


图 1-11 Visual C++.net

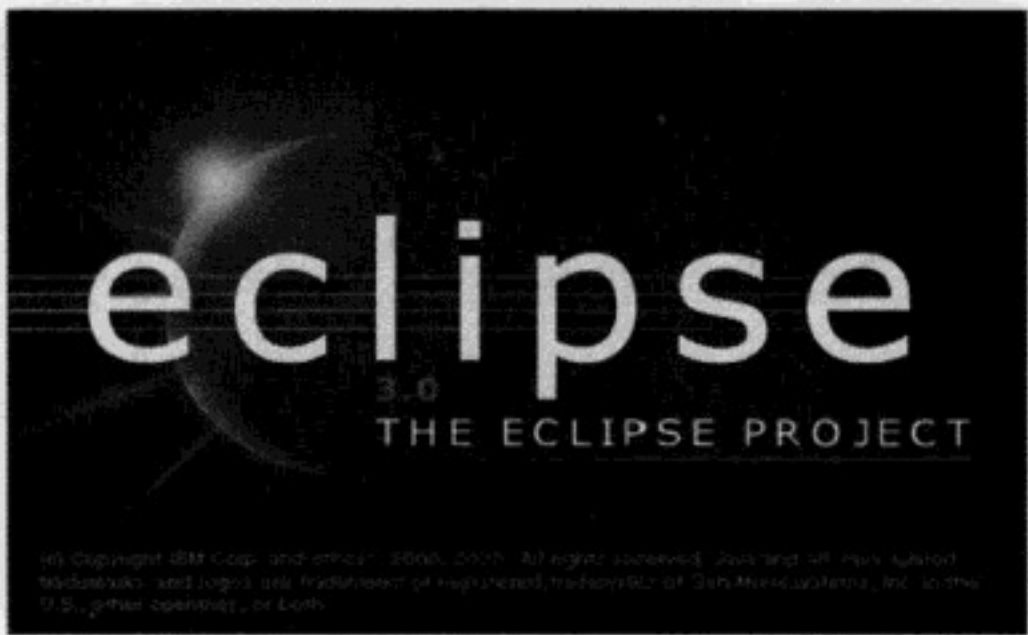


图 1-12 Eclipse

Java 程序设计语言的优势领域包括：基于 Applet 的网页游戏和手机游戏。

在目前的手机游戏市场中，平台包括 Java 平台、Symbian（塞班）平台（只有少部分手机品牌支持，如诺基亚、索爱、三星等）、iOS 平台（苹果手机平台）、Android（安卓）平台和 Windows Mobile 平台。但是在智能手机市场中，iOS 平台受制于终端的数量增长缓慢，另外一个 Android 平台则处在爆发式的增长期，主要源于 Android 平台终端的普及和性能提升，还有联运平台和运营商的发力。基于 Android 平台开发的游戏采用的是 Java 语言，许多 PC 上开发的 Java 游戏可以移植到 Android 智能手机平台上。

3. Flash ActionScript

FlashActionScript 属于一种脚本语言，通常嵌入在 Flash 文件中，负责对 Flash 动画流程进行控制。使用 FlashActionScript 语言设计出来的游戏画面精美，容量也较小，因此在小游戏的设计领域迅速走红。2D 平面游戏都可以使用 Flash 编写，也可适当地规划制作出闯关游戏和平面 RPG 游戏。比较好的 Flash 游戏设计的工具是 Adobe 公司的 Flash CS 系列产品，如图 1-13 所示。

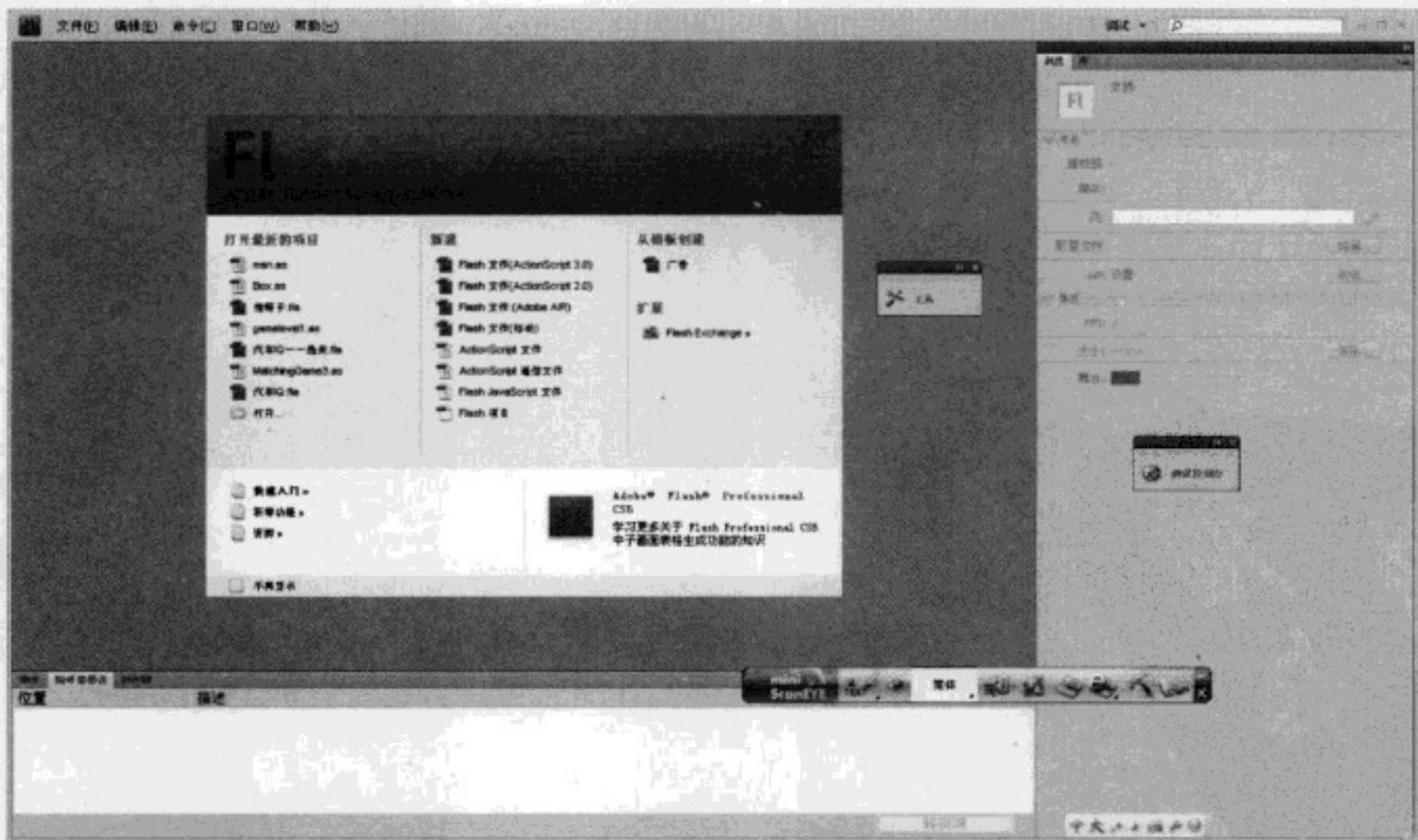


图 1-13 FlashCS4

4. C#程序设计语言

C#（读作“C sharp”）是由 C++和 Java 发展演化而来的程序语言，同时具备了两种语言的

优点，既支持面向对象程序设计，又具有很高的运行速度。微软公司还在 C#语言和.NET 框架基础上推出了一个专用的游戏开发工具 XNA，极大地简化了游戏设计过程。基于 Visual Studio.NET (C#开发环境，如图 1-14 所示) 的 XNA 游戏开发平台具备快速开发游戏的优势，其提供两种开发模式：基于 Windows 和基于 Xbox。C#作为微软公司当前最重要的编程开发语言，在 Windows 平台上得到了很好的维护和支持，但是微软公司并没有提供其他平台上的官方支持。



图 1-14 Visual Studio.NET

第 2 章

游戏图形界面开发基础

对于一款游戏软件来说，不但要有完善的功能，而且还要有一个简洁美观的界面。本章主要学习如何进行图形界面编程，其中包括 AWT 和 Swing 两部分内容。

2.1 AWT 简介

AWT (Abstract Window Toolkit, 抽象窗口工具集) 是一个特殊的组件，其中包含其他的组件。它的库类也非常丰富，包括创建 Java 图形界面程序的所有工具。用户可以利用 AWT，在容器中创建标签、按钮、复选框、文本框等用户界面元素。

AWT 中包括图形界面编程的基本类库，是 Java 语言 GUI 程序设计的核心，它为用户提供基本的界面构件。这些构件是为了使用户和计算机之间能够更好地进行交互，而用来建立图形用户界面的独立平台，其中主要由以下几部分组成。

组件类 (Component)、容器类 (Container)、图形类 (Graphics) 和布局管理器 (LayoutManager)。Component (组件) ——按钮、标签、菜单等组件的抽象基本类。

Container (容器) ——扩展组件的抽象基本类。例如 Panel、Applet、Window、Dialog 和 Frame 等是由 Container 演变的类，容器中可以包括多个组件。

LayoutManager (布局管理器) ——定义容器中组件摆放位置和大小的接口。Java 中定义了几种默认的布局管理器。

Graphics (图形类) ——组件内与图形处理相关的类，每个组件都包含一个图形类的对象。

在 AWT 中存在缺少剪贴板、打印支持等缺陷，甚至没有弹出式菜单和滚动窗口等，因此 Swing 的产生也就成为必然。Swing 是纯 Java 语言实现的轻量级 (light-weight) 组件，它不依赖系统的支持。本章主要讨论 Swing 组件基本的使用方法和使用 Swing 组件创建用户界面的初步方法。

2.2 Swing 基础

Swing 是 Sun 公司推出的第二代图形用户接口工具包，通过 Swing 可以开发出功能强大、界面优美的客户应用程序。Swing 中不但提供了很多功能完善的组件，而且还具有良好的扩展能力，用 Swing 进行交互界面的开发是一项令开发人员非常愉快的工作。

Swing 元素的屏幕显示性能要比 AWT 要好，而且 Swing 是使用纯 Java 语言来实现的，因此

Swing 也理所当然地具有 Java 的跨平台性。但 Swing 并不是真正使用原生平台来提供设备，而是仅仅在模仿。因此，可以在任何平台上使用 Swing 图形用户界面组件。Swing 被称为“轻量级 (light-weight)”组件，AWT 被称为“重量级 (heavy-weight)”组件。“重量级”组件与“轻量级”组件一起使用时，如果组件区域有重叠，则“重量级”组件总是显示在上面。

虽然 AWT 是 Swing 的基础，但是 Swing 中却提供了比 AWT 更多的图形界面组件，而且 Swing 中组件的类名都是由字母“J”开头，还增加了一些比较复杂的高级组件，例如 JTable、JTree 等。

2.3 Swing 组件

Swing 的组件与 AWT 组件相似，但又为每一个组件增添了新的方法，并提供了更多的高级组件。本节 Swing 的基本组件选取几个比较典型的组件进行详细讲解，本节没有讨论到的组件，读者在使用中如果遇到困难，可以参考 API 文档。

与 AWT 组件不同，Swing 组件不能直接添加到顶层容器中，它必须添加到一个与 Swing 顶层容器相关联的内容面板 (content pane) 容器上。内容面板是顶层容器包含的一个普通容器，它是一个轻量级组件。

2.3.1 按钮 (Jbutton)

Swing 中的按钮是 JButton，它是 javax.swing.AbstractButton 类的子类，Swing 中的按钮可以显示图像，并且可以将按钮设置为窗口的默认图标，而且还可以将多个图像指定给一个按钮。

在 JButton 中有如下几个比较常用的构造方法。

- JButton(Icon icon): 按钮上显示图标。
- JButton(String text): 按钮上显示字符。
- JButton(String text, Icon icon): 按钮上既显示图标又显示字符。

JButton 类的方法：

- setText(String text): 设置按钮的标签文本。
- setIcon(Icon defaultIcon): 设置按钮在默认状态下显示的图片。
- setRolloverIcon(Icon rolloverIcon): 设置当光标移动到按钮上方时显示的图片。
- setPressedIcon(Icon pressedIcon): 设置当按钮被按下时显示的图片。
- setContentAreaFilled(boolean b) : 设置按钮的背景为透明，当设为 false 时表示不绘制，默认为绘制。
- setBorderPainted(boolean b) : 设置为不绘制按钮的边框，当设为 false 时表示不绘制，默认为绘制。

按钮组件是 GUI 中最常用到的一种组件。按钮组件可以捕捉到用户的单击事件，同时利用按钮事件处理机制响应用户的请求。JButton 类是 Swing 提供的按钮组件，在单击 JButton 类对象创建的按钮时，会产生一个 ActionEvent 事件。

2.3.2 单选按钮 (JRadioButton)

JRadioButton 组件实现的是一个单选按钮。JRadioButton 类可以单独使用，也可以与 ButtonGroup 类联合使用，当单独使用时，该单选按钮可以被选定和取消选定；当与 ButtonGroup

类联合使用时,需要使用 add()方法将 JRadioButton 添加到 ButtonGroup 中,并组成一个单选按钮组。此时用户只能选定按钮组中的一个单选按钮。



使用 ButtonGroup 对象进行分组是逻辑分组而不是物理分组。创建一组按钮通常需
要创建一个 JPanel 或者类似容器,并将按钮添加到容器中。

JRadioButton (单选按钮) 组件的常用方法:

- setText(String text): 设置单选按钮的标签文本。
- setSelected(boolean b): 设置单选按钮的状态,默认情况下未被选中,当设为 true 时表示单选按钮被选中。
- add(AbstractButton b): 添加按钮到按钮组中。
- remove(AbstractButton b): 从按钮组中移除按钮。
- getButtonCount(): 返回按钮组中包含按钮的个数,返回值为 int 型。
- getElements(): 返回一个 Enumeration 类型的对象,通过该对象可以遍历按钮组中包含的所有按钮对象。
- isSelected(): 返回单选按钮的状态,当设为 true 时为选中。
- setSelected(boolean b): 设定单选按钮的状态。

【例 2-1】 示例功能是选择用户所喜欢的城市。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JRadioButtonTest {
    JFrame f = null;
    JRadioButtonTest() {
        f = new JFrame("单选框示例"); //创建一个 JFrame 的对象
        Container contentPane = f.getContentPane(); //创建一个内容面板容器
        contentPane.setLayout(new FlowLayout()); //设置该窗口的布局
        JPanel p1 = new JPanel(); //创建一个面板对象 p1
        p1.setLayout(new GridLayout(1, 3)); //设置布局管理器的格式
        p1.setBorder(BorderFactory.createTitledBorder("选择你喜欢的城市"));
        //定义 3 个 JRadioButton 单选按钮
        JRadioButton r1 = new JRadioButton("北京");
        JRadioButton r2 = new JRadioButton("上海");
        JRadioButton r3 = new JRadioButton("青岛");
        p1.add(r1);
        p1.add(r2);
        p1.add(r3);
        r1.setSelected(true); //设置“北京”单选按钮的状态为被选中
        contentPane.add(p1); //面板对象 p1 加到窗口内容面板容器中
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter() { //添加一个窗口监听器
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}
```



```
    }  
    public static void main(String args[]) {  
        new JRadioButtonTest();  
    }  
}
```

程序运行结果如图 2-1 所示。程序首先创建 JFrame 窗口对象 f、内容面板 (contentPane)，并设置窗口的布局格式为流布局 FlowLayout(); 之后定义 3 个 JRadioButton 对象，并设置各自的显示文本同时添加到面板对象 p1 中；然后为窗口设置事件监听；最后创建主方法，并在主方法中调用构造方法 JRadioButtonTest()。

2.3.3 复选框 (JCheckBox)

使用复选框可以完成多项选择。Swing 中的复选框与 AWT 中的复选框相比，优点是 Swing 复选框中可以添加图片。复选框可以为每一次的单击操作添加一个事件。

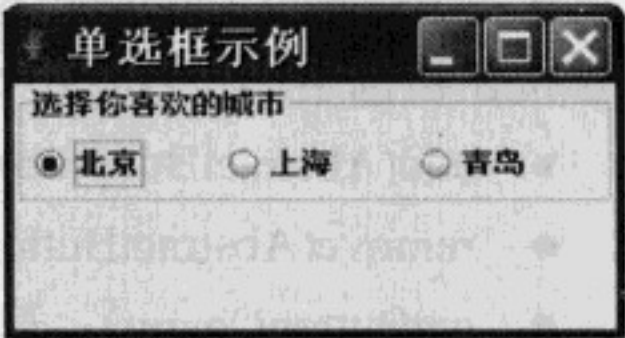


图 2-1 例 2-1 的运行结果

复选框的构造方法如下。

- JCheckBox(Icon icon): 创建一个有图标，但未被选中的复选框。
- JCheckBox(Icon icon, boolean selected): 创建一个有图标复选框，并且指定是否被选中。
- JCheckBox(String text): 创建一个有文本，但未被选中的复选框。
- JCheckBox(String text, boolean selected): 创建一个有文本复选框，并且指定是否被选中。
- JCheckBox(String text, Icon icon): 创建一个指定文本和图标，但未被选中的复选框。
- JCheckBox(String text, Icon icon, boolean selected): 创建一个指定文本和图标，并且指定是否被选中的复选框。

是否被选中的复选框。

常用方法：

- public boolean isSelected(): 返回复选框状态，true 时为选中。
- public void setSelected(boolean b): 设定复选框状态。

【例 2-2】 设计一个继承面板的 Favorite 类，类别有：运动、电脑、音乐、读书，界面如图 2-2 所示。

```
import javax.swing.*;  
import java.awt.*;  
class Favorite extends JPanel{  
    JCheckBox sport,computer,music,read;  
    Favorite(){  
        sport = new JCheckBox("运动");  
        computer = new JCheckBox("电脑");  
        music = new JCheckBox("音乐");  
        read = new JCheckBox("读书");  
        add(new JLabel("爱好"));  
        add(sport);add(computer);add(music);add(read);  
        sport.setSelected(false);  
        computer.setSelected(false);  
        music.setSelected(false);  
        read.setSelected(false);  
    }  
}
```



图 2-2 例 2-2 的界面

下面是显示 Favorite 面板对象的窗口。

```
import javax.swing.*;
```



```

import java.awt.*;
public class JCheckBoxExample extends JFrame{
    public JCheckBoxExample (){
        super("复选框");
        Container container=getContentPane();
        container.setLayout(new FlowLayout());
        Favorite f=new Favorite();
        container.add(f);
        pack();
        setVisible(true);
    }
    public static void main(String args[]){
        JCheckBoxExample jcbe=new JCheckBoxExample ();
        jcbe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

2.3.4 组合框（JComboBox）

JComboBox 组件用来创建组合框对象。通常，根据组合框是否可编辑的状态，可以将组合框分成两种常见的外观。可编辑状态外观可视为文本框和下拉列表的组合，不可编辑状态的外观可视为按钮和下拉列表的组合。在按钮或文本框的右边有一个带三角符号的下拉按钮，用户可以单击该下拉按钮，便可出现一个内容列表，这也是组合框的得名。组合框通常用于从列表的“多个项目中选择一个”的操作。

JComboBox 的构造方法有如下几种：

- JComboBox(): 创建一个默认模型的组合框。
- JComboBox(ComboBoxModel aModel): 创建一个指定模型的组合框。
- JComboBox(Object[] items): 创建一个具有数组定义列表内容的组合框。

【例 2-3】 利用 JComboBox 设计一个选择城市的程序，界面如图 2-3 所示。

```

import javax.swing.*;
import java.awt.*;
public class JComboBoxExample extends JFrame{
    JComboBox comboBox1,comboBox2;
    String cityNames[]={"北京","上海","重庆","南京","武汉","杭州"};
    public JComboBoxExample(){
        super("组合框");
        Container container=getContentPane();
        container.setLayout(new FlowLayout());
        comboBox1=new JComboBox(cityNames);
        comboBox1.setSelectedIndex(3);
        comboBox1.setEditable(false);
        comboBox2=new JComboBox(cityNames);
        comboBox2.setSelectedItem(cityNames[1]);
        comboBox2.addItem(new String("长沙"));
        comboBox2.setEditable(true);
        container.add(comboBox1);
        container.add(comboBox2);
        pack();
        setVisible(true);
    }
    public static void main(String args[]){

```



```
JComboBoxExample jcbe=new JComboBoxExample();
jcbe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

2.3.5 列表（JList）



图 2-3 例 2-3 的界面

JList 组件用于定义列表，允许用户选择一个或多个项目。与 JTextArea 类似，JList 本身不支持滚动功能，如果要显示超出显示范围的项目，可以将 JList 对象放置到滚动窗格 JScrollPane 对象中，便可以为列表对象实现滚动操作。

JList 的构造方法如下：

- JList(): 创建一个空模型的列表。
- JList(ListModel dataModel): 创建一个指定模型的列表。
- JList(Object[] listdatas): 创建一个具有数组指定项目内容的列表。

常用方法如下：

- int getFirstVisibleIndex(): 获取第一个可见单元的索引。
- void setFirstVisibleIndex(int): 设置第一个可见单元的索引。
- int getLastVisibleIndex(): 获取最后一个可见单元的索引。
- void setLastVisibleIndex(int): 设置最后一个可见单元的索引。
- int getSelectedIndex(): 获取第一个已选的索引。
- void setSelectedIndex(int): 设置第一个已选的索引。
- Object getSelectedValue(): 获取第一个已选的对象。
- void setSelectedValue(Object): 设置第一个已选的对象。
- Object[] getSelectedValues(): 获取已选的所有对象。
- Color getSelectionBackground(): 获取选中项目的背景色。
- void setSelectionBackground(): 设置选中项目的背景色。
- Color getSelectionForeground(): 获取选中项目的前景色。
- void setSelectionForeground(): 设置选中项目的前景色。

2.3.6 文本框（JTextField 和 JPasswordField）

JTextField 组件用于创建文本框。文本框是用来接收用户的单行文本信息输入的区域。通常文本框用于接收用户信息或其他文本信息的输入。当用户输入文本信息后，如果为 JTextField 对象添加了事件处理，按回车键后就会触发一定的操作。

JPasswordField 是 JTextField 的子类，是一种特殊的文本框，也是用来接收单行文本信息输入的区域，但是会用回显字符串代替输入的文本信息。因此，JPasswordField 组件也称为密码文本框。JPasswordField 默认的回显字符是“*”，用户可以自行设置回显字符。

JTextField 的常见构造方法有如下几种：

- JTextField(): 创建一个空文本框。
- JTextField(String text): 创建一个具有初始文本信息 text 的文本框。
- JTextField(String text,int columns): 创建一个具有初始文本信息 text 以及指定列数的文本框。

JTextField 的常用方法：

- void setText(String): 设置显示内容。

- String getText(): 获取显示内容。

JPasswordField 的构造方法有如下几种:

- JPasswordField(): 创建一个空的密码文本框。
- JPasswordField(String text): 创建一个指定初始文本信息的密码文本框。
- JPasswordField(String text,int columns): 创建一个指定文本和列数的密码文本框。
- JPasswordField(int columns): 创建一个指定列数的密码文本框。

JPasswordField 是 JTextField 的子类, 因此 JPasswordField 也具有与 JTextField 类似的名称和功能的方法, 此外, 它还具有自己的独特方法:

- boolean echoCharIsSet(): 获取设置回显字符的状态。
- void setEchoChar(char): 设置回显字符。
- char getEchoChar(): 获取回显字符。
- char[] getPassword(): 获取组件的文本。

2.3.7 面板 (JPanel)

JPanel 组件定义的面板实际上是一种容器组件, 用来容纳各种其他轻量级的组件。此外, 用户还可以用这种面板容器绘制图形。

JPanel 的构造方法如下:

- JPanel(): 创建具有双缓冲和流布局 (FlowLayout) 的面板。
- JPanel(LayoutManager layout): 创建具有指定布局管理器的面板。

JPanel 的常用方法:

- void add(Component): 添加组件。
- void add(Component,int): 添加组件至索引指定位置。
- void add(Component,Object): 按照指定布局限制添加组件。
- void add(Component,Object,int): 按照指定布局管理器限制添加组件至指定位置。
- void remove(Component): 移除组件。
- void remove(int): 移除指定位置的组件。
- void removeAll(): 移除所有组件。
- void paintComponent(Graphics): 绘制组件。
- void repaint(): 重新绘制。
- void setPreferredSize(Dimension): 设置最佳尺寸。
- Dimension getPreferredSize(): 获取最佳尺寸。

【例 2-4】 利用 JPanel 设计一个程序, 界面如图 2-4 所示。

```
import javax.swing.*;
import java.awt.*;

public class JPanelExample extends JFrame{
    JButton[] buttons;
    JPanel panel1;
    CustomPanel panel2;
    public JPanelExample(){
        super("面板示例");
        Container container=getContentPane();
        container.setLayout(new BorderLayout());
```



```
panel1=new JPanel(new FlowLayout()); //创建一个流布局管理器的面板
buttons=new JButton[4];
for(int i=0;i<buttons.length;i++){
    buttons[i]=new JButton("按钮 "+(i+1));
    panel1.add(buttons[i]); //添加按钮到面板 panel1 中
}
panel2=new CustomPanel();
container.add(panel1, BorderLayout.NORTH);
container.add(panel2, BorderLayout.CENTER);
pack();
setVisible(true);
}
public static void main(String args[]){
    JPanelExample jpe=new JPanelExample();
    jpe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
class CustomPanel extends JPanel{ //定义内部类 CustomPanel
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawString("Welcome to Java Shape World",20,20);
        g.drawRect(20,40,130,130);
        g.setColor(Color.green); //设置颜色为绿色
        g.fillRect(20,40,130,130); //绘制矩形
        g.drawOval(160,40,100,100); //绘制椭圆
        g.setColor(Color.orange); //设置颜色为橙色
        g.fillOval(160,40,100,100); //绘制椭圆
    }
    public Dimension getPreferredSize(){ //设置最佳尺寸
        return new Dimension(200,200);
    }
} //结束内部类的定义
```

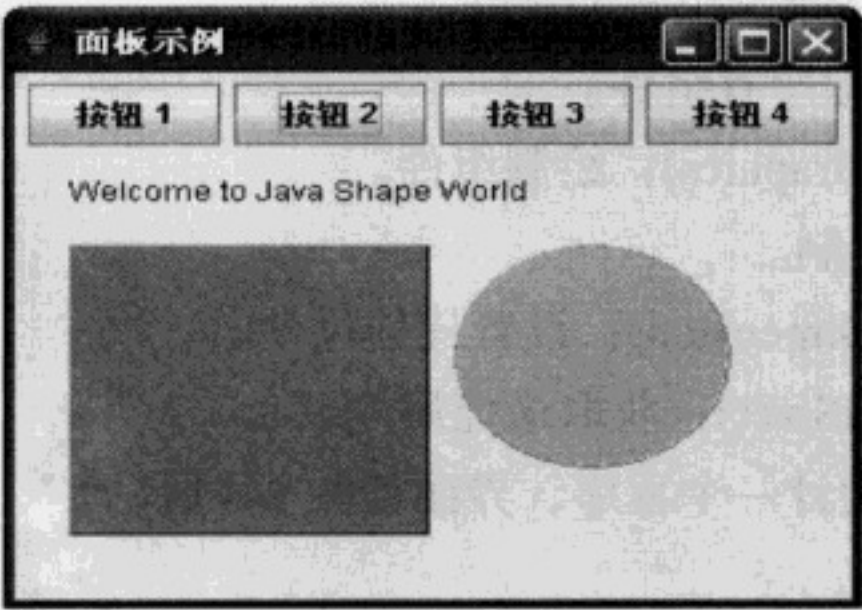


图 2-4 例 2-4 的界面

2.3.8 表格 (JTable)

表格是 Swing 新增加的组件，主要功能是把数据以二维表格的形式显示出来。使用表格，依据 M-V-C 的思想，最好先生成一个 MyTableModel 类型的对象来表示数据，这个类是从 AbstractTableModel 类中继承来的，其中有几个方法一定要重写，例如 getColumnCount、

getRowCount、getColumnName 和 getValueAt。因为 JTable 会从这个对象中自动获取表格显示所必需的数据，AbstractTableModel 类的对象负责表格大小的确定（行、列）、内容的填写、赋值、表格单元更新的检测等一切与表格内容有关的属性及其操作。JTable 类生成的对象以该 TableModel 为参数，并负责将 TableModel 对象中的数据以表格的形式显示出来。

2.3.9 框架（JFrame）

框架是 Swing GUI 应用程序的主窗口，窗口包括边界、标题、关闭按钮等。

JFrame 类是 java.awt 包中 Frame 类的子类，其子类创建的对象是窗体，对象（窗体）是重量容器。不能把组件直接添加到 Swing 窗体中，其含有内容面板容器，应该把组件添加到内容面板中；不能为 Swing 窗体设置布局，而应当为 Swing 窗体的内容面板设置布局。

Swing 窗体通过 getContentPane()方法获得 JFrame 的内容面板，再对其加入组件：

```
JFrame frame=new JFrame();
Container ct= frame.getContentPane();//获取内容面板容器
ct.add(childComponent);          //将内容面板容器加入组件
```

框架（JFrame）常用的方法和事件：

- frame.setVisible(true): 显示框架对象代表的框架窗口。
- frame.setSize(200,100)或者 frame.pack(): 设置框架的初始显示大小。
- frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE): 当用户单击框架的关闭按钮则退出程序，或者添加 WindowListener 监听器实现关闭按钮退出程序。

【例 2-5】 基于 JFrame 实现的窗口界面，窗口界面中有一个按钮组件。

```
import java.awt.*;
import javax.swing.*;
public class JFrameDemo{
    JFrame f;
    JButton b;
    Container c;
    public JFrameDemo(){
        f=new JFrame("JFrame Demo");
        b=new JButton("Press me");
        c=f.getContentPane();//获取内容面板容器
        c.add(b);          //为内容面板容器添加按钮组件
        f.setSize(200,200);
        f.setVisible(true);
    }
    public static void main(String args[]){
        new JFrameDemo();
    }
}
```

2.4 布局管理器

Java 语言中，将创建的组件放置到窗口中时，需要设置窗口界面的格式，这时候必须使用布局管理器（layout manager）的类，来排列界面上的组件。

2.4.1 布局管理器概述

当组件被加入容器中时，由布局管理器排列组件。同时在整个程序编写的过程中，容器内的所有组件都由布局管理器来进行管理。Java 中的布局管理器包括：FlowLayout、GridLayout、BorderLayout、CardLayout 和 GridBagLayout。当创建好需要的布局管理器后，就可以调用容器的 `setLayout()` 方法，来设定该容器的布局方式。在组件加入容器中之前，如果不设定布局管理器的方式，则会采用默认的布局管理器。面板的默认布局管理器是 FlowLayout；窗口及框架的默认布局管理器是 BorderLayout。下面具体介绍几种主要的布局管理器。

2.4.2 流布局管理器 FlowLayout

FlowLayout 类是流布局管理器。这种管理器的特点是：组件在容器内依照指定方向，按照组件添加的顺序依次加入容器中。这个指定方向取决于 FlowLayout 管理器的组件方向属性。该属性有两种可能：从左到右方向和从右向左方向。默认情况下，这个指定方向是从左到右的。许多容器采用流布局管理器作为默认布局管理方式，例如 JPanel。

2.4.3 边界布局 BorderLayout

边界（边框）布局是使用 BorderLayout 类来创建的。该布局方式将容器分为 5 部分，分别是：东、西、南、北和中央。中央是一个大组件，四周是 4 个小的组件。

如果一个面板被设置成边界布局后，所有填入某一区域的组件都会按照该区域的空间进行调整，直到完全充满该区域。如果此时将面板的大小进行调整，则四周区域的大小不会发生改变，只有中间区域被放大或缩小。

【例 2-6】 一个使用边界布局管理器的示例，代码如下。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class BorderLayoutTest {
    public BorderLayoutTest() {
        JFrame jf = new JFrame();
        Container contentPane = jf.getContentPane();
        //设置容器的布局方式为 BorderLayout
        contentPane.setLayout(new BorderLayout());
        contentPane.add(new JButton("东"), BorderLayout.EAST); //按钮放到东侧
        contentPane.add(new JButton("西"), BorderLayout.WEST); //按钮放到西侧
        contentPane.add(new JButton("南"), BorderLayout.SOUTH); //按钮放到南侧
        contentPane.add(new JButton("北"), BorderLayout.NORTH); //按钮放到北侧
        //将标签放到中间
        contentPane.add(new JLabel("中", JLabel.CENTER), BorderLayout.CENTER);
        jf.setTitle("BorderLayout 布局管理器示例"); //设置标题
        jf.pack();
        jf.setVisible(true);
        //对一个窗口进行关闭操作的事件
    }
}
```



```

        jf.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args) {
        new BorderLayoutTest();
    }
}

```

以上程序中,首先创建容器并设置布局格式为边界布局;然后在容器中添加5个按钮,分别设置为东、西、南、北、中;最后对窗口属性进行设置。程序运行效果对比如图2-5和图2-6所示。

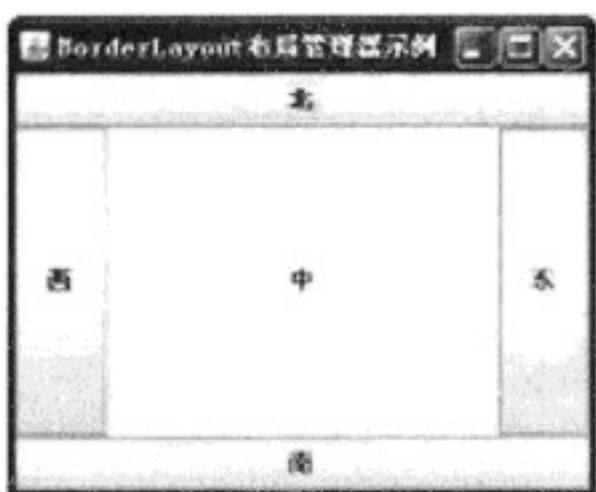


图 2-5 边界布局管理器效果

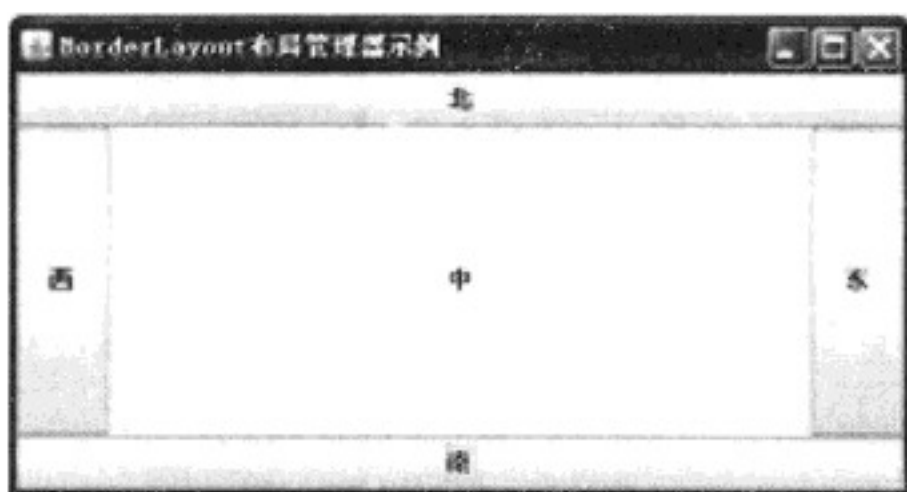


图 2-6 边界布局管理器效果对比

2.4.4 卡片布局管理器 CardLayout

卡片布局管理器能将容器中的组件看成不同的卡片层叠排列,每次只能显示一张卡片,每张卡片只能容纳一个组件。初次显示时,显示的是第一张卡片。卡片布局管理器是通过 AWT 包的 CardLayout 类来创建的。用一个形象的比喻,卡片布局管理器 CardLayout 就像是一副扑克牌,而每一次只能看到最上面的一张。

选项卡窗格 (JTabbedPane) 的默认布局是 CardLayout。

2.4.5 网格布局管理器 GridLayout

网格布局是一种常用的布局方式,将容器的区域划分成矩形网格,每个矩形大小规格一致,组件可以放置在其中的一个矩形中。Java 语言中通过 java.awt.GridLayout 类创建网格布局管理器对象,实现对容器中的各组件的网格布局排列。具体的排列方向取决于容器的组件方向属性,组件方向属性有两种:从左向右和从右向左。用户可以根据实际要求来设定方向属性,默认的方向是从右向左。

【例 2-7】 下面是网格布局管理器 GridLayout 的示例,运行效果如图 2-7 所示。

```

import javax.swing.*;
import java.awt.*;
public class GridLayoutExample extends JFrame{
    JButton buttons[];
    GridLayout layout;
    public void init(){
        this.setTitle("网格布局管理器示例");
        layout=new GridLayout(4,3,20,10);
        setLayout(layout);           //设置 4 行 3 列的网格布局
    }
}

```



```
        buttons=new JButton[10];
        for(int i=0;i<buttons.length;i++){
            buttons[i]=new JButton("按钮"+(i+1));
            add(buttons[i]); }
    }
    public static void main(String args[]){
        GridLayoutExample gle=new GridLayoutExample();
        gle.init();
        gle.pack();
        gle.setVisible(true);
        gle.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



图 2-7 网格布局效果

2.4.6 null 布局管理器

null 布局管理器是空的管理器，这意味着用户可以利用 GUI 组件对象的方法 setBounds()自行设定各个组件的位置和大小。通常，GUI 组件具有 setBounds()的两种形式：void setBounds(int x,int y,int width,int height)和 setBounds(Rectangle rect)。null 布局管理器的这种方式应用起来比较复杂，多用于组件可以随意移动的情况。

2.5 常用事件处理

开发应用程序时，为了实现用户与软件的交互，对事件的处理是必不可少的。常用事件包括：动作事件处理、鼠标事件处理和键盘事件处理。一般事件类处于 java.awt.event 包中。

2.5.1 动作事件处理

动作事件由 ActionEvent 类定义，最常用的是当单击按钮后将产生动作事件，可以通过实现 ActionListener 接口的类来处理相应的动作事件。

ActionListener 接口只有一个抽象方法，将在动作发生后被触发，例如单击按钮之后。ActionListener 接口的具体定义如下：

```
public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent e);
}
```

实现 ActionListener 接口的类必须给出抽象方法 actionPerformed() 的方法体，即对动作事件 ActionEvent 处理的代码。

【例 2-8】 按钮动作事件示例，程序运行界面如图 2-8 所示。

```
import java.awt.Color;
import javax.swing.*;
import java.awt.event.*;
public class Action2 extends JFrame{
    static JButton b1=new JButton("红色");
    static JButton b2=new JButton("蓝色");
    static JButton b3=new JButton("黄色");
    static JPanel p=new JPanel();
    static JLabel l=new JLabel("请单击下面按钮");
    public Action2(){
        super("动作事件");
        setBounds(10,20,220,200);
        l.setOpaque(true);
        l.setBounds(0,0,220,150);
        l.setHorizontalAlignment(JLabel.CENTER);
        add(l,"Center");
        p.add(b1);      p.add(b2);      p.add(b3);
        add(p,"South");
        b1.addActionListener(new B());
        b2.addActionListener(new B());
        b3.addActionListener(new B());
        setVisible(true);
    }
    public static void main(String args[]){
        Action2 f=new Action2(); }
}
class B implements ActionListener{
    public void actionPerformed( ActionEvent e){
        if(e.getSource()== Action2.b1){
            Action2.l.setText("按下的是红色按钮");
            Action2.l.setBackground(Color.red);
        }
        if(e.getSource()== Action2.b2){
            Action2.l.setText("按下的是蓝色按钮");
            Action2.l.setBackground(Color.blue);
        }
        if(e.getSource()== Action2.b3){
            Action2.l.setText("按下的是黄色按钮");
            Action2.l.setBackground(Color.yellow);
        }
    }
}
```



图 2-8 例 2-8 的界面

2.5.2 鼠标事件处理

鼠标事件由 `MouseEvent` 类捕获，所有的组件都能产生鼠标事件，可以通过实现 `MouseListener` 接口来处理相应的鼠标事件。

`MouseListener` 接口有 5 个抽象方法，分别在光标移入（出）组件时、鼠标按键被按下（释放）时以及发生单击事件时被触发。


```
        Mouse1 f=new Mouse1();
    }
    public void mouseEntered(MouseEvent e){
        System.out.println("鼠标进入标签");
    }
    public void mousePressed(MouseEvent e){
        if(e.getButton()==1)
            System.out.println("鼠标左键被按下");
        if(e.getButton()==2)
            System.out.println("鼠标滚轮");
        if(e.getButton()==3)
            System.out.println("鼠标右键被按下");
    }
    public void mouseReleased(MouseEvent e){
        System.out.println("鼠标被释放");
    }
    public void mouseClicked(MouseEvent e){
        System.out.println("鼠标单击");
        System.out.println("鼠标单击了"+e.getClickCount()+"次");
    }
    public void mouseExited(MouseEvent e){
        System.out.println("鼠标移除了标签");    }
}
```

运行结果：

```
鼠标进入标签
鼠标左键被按下
鼠标被释放
鼠标单击
鼠标单击了1次
鼠标移除了标签
鼠标进入标签
鼠标移除了标签
```

2.5.3 键盘事件处理

键盘事件由 `KeyEvent` 类捕获，最常用的是当向文本框输入内容时将发生键盘事件，可以通过实现 `KeyListener` 接口来处理相应的键盘事件。

`KeyListener` 接口有 3 个抽象方法，分别在发生击键事件、键被按下和释放时触发，`KeyListener` 接口的具体定义如下：

```
public interface KeyListener extends EventListener {
    public void keyTyped(KeyEvent e);
    public void keyPressed(KeyEvent e);
    public void keyReleased(KeyEvent e);
}
```

`KeyEvent` 类中比较常用的方法如表 2-3 所示。

表 2-3

KeyEvent 类中的常用方法

方 法	功 能
getSource()	用来获得触发此次事件的组件对象，返回值为 Object 类型
getKeyChar()	用来获得与此事件中的键相关联的字符
getKeyCode()	用来获得与此事件中的键相关联的整数 keyCode
getKeyText(int keyCode)	用来获得描述 keyCode 的标签，例如“F1”、“HOME”等
isActionKey()	用来查看此事件中的键是否为“动作”键
isControlDown()	用来查看“Ctrl”键在此次事件中是否被按下
isAltDown()	用来查看“Alt”键在此次事件中是否被按下
isShiftDown()	用来查看“Shift”键在此次事件中是否被按下

【例 2-10】 键盘事件示例。

```
import javax.swing.*;
import java.awt.event.*;

public class Key1 extends JFrame implements KeyListener {
    JTextArea t = new JTextArea();
    public Key1() {
        super("键盘事件");
        setBounds(0, 0, 400, 300);
        JScrollPane sp = new JScrollPane();
        sp.setViewportView(t);
        add(sp, "Center");
        t.addKeyListener(this);
        setVisible(true);
    }
    public void keyPressed(KeyEvent e) {
        String keyText = KeyEvent.getKeyText(e.getKeyCode());
        if (e.isActionKey())
            System.out.println("您按下的是动作键 " + keyText + " ");
        else {
            System.out.println("您按下的是非动作键 " + keyText + " ");
        }
    }
    public void keyTyped(KeyEvent e) {
        System.out.println("此次输入的是 " + e.getKeyCode() + " ");
    }
    public void keyReleased(KeyEvent e) {
        System.out.println("您释放的是 " + e.getKeyChar() + " ");
    }
    public static void main(String args[]) {
        Key1 f = new Key1();
    }
}
```


第 3 章

Java 图形处理和 Java 2D

Java 语言的类库提供了丰富的绘图方法，其中大部分对图形、文本、图像的操作方法都定义在 Graphics 类中，Graphics 类是 java.awt 程序包的一部分。本章介绍的内容包括颜色、字体处理、基本图形绘制方法、文本处理以及 Java 2D 中 Graphics2D 提供的基本图形绘制和图形特殊效果处理等方面的内容。

3.1 Java 图形坐标系统和图形上下文

要将图形在屏幕上绘制出来，必须有一个精确的图形坐标系统来给该图形定位。与大多数其他计算机图形系统所采用的二维坐标系统一样，Java 的坐标原点 $(0, 0)$ 位于屏幕的左上角，坐标度量以像素为单位，水平向右为 x 轴的正方向，竖直向下为 y 轴的正方向，每个坐标点的值表示屏幕上的一个像素点的位置，所有坐标点的值都取整数，如图 3-1 所示。这种坐标系统与传统坐标系统（见图 3-2）有所不同。

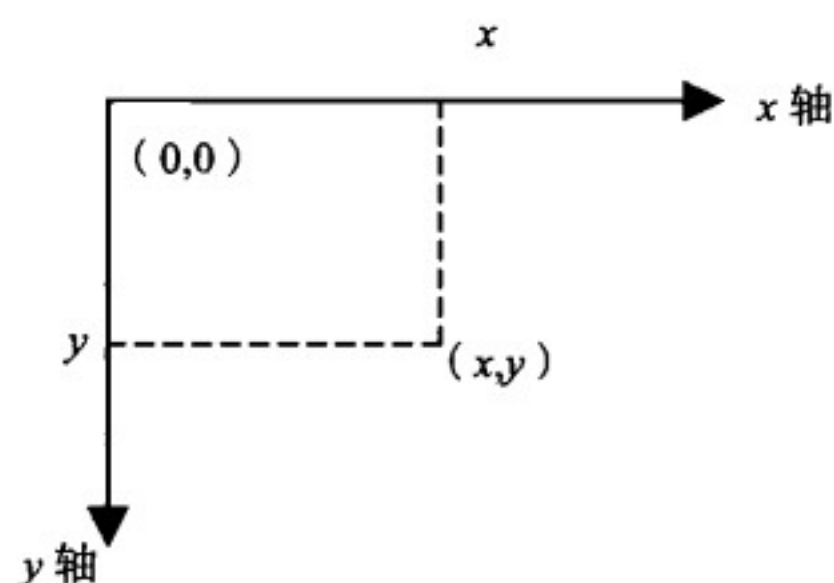


图 3-1 Java 坐标系

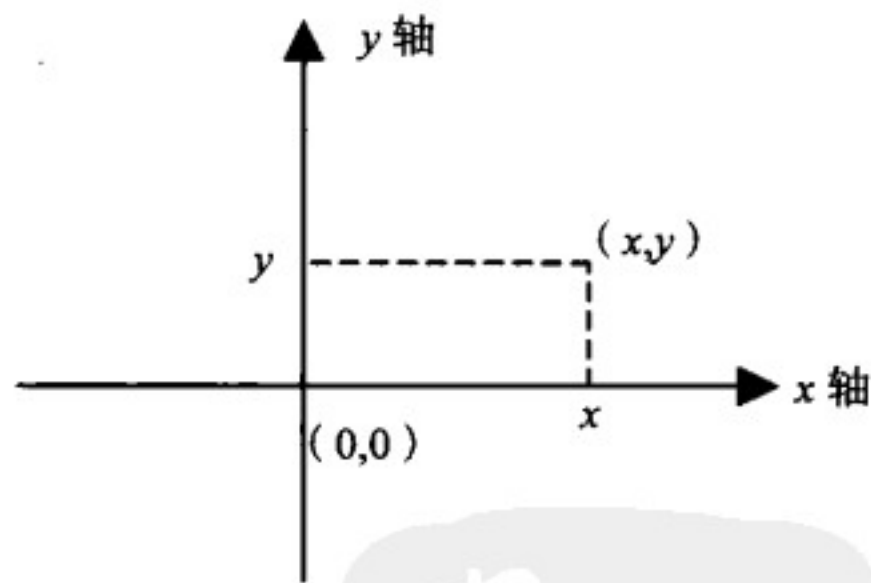


图 3-2 传统坐标系

在屏幕上绘制图形时，所有输出都通过一个图形上下文（graphics context）来产生。图形上下文有时也称为图形环境，是指允许用户在屏幕上绘制图形的信息，它由 Graphics 类封装，Graphics 类的对象可以通过 Component 类的 getGraphics() 方法获得。图形上下文表示一个绘制图层，如组件的显示区、打印机上的一页、或者一个屏幕外的图像缓冲区。它提供了绘制 3 种图形对象（形状、文本和图像）的方法。

在 Java 中，可以通过 Graphics 的对象对图形上下文进行管理，Graphics 类定义了多种绘图函

数，用户可以通过其提供的函数实现不同的图形绘制和处理。在游戏程序开发中，常常在组件的 `paint()` 方法内获得 `java.awt` 包中的 `Graphics` 类的对象，然后调用 `Graphics` 类中相应的绘制函数来实现输出。`paint()` 方法是 `java.awt.Component` 类（所有窗口对象的基类）所提供的一个方法，当系统需要重新绘制组件时，将调用该方法。`paint()` 方法只有一个参数，该参数是 `Graphics` 类的实例。下面给出一个实例。

```
public void paint(Graphics g)
{
    Color myColor= new Color(255, 0, 0);
    g.setColor(myColor);
    g.drawString("这是 Java 中的带颜色的文字串", 100,100) ;
    g.drawRect( 10,10,100 ,100) ;
}
```

绘制组件的时机如下所述。

（1）组件外形发生变化时：如窗口的大小、位置、图标等显示区域更新时，AWT 自动从高层直到叶结点组件相应地调用各组件的 `paint()` 方法，但这可能有一个延迟感。

（2）程序员也可直接调用某一个组件的 `repaint()` 或 `paint()` 方法，以立即更新外观（例如在添加新的显示内容后）。



如果要求保留上次的输出结果时可以调用 `paint()` 方法，不要求保留上次的输出结果只希望用户能看到最新的输出结果时则可以调用 `repaint()` 方法。

3.2 Color 类

可以使用 `java.awt.Color` 类为绘制的图形设置颜色。`Color` 类使用了 `sRGB`（`standard RGB`，标准 `RGB`）颜色空间来表示颜色值。颜色由红（`R`）、绿（`G`）、蓝（`B`）三原色构成，每种原色的强度用一个 `byte` 值表示，每种原色取值范围从 0（最暗）~ 255（最亮），可以根据这 3 种颜色值的不同组合，显示不同的颜色效果，例如（0,0,0）表示黑色，（255,255,255）表示白色。

在 Java 中 `Color` 类定义了 13 种颜色常量供用户使用，它们分别为：`Color.black`、`Color.blue`、`Color.cyan`、`Color.darkGray`、`Color.gray`、`Color.green`、`Color.lightGray`、`Color.magenta`、`Color.orange`、`Color.pink`、`Color.red`、`Color.white` 和 `Color.yellow`。从 JDK 1.4 开始，也可以使用 `Color` 类中定义的新常量，它们与上述颜色常量一一对应，分别为：`Color.BLACK`、`Color.BLUE`、`Color.CYAN`、`Color.DARK_GRAY`、`Color.GRAY`、`Color.GREEN`、`Color.LIGHT_GRAY`、`Color.MAGENTA`、`Color.ORANGE`、`Color.PINK`、`Color.RED`、`Color.WHITE` 和 `Color.YELLOW`。

除此之外，用户也可以通过 `Color` 类提供的构造方法 `Color(int r,int g,int b)` 创建自己需要的颜色。该构造方法通过指定红、绿、蓝 3 种颜色的值来创建一个新的颜色，参数 `r`、`g`、`b` 的取值范围为 0~255。例如：

```
Color color = new Color(255,0,255);
```

一旦用户生成了自己需要的颜色，就可以通过 `java.awt.Component` 类中的 `setBackground (Color c)` 和 `setForeground (Color c)` 方法来设置组件的背景色和前景色，也可以使用该颜色作为当前的绘图颜色。

3.3 Font 类和 FontMetrics 类

3.3.1 Font 类

可以使用 java.awt.Font 类创建字体对象。Java 提供了物理字体和逻辑字体两种字体。AWT 定义了 5 种逻辑字体，分别为 SansSerif、Serif、Monospaced、Dialog 和 DialogInput。

Font 类的构造方法如下：

```
Font (String name, int style, int size);
```

其中参数 name 为字体名，可以设置为系统上可用的任一种字体，如 SansSerif、Serif、Monospaced、Dialog 或 DialogInput 等；参数 style 为字型，可以设置为 Font.PLAIN、Font.BOLD、Font.ITALIC 或 Font.BOLD + Font.ITALIC 等；参数 size 为字号，其取值为正整数。例如：

```
Font font = new Font("Serif", Font.ITALIC,10);
```

如果需要找到系统上的所有可用字体，可以通过创建 java.awt.GraphicsEnvironment 类的静态方法 getLocalGraphicsEnvironment()的实例，调用 GetAllFonts()方法来获得系统的所有可用字体；或者通过 getAvailableFontFamilyName()方法来取得可用字体的名字。例如：

在生成可用的字体对象后，可以通过 java.awt.Component 类中的 setFont(Font f)方法设置组件的字体。

【例 3-1】 在控制台输出系统所有的可用字体，程序运行结果如图 3-3 所示。

```
//ex1.java
import java.awt.*;
public class ex1 {
    public static void main(String[] args) {
        GraphicsEnvironment e =
            GraphicsEnvironment.getLocalGraphicsEnvironment();
        String[] fontNames = e.getAvailableFontFamilyNames();//获得可用字体的名称
        int j=0;
        for(int i = 0; i < fontNames.length; i++)
        {
            System.out.printf("%25s",fontNames[i]);
            j++;
            if(j%3==0) System.out.println();
        }
    }
}
```

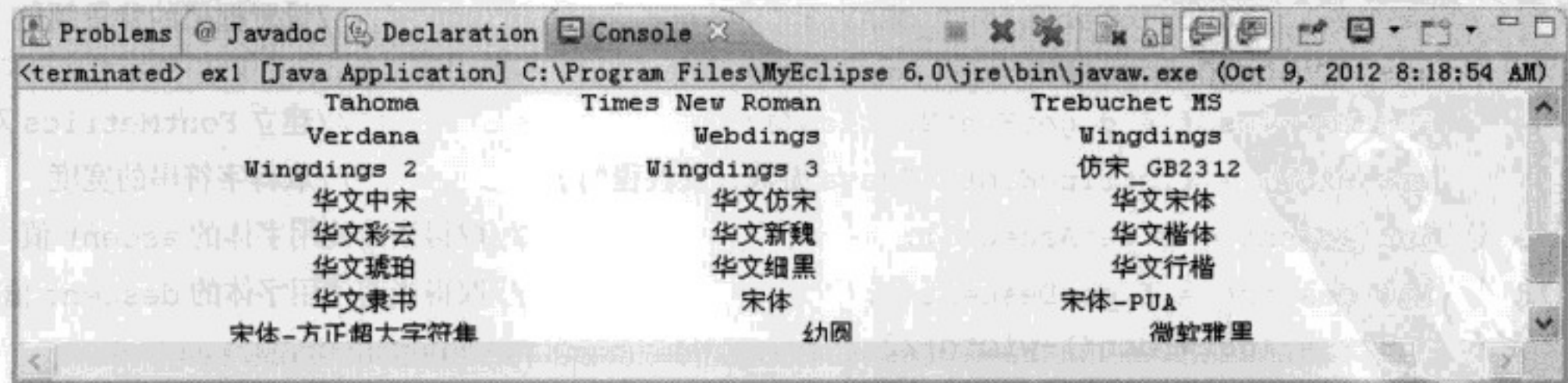


图 3-3 例 3-1 的运行结果

3.3.2 FontMetrics 类

使用 `drawString(String s, int x, int y)` 方法我们可以指定在框架的 (x, y) 位置开始显示字符串，但是如果想在框架的中央显示字符串，则需要使用 `FontMetrics` 类。`FontMetrics` 类是一个抽象类，要使用 `FontMetrics` 对象，可以通过调用 `Graphics` 类中的 `getFontMetrics()` 方法。`FontMetrics` 类定义字体的度量，给出了关于在特定的组件上描绘特定字体的信息。这些字体信息包括 `ascent`（上升量）、`descent`（下降量）、`leading`（前导宽度）和 `height`（高度）。其中 `leading` 用于描述两行文本间的间距，如图 3-4 所示。

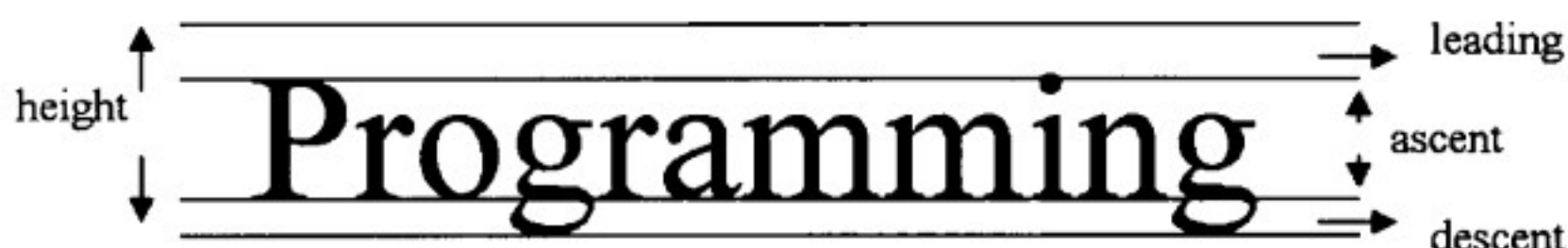


图 3-4 字体信息示意图

`FontMetrics` 类提供了下面几种方法用于获取 `ascent`、`descent`、`leading` 和 `height`：

- `int getAscent();` //取得由当前 `FontMetrics` 对象描述的字体的 `ascent` 值。
- `int getDescent();` //取得由当前 `FontMetrics` 对象描述的字体的 `descent` 值。
- `int getLeading();` //取得由当前 `FontMetrics` 对象描述的字体的 `leading` 值。
- `int getHeight();` //取得使用当前字体的一行文本的标准高度。

【例 3-2】 在框架中央位置显示字符串“Java 游戏开发教程”，并将字体设置为楷体、粗斜体、大小为 30，颜色为红色，将框架背景设置为淡灰色。程序源代码见 `FontMetricsDemo.java`，程序运行结果如图 3-5 所示。

```
//FontMetricsDemo.java
import java.awt.*;
import javax.swing.JFrame;
public class FontMetricsDemo extends JFrame{
    public FontMetricsDemo() {
        super();
        setTitle("FontMetrics 演示");
        setSize(300,200);
        setVisible(true);
    }
    public void paint(Graphics g) {
        Font font = new Font("楷体",Font.BOLD+Font.ITALIC,30); //建立字体
        g.setFont(font); //设置当前使用的字体
        setBackground(Color.LIGHT_GRAY); //设置框架的背景颜色
        g.setColor(Color.RED);
        FontMetrics f = g.getFontMetrics(); //建立 FontMetrics 对象
        int width = f.stringWidth("Java 游戏开发教程"); //取得字符串的宽度
        int ascent = f.getAscent(); //取得当前使用字体的 ascent 值
        int descent = f.getDescent(); //取得当前使用字体的 descent 值
        int x = (getWidth()-width)/2;
        int y = (getHeight()+ascent)/2;
        g.drawString("Java 游戏开发教程",x,y);
    }
}
```



```

    }
    public static void main(String[] args) {
        FontMetricsDemo fmd = new FontMetricsDemo();
        fmd.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

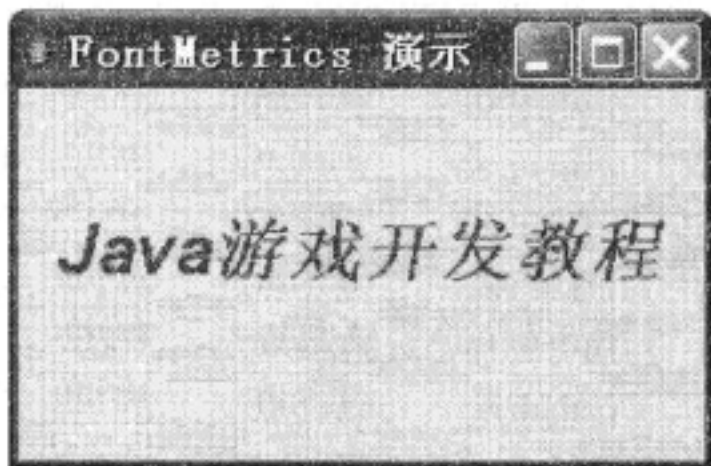


图 3-5 例 3-2 的运行结果

3.4 常用的绘图方法

3.4.1 绘制直线

在 Java 中可以使用下面方法绘制一条直线：

```
drawLine(int x1, int y1, int x2, int y2);
```

其中，参数 $x1$ 、 $y1$ 、 $x2$ 、 $y2$ 分别表示该直线的起点($x1, y1$)和终点($x2, y2$)的坐标值。

3.4.2 绘制矩形

Java 中提供了绘制空心矩形（只绘制矩形的轮廓）和填充矩形的方法，分别针对普通直角矩形、圆角矩形和三维矩形有不同的绘制方法。

（1）普通直角矩形

可以使用下面方法绘制普通直角矩形的轮廓：

```
drawRect(int x, int y, int width, int height);
```

如果需要绘制一个有填充颜色的普通直角矩形，可以使用下面的方法：

```
fillRect(int x, int y, int width, int height);
```

这两种方法的参数含义相同， x 、 y 分别表示矩形左上角的 x 坐标和 y 坐标， $width$ 、 $height$ 分别表示矩形的宽和高。

（2）圆角矩形

可以使用下面的方法绘制圆角矩形的轮廓：

```
drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight);
```

如果需要绘制一个有填充颜色的圆角矩形，可以使用下面的方法：

```
fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight);
```

这两种方法的参数含义相同， x 、 y 分别表示矩形左上角的 x 坐标和 y 坐标， $width$ 、 $height$ 分别表示矩形的宽和高，参数 $arcWidth$ 和 $arcHeight$ 分别表示圆角弧的水平直径和竖直直径，如图 3-6 所示。

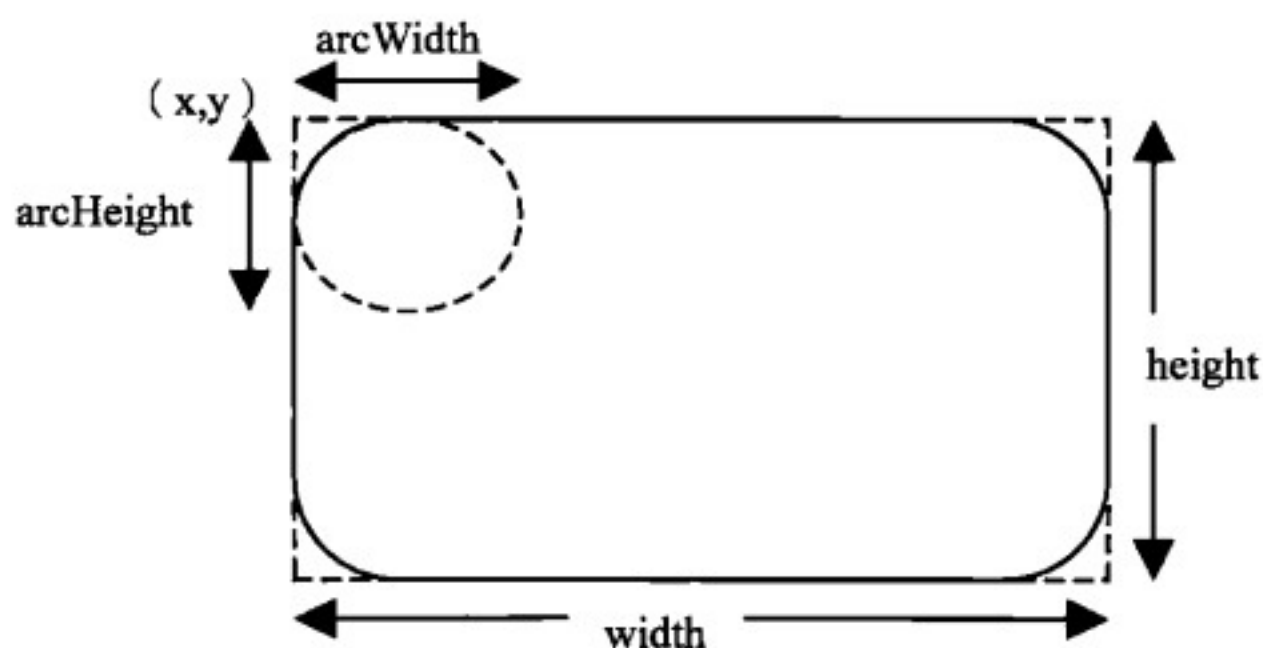


图 3-6 绘制圆角矩形示意图

(3) 三维矩形

可以使用下面方法绘制三维矩形的轮廓：

```
draw3DRect(int x, int y, int width, int height, boolean raised);
```

如果需要绘制一个有填充颜色的三维矩形，可以使用下面的方法：

```
fill3DRect(int x, int y, int width, int height, boolean raised);
```

这两种方法的参数含义相同， x 、 y 分别表示矩形左上角的 x 坐标和 y 坐标， $width$ 、 $height$ 分别表示矩形的宽和高， $raised$ 为真（True）表示矩形从表面凸起， $raised$ 为假（False）表示矩形从表面凹进。

3.4.3 绘制椭圆

可以使用下面的方法绘制空心椭圆：

```
drawOval(int x, int y, int width, int height);
```

如果需要绘制一个有填充颜色的椭圆，可以使用下面的方法：

```
fillOval(int x, int y, int width, int height);
```

这两种方法的参数含义相同， x 、 y 分别表示该椭圆外接矩形左上角的 x 坐标和 y 坐标， $width$ 、 $height$ 分别表示外接矩形的宽和高，如图 3-7 所示。如果设置外接矩形为正方形，即 $width$ 和 $height$ 相等，则可以绘制圆。

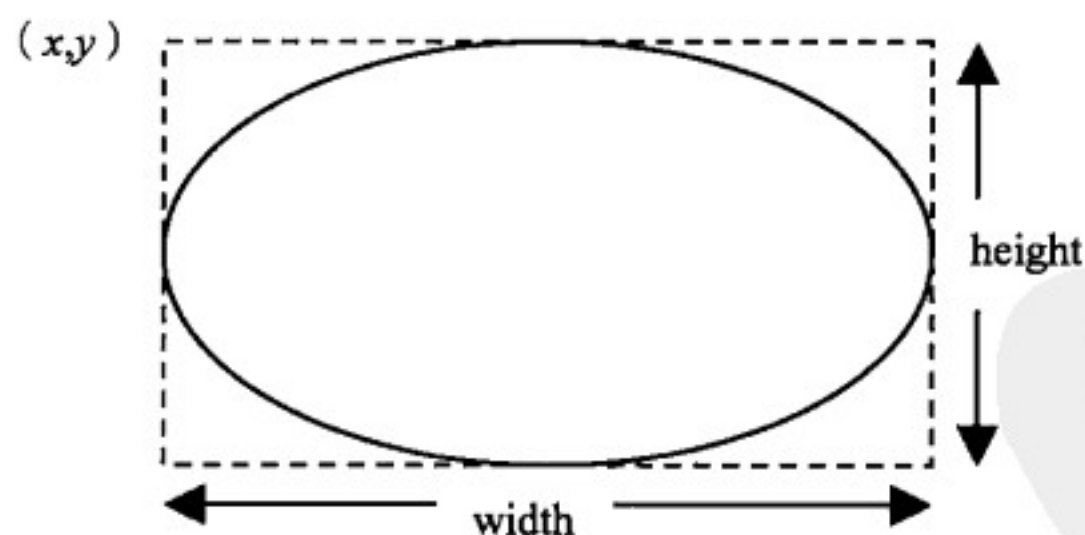


图 3-7 绘制椭圆示意图

【例 3-3】 在框架中绘制直线、矩形和椭圆。程序源代码见 DrawImageDemo.java，程序运行结果如图 3-8 所示。

```
import java.awt.*;
import javax.swing.*;
public class DrawImageDemo extends JFrame{
    public DrawImageDemo() {
```



```
super();
setTitle("Draw Line Rectangle Ellipse");
setSize(300,300);
setVisible(true);
}
public void paint(Graphics g) {
    //绘制直线、空心矩形和空心椭圆
    g.setColor(Color.red);
    g.drawRect(10,30,getWidth()/2-50,getHeight()/2-50);
    g.drawOval(10,30,getWidth()/2-50,getHeight()/2-50);
    g.drawLine(10,30,5+getWidth()/2-50,30+getHeight()/2-50);

    //绘制填充色为淡灰色的 3D 矩形、圆角矩形和椭圆
    g.setColor(Color.LIGHT_GRAY);
    g.fill3DRect(10,180,getWidth()/2-50,getHeight()/2-50,true);
    g.fillRoundRect(130,30,getWidth()/2-50,getHeight()/2-50,30,40);
    g.fillOval(130,180,getWidth()/2,getHeight()/2-50);
}
public static void main(String[] args) {
    DrawImageDemo d = new DrawImageDemo();
    d.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

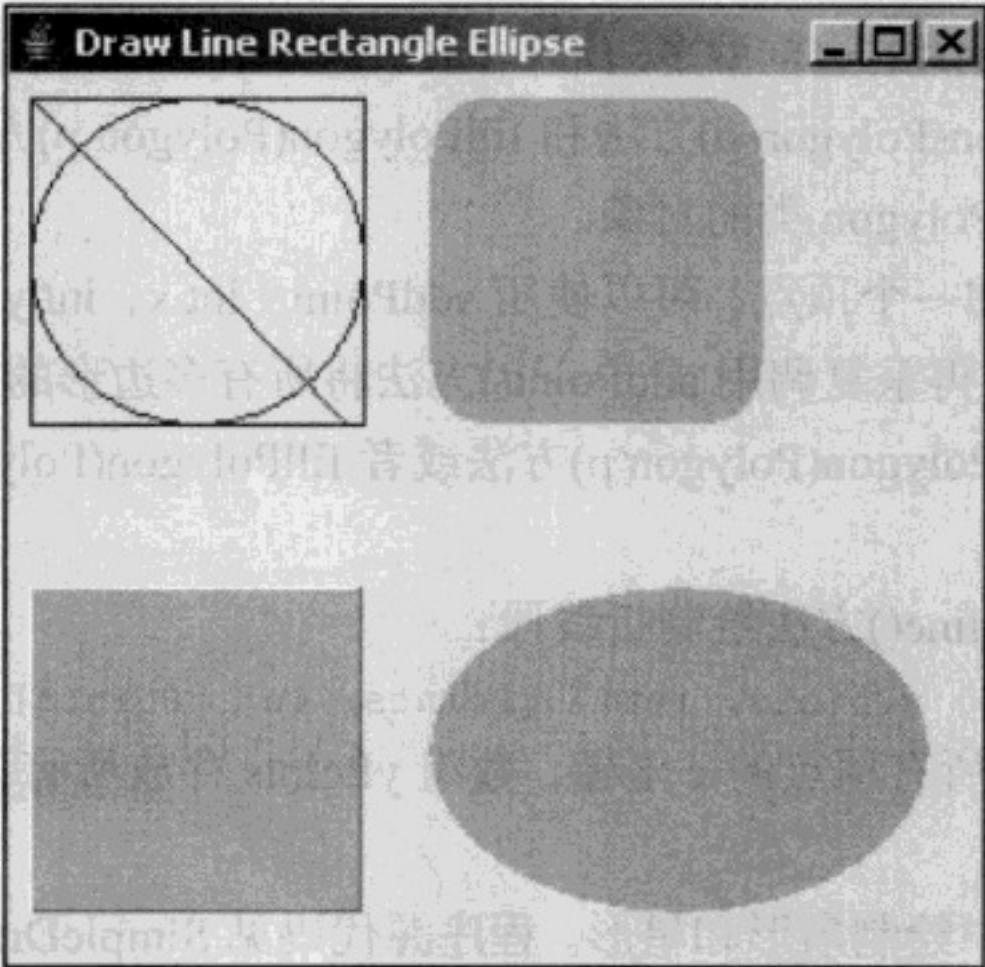


图 3-8 例 3-3 运行结果

3.4.4 绘制弧形

弧形可以看做椭圆的一部分，因此它的绘制也是根据其外接矩形进行的。通过 `drawArc()` 方法和 `fillArc()` 方法可以分别绘制弧线和扇形。这两种方法为：

```
drawArc(int x, int y, int width, int height, int startAngle, int arcAngle);
fillArc(int x, int y, int width, int height, int startAngle, int arcAngle);
```

其中 `x`、`y`、`width`、`height` 参数的含义与 `drawOval()` 方法的参数含义相同，参数 `startAngle` 表示该弧的起始角度，参数 `arcAngle` 表示生成角度（从 `startAngle` 开始转了多少度），且水平向右方向表示 0 度，从 0 度开始沿逆时针方向旋转为正角，如图 3-9 所示。

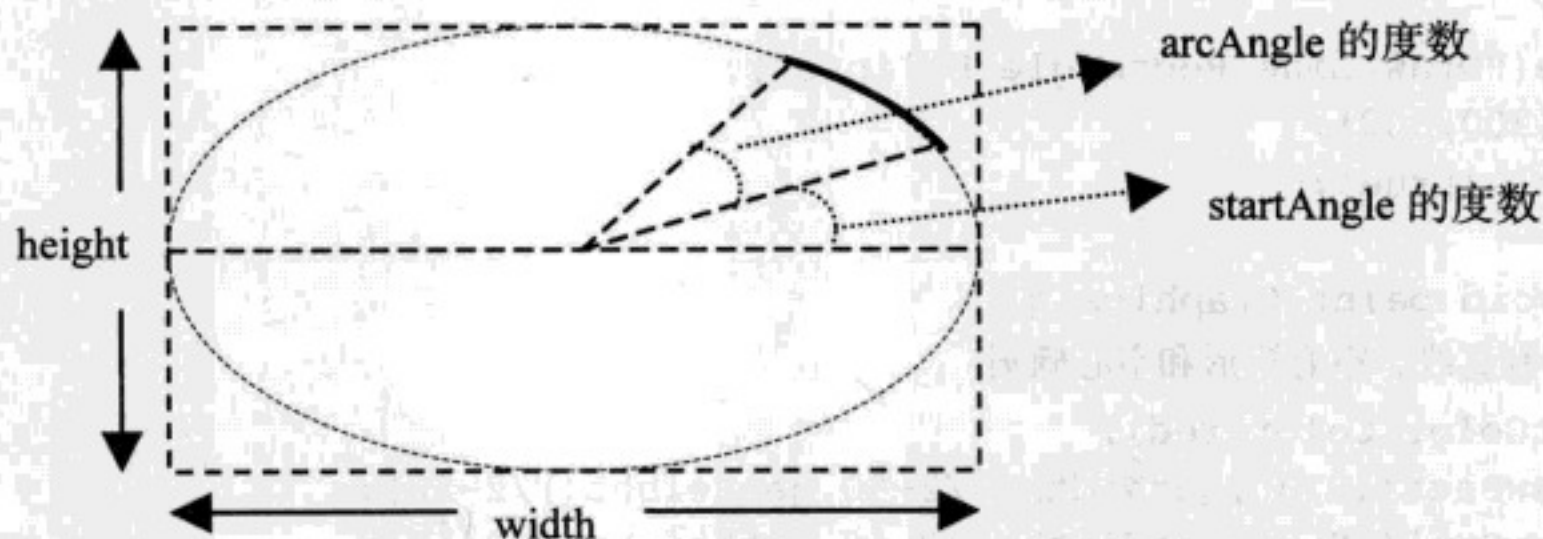


图 3-9 绘制弧形示意图

3.4.5 绘制多边形和折线段

(1) 绘制多边形

使用 `drawPolygon()` 方法和 `fillPolygon()` 方法可以分别绘制多边形的外框轮廓和填充多边形。

```
drawPolygon(int[] xPoints, int[] yPoints, int nPoints);
fillPolygon(int[] xPoints, int[] yPoints, int nPoints);
```

其中多边形的顶点是由数组 `xPoints` 和 `yPoints` 中对应下标的相应元素组成的坐标来指定，数组 `xPoints` 存放所有顶点的 x 坐标，数组 `yPoints` 存放所有顶点的 y 坐标，参数 `nPoints` 指定多边形的顶点个数。`drawPolygon()` 方法在绘制多边形时并不自动关闭多边形的最后一条边，而仅是一段开放的折线。因此，如果需要画封闭的边框型多边形，记住在数组的尾部再添上一个起始点的坐标。

除此以外，`drawPolygon(Polygon p)` 方法和 `fillPolygon(Polygon p)` 方法也可以用来绘制多边形。这两种方法的参数是一个 `Polygon` 类的对象。

如果想在多边形上增加一个顶点，可以使用 `addPoint(int x, int y)` 方法。因此可以通过先创建一个空的 `Polygon` 对象，再重复调用 `addPoint()` 方法将所有多边形的顶点加入创建的 `Polygon` 对象中，然后通过调用 `drawPolygon(Polygon p)` 方法或者 `fillPolygon(Polygon p)` 方法绘制多边形。

(2) 绘制折线段

可以使用 `drawPolygone()` 方法绘制折线段：

```
drawPolygone(int[] xPoints, int[] yPoints, int nPoints);
```

其中数组 `xPoints` 存放所有顶点的 x 坐标，数组 `yPoints` 存放所有顶点的 y 坐标，`nPoints` 指定折线段顶点的个数。

【例 3-4】 在 `JPanel` 中绘制扇形和星形。程序源代码见 `SimpleDraw.java`，程序运行结果如图 3-10 所示。

```
import java.awt.*;
import javax.swing.*;
public class SimpleDraw {
    public static void main(String[] args) {
        DrawFrame frame = new DrawFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
class DrawFrame extends JFrame {
    public DrawFrame() {
        setTitle("简单图形绘制");
        setSize(300, 300);
    }
}
```



```

        DrawPanel panel = new DrawPanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
    }
}

class DrawPanel extends JPanel {
    public void paint(Graphics g){
        int x1 = 50, y1 = 50, x2 = 50, y2 = 150;
        int radius = 100;        //半径
        int startAngle = -90;    //起始角度
        int arcAngle = 180;      //弧的角度
        g.drawLine(x1, y1, x2, y2); //画线
        g.drawArc(x1-radius/2, y1, radius, radius, startAngle, arcAngle);
        Polygon p = new Polygon();
        x1 += 150; y1 += 50; radius /= 2;
        for (int i = 0; i < 6; i++)
            p.addPoint((int)(x1 + radius * Math.cos(i * 2 * Math.PI / 6)),
                (int)(y1 + radius * Math.sin(i * 2 * Math.PI / 6)));
        g.drawPolygon(p);        //画六边形
    }
}

```

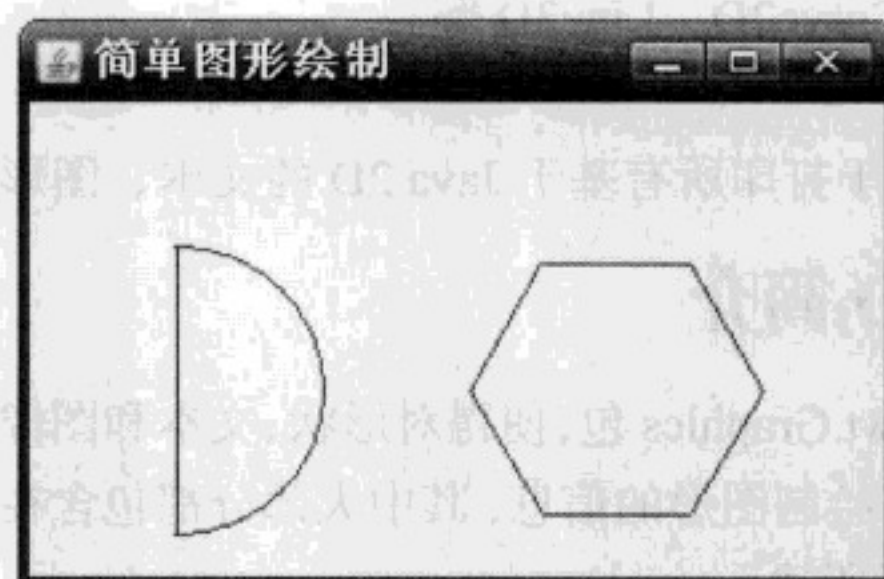


图 3-10 例 3-4 的运行结果

3.4.6 清除绘制的图形

可以使用 `clearRect()` 方法清除绘制的图形。`clearRect(int x, int y, int width, int height)` 用背景色填充指定矩形以达到清除该矩形的效果，也就是说当一个 `Graphics` 对象使用该方法时，相当于在使用一个“橡皮擦”。参数 `x`、`y` 是被清除矩形的左上角的坐标，另外两个参数是被清除矩形的宽和高。

3.5 Java 2D 简介

3.5.1 Java 2D API

Java 2D API (Application Programming Interface) 增强了抽象窗口工具包 (AWT) 的图形、文本和图像功能，可以创建高级图形库，开发更为强大的用户接口和新型的 Java 应用程序。Java 2D API 对 AWT 进行了扩展，提供了更加灵活、功能更全面的绘制包，使其支持更多的图形绘

制操作。

Java 2D 是 Java 核心类库的一部分，它包含的包有如下几个。

- java.awt

java.awt 包含了一些新增的 2D API 类和接口。其中 Graphics2D 继承自 java.awt.Graphics，是描绘 2D 图形的对象。同时在 Graphics2D 中新增了许多状态属性，如 Stroke、Paint、Clip、Transform 等。

- java.awt.image

- java.awt.image.renderable

java.awt.image和java.awt.image.renderable包中包含用于图像定义和绘制的类和接口。

- java.awt.color

- java.awt.font

java.awt.font 包中包含用于文本布局和字体定义的类和接口；Font 类的功能已得到增强，可支持详细的字体信息规范和复杂印刷特性的使用。Font 对象代表系统上可用字样集的字样实例。

- java.awt.geom

java.awt.geom 包则包含可以勾勒任何形状的 GeneralPath 类。它可以由许多不同种类的 subpath 构成，例如 lines 和 quadratic curves 等。为了兼顾方便性，在此包中还定义了许多基本几何图形，包括 Arc2D、CubicCurve2D、Line2D 等。

- java.awt.print

java.awt.print 包中包含用于打印所有基于 Java 2D 的文本、图形和图像的类及接口。

3.5.2 Graphics2D 简介

Graphics2D 扩展了 java.awt.Graphics 包，使得对形状、文本和图像的控制更加完善。Graphics2D 对象保存了大量用来确定如何绘制图形的信息，其中大部分都包含在一个 Graphics2D 对象的 6 个属性中，这 6 个属性分别如下所述。

(1) 绘制 (paint)：该属性确定所绘制线条的颜色，以及填充图形的颜色和图案等。用户可以通过 setPaint(Paint p)方法进行该属性值的设置。

(2) 画笔 (stroke)：该属性可以确定线条的类型以及粗细，还有线段端点的形状。用户可以通过 setStroke(Stroke s)方法进行该属性值的设置。

(3) 字体 (font)：该属性可以确定所显示字符串的字体。用户可以通过 setFont(Font f)方法进行该属性值的设置。

(4) 转换 (transform)：该属性确定了图形绘制过程中要应用的转换方法，通过指定转换方法可将所画内容进行平移、旋转和缩放。用户可以通过 setTransform()方法进行该属性值的设置。

(5) 剪切 (clip)：该属性定义了组件上某区域的边界。用户可以通过 setClip(Clip c)方法进行该属性值的设置。

(6) 合成 (composite)：该属性定义了如何绘制重叠的几何图形，使用合成规则可以确定重叠区域的显示效果。用户可以通过 setComposite(Composite c)方法来设置该属性的值。

一般情况下，我们使用 Graphics2D 对象的方法进行图形的绘制，Graphics2D 对象的常用方法如下所述。

(1) abstract void clip(Shape s)：将当前 Clip 与指定 Shape 的内部区域相交，并将 Clip 设置为所得的交集。

- (2) `abstract void draw(Shape s)`: 使用当前 `Graphics2D` 上下文的设置勾画 `Shape` 的轮廓。
- (3) `abstract void drawImage(BufferedImage img, BufferedImageOp op, int x, int y)`: 呈现使用 `BufferedImageOp` 过滤的 `BufferedImage` 应用的呈现属性, 包括 `Clip`、`Transform` 和 `Composite` 属性。
- (4) `abstract boolean drawImage(Image img, AffineTransform xform, ImageObserver obs)`: 呈现一幅图像, 在绘制前进行从图像空间到用户空间的转换。
- (5) `abstract void drawString(String s, float x, float y)`: 使用 `Graphics2D` 上下文中当前文本属性状态呈现由指定 `String` 指定的文本。
- (6) `abstract void drawString(String str, int x, int y)`: 使用 `Graphics2D` 上下文中的当前文本属性状态呈现指定 `String` 的文本。
- (7) `abstract void fill(Shape s)`: 使用 `Graphics2D` 上下文的设置, 填充 `Shape` 的内部区域。
- (8) `abstract Color getBackground()`: 返回用于清除区域的背景色。
- (9) `abstract Composite getComposite()`: 返回 `Graphics2D` 上下文中的当前 `Composite`。
- (10) `abstract Paint getPaint()`: 返回 `Graphics2D` 上下文中的当前 `Paint`。
- (11) `abstract Stroke getStroke()`: 返回 `Graphics2D` 上下文中的当前 `Stroke`。
- (12) `abstract boolean hit(Rectangle rect, Shape s, boolean onStroke)`: 检查指定的 `Shape` 是否和设备空间中的指定 `Rectangle` 相交。
- (13) `abstract void rotate(double theta)`: 将当前的 `Graphics2D Transform` 与旋转转换连接。
- (14) `abstract void rotate(double theta, double x, double y)`: 将当前的 `Graphics2D Transform` 与平移后的旋转转换连接。
- (15) `abstract void scale(double sx, double sy)`: 将当前 `Graphics2D Transform` 与可缩放转换连接。
- (16) `abstract void setBackground(Color color)`: 设置 `Graphics2D` 上下文的背景色。
- (17) `abstract void setComposite(Composite comp)`: 为 `Graphics2D` 上下文设置 `Composite`。
`Composite` 用于所有绘制方法中, 例如 `drawImage`、`drawString`、`draw` 和 `fill`。它指定新的像素如何在呈现过程中与图形设备上的现有像素组合。
- (18) `abstract void setPaint(Paint paint)`: 为 `Graphics2D` 上下文设置 `Paint` 属性。
- (19) `abstract void setStroke(Stroke s)`: 为 `Graphics2D` 上下文设置 `Stroke`。
- (20) `abstract void setTransform(AffineTransform Tx)`: 重写 `Graphics2D` 上下文中的 `Transform`。
- (21) `abstract void shear(double shx, double shy)`: 将当前 `Graphics2D Transform` 与剪裁转换连接。
- (22) `abstract void translate(double tx, double ty)`: 将当前的 `Graphics2D Transform` 与平移转换连接。
- (23) `abstract void translate(int x, int y)`: 将 `Graphics2D` 上下文的原点平移到当前坐标系统中的点(`x`, `y`)。

3.5.3 Graphics2D 绘制

`Graphics2D` 是 `Graphics` 类的子类, 也是一个抽象类, 不能实例化 `Graphics2D` 对象, 为了使用 `Graphics2D`, 可以通过 `Graphics` 对象传递一个组件的绘制方法给 `Graphics2D` 对象。方法如下面的代码段所示:

```
public void paint(Graphics g){
    Graphics2D g2=(Graphics 2D)g;
    ...
}
```


Java 2D API 提供了几种定义点、直线、曲线、矩形、椭圆等常用几何对象的类，这些新几何类是 `java.awt.geom` 包的组成部分，包括 `Point2D`、`Line2D`、`Arc2D`、`Rectangle2D`、`Ellipse2D`、`CubicCurve2D` 等。每个类都有单精度和双精度两种像素定义方式，例如 `Point2D.double` 和 `Point2D.float`，`Line2D.double` 和 `Line2D.float` 等，用这些类可以很容易地绘制基本的二维图形对象。

【例 3-5】 使用 `Graphics2D` 绘制直线、矩形、曲线和椭圆，程序运行结果如图 3-11 所示。

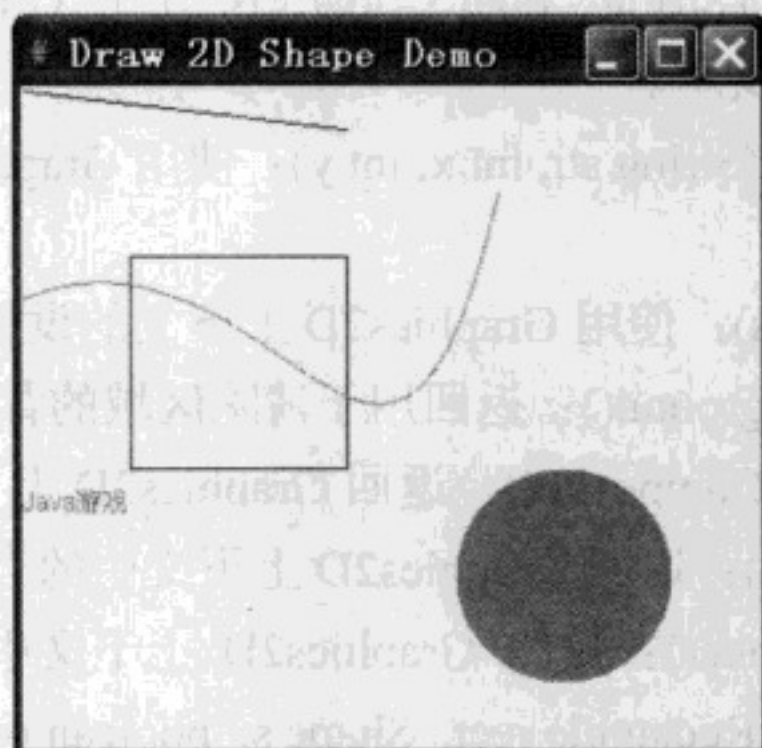


图 3-11 例 3-5 的运行结果

```
//MapJFrame.java
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

public class MapJFrame extends JFrame
{
    public MapJFrame()
    {
        super("Draw 2D Shape Demo");
        setSize(350,350);
        MapPane map=new MapPane();
        getContentPane().add(map);
    }

    public static void main(String [] arg)
    {
        MapJFrame frame=new MapJFrame();
        frame.setVisible(true);
    }
}

class MapPane extends JPanel
{
    public void paintComponent(Graphics g)
    {
        //建立 Graphics2D 对象
        Graphics2D g2 = (Graphics2D) g ;
        //建立 Line2D 对象
        Line2D l = new Line2D.Float(1.0f,2.0f,150.0f,20.0f);
        g2.draw(l);
        //建立 Rectangle2D 对象
        Rectangle2D r = new Rectangle2D.Float(50, 80, 100, 100);
        Color c = new Color(10,20,255);
        g2.setColor(c);
        g2.draw(r);
        //建立 Ellipse2D 对象
        Ellipse2D e = new Ellipse2D.Double(200, 180, 100, 100);
        g2.setColor(Color.GRAY);
```



```

        g2.fill(e);
        g2.drawString("Java 游戏", 0, 200);
        //建立 CubicCurve2D 对象, 比直线多了两个控制点
        CubicCurve2D cubic = new CubicCurve2D.Float(0, 100, 120, 50, 170, 270, 220, 50);
        g2.draw(cubic);
    }
}

```

3.5.4 Graphics2D 的属性设置

Graphics2D 通过设置对象的属性来确定如何绘制图形的信息, 在前面已经介绍了 Graphics2D 对象的 6 种属性, 包括 paint、stroke、font、transform、clip 和 composite。下面将介绍如何在 Graphics2D 的图形上下文中设置常用的属性 paint、stroke 和 composite。

(1) paint 用于填充绘制图形的颜色或图案, 在 Java 2D API 中提供了两种 paint 属性的填充方式: GradientPaint 和 TexturePaint。GradientPaint 定义在两种颜色间渐变的填充方式, 而 TexturePaint 是利用重复图像片段的方式定义一种纹理填充方式。我们可以用 setPaint() 方法设置定义好的填充方式并将其应用于绘制图形中。

GradientPaint 类提供了下面的构造方法来建立颜色渐变方式:

- ① GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2);
- ② GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2, boolean cyclic);
- ③ GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2);
- ④ GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2, boolean cyclic);

其中, 构造方法①和②中参数 x1、y1 指定颜色渐变的起点坐标, x2、y2 指定颜色渐变的终点坐标, 填充颜色从 color1 渐变至 color2。构造方法②中参数 cyclic 为 True 时, 填充方式与构造方法①定义的相同; 如果 cyclic 为 False, 则填充方式为非周期性渐变。构造方法③与构造方法①类似, 构造方法④与构造方法②类似, 只是在构造方法③和④中使用了 Point2D 对象来指定填充颜色渐变的起始 (p1) 和结束 (p2) 位置。

TexturePaint 类的构造方法为:

TexturePaint(BufferedImage txtr, Rectangle2D anchor)。

其中 txtr 用来定义一个单位的填充图像的材质, anchor 用来复制材质。

【例 3-6】 使用 GradientPaint 渐变填充方式和 TexturePaint 纹理填充方式绘制图形。程序源代码见 PaintDemo.java, 程序运行结果如图 3-12 所示。

```

import java.awt.*;
import java.awt.image.*;
import java.awt.geom.*;
import java.awt.event.*;
import javax.swing.*;
public class PaintDemo extends JFrame
{
    public PaintDemo ()
    {
        super("颜色渐变及纹理填充演示示例");
        setSize(200, 150);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}

```



图 3-12 例 3-6 的运行结果


```

public void paint(Graphics g)
{
    Graphics2D g2d=(Graphics2D)g;
    //以颜色从白到蓝周期性渐变设置绘图方式
    g2d.setPaint(new GradientPaint(20f,50f,Color.white,80f,80f,Color.blue, true));
    g2d.fill(new Rectangle2D.Double(20,50,70,70)); //以上述方式绘制矩形
    BufferedImage texture=new BufferedImage(5,5,1);//创建 BufferedImage 对象
    Graphics2D pattern=texture.createGraphics();//创建一个 Graphics2D 对象
    pattern.setColor(Color.blue); //设置纹理图案的颜色
    pattern.fillRect(0,0,5,5); //填充纹理图案
    pattern.setColor(Color.white); //设置纹理图案的颜色
    pattern.fillOval(0,0,5,5); //以外接圆方式填充纹理图案
    Rectangle rect=new Rectangle(0,0,5,5); //创建用于定位和复制纹理的 Rectangle2D 对象
    g2d.setPaint(new TexturePaint(texture,rect)); //以指定的纹理设置绘图方式
    g2d.fill(new Rectangle2D.Double(100,50,70,70)); //以纹理方式绘制矩形
}
public static void main(String args[]) //主方法 main()
{
    new PaintDemo ();
} //主方法 main() 结束
}

```

(2) **stroke** 用于在绘制图形的轮廓时确定线条的形状和粗细，通常使用 **BasicStroke** 对象来定义，通过 **setStroke()** 方法设定 **stroke** 的属性值。**BasicStroke** 定义的特性包括线条宽度、笔形样式、线段连接样式和短划线图案等。使用 **stroke** 属性设定图形轮廓的绘制方式时，首先调用 **setStroke()** 方法设定轮廓的绘制方式，然后使用 **setPaint()** 方法定义画笔如何绘制该图形，最后使用 **draw()** 方法来绘制该图形。

在 **BasicStroke** 中定义了一组基本的简单图形轮廓的直线绘制和点划线绘制方式，它提供了 3 种绘制粗线的末端样式：**CAP_BUTT**、**CAP_ROUND** 和 **CAP_SQUARE**；以及 3 种线段连接样式：**JOIN_BEVEL**、**JOIN_MITER** 和 **JOIN_ROUND**。

BasicStroke 类提供了下面的构造方法来建立画笔的绘制方式：

- ① **BasicStroke ()**;
- ② **BasicStroke (float width)**;
- ③ **BasicStroke (float width, int cap, int join)**;
- ④ **BasicStroke (float width, int cap, int join, float miterlimit)**;
- ⑤ **BasicStroke (float width, int cap, int join, float miterlimit, float[] dash, float dash_phase)**;

其中 **width** 表示轮廓线的宽度；**cap** 表示轮廓线末端的样式；**join** 表示相交线段的连接样式；**miterlimit** 表示在 **JOIN_MITER** 模式下若相交的尖形末端大于 **miterlimit**，则超过部分被削去；**dash** 为虚线的样式，数组内的值为短划线和空白间距的值；**dash_phase** 为虚线样式数组的起始索引。

【例 3-7】 使用 **BasicStroke** 类设定画笔绘制样式。程序源代码见 **StrokeDemo.java**，程序运行结果如图 3-13 所示。

```

//StrokeDemo.java
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

```



```

public class StrokeDemo extends JFrame{
    public StrokeDemo() {
        super();
        setTitle("Stroke Demo");
        setSize(300,200);
        setVisible(true);
    }
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g ;
        Line2D l = new Line2D.Double(30, 50, 100, 80);
        //创建一个 BasicStroke 对象来设置直线的绘制方式
        Stroke stroke = new BasicStroke(10.0f,
            BasicStroke.CAP_ROUND,BasicStroke.JOIN_BEVEL);
        g2.setStroke(stroke);
        g2.draw(l);

        Ellipse2D e = new Ellipse2D.Double(150, 50, 90, 90);
        //创建一个 BasicStroke 对象来设置椭圆的绘制方式
        stroke = new BasicStroke(8, BasicStroke.CAP_BUTT,
            BasicStroke.JOIN_BEVEL, 0, new float[] { 10,5 }, 0);
        g2.setStroke(stroke);
        g2.draw(e);
        Rectangle2D r = new Rectangle2D.Double(30, 100, 80, 80);
        //创建一个 BasicStroke 对象来设置矩形的绘制方式
        stroke = new BasicStroke(10, BasicStroke.CAP_SQUARE,
            BasicStroke.JOIN_ROUND, 0);
        g2.setStroke(stroke);
        g2.draw(r);
    }
    public static void main(String[] args) {
        StrokeDemo s = new StrokeDemo();
        s.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

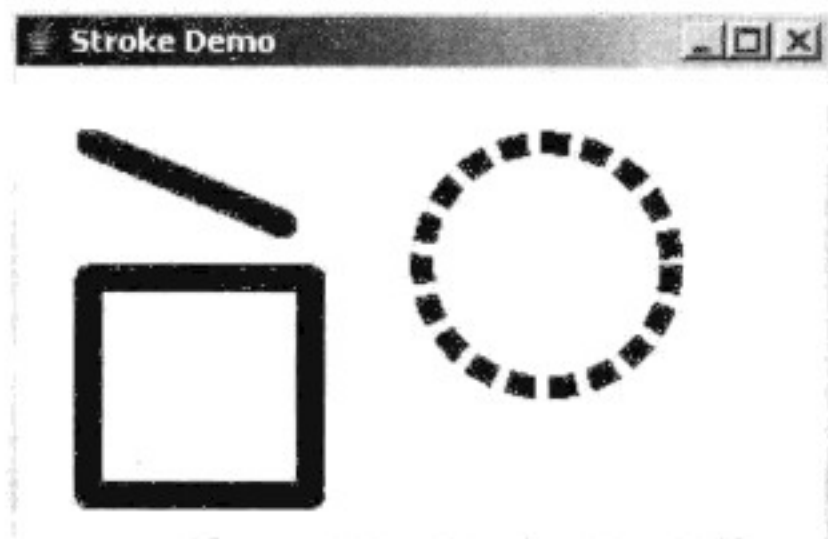


图 3-13 例 3-7 的运行结果

(3) composite 用于定义绘制重叠图形的绘制方式。当绘制多个图像时,遇到图像重叠的情况,需要确定重叠部分的颜色显示方式,重叠区域的像素颜色决定了该部分图像的透明程度。在 composite 的定义中,最常用的就是 AlphaComposite, 我们可以通过 setComposite()方法来将 AlphaComposite 对象添加到 Graphics2D 上下文中,设置图像重叠部分的复合样式。AlphaComposite 类中定义了多种新颜色与已有颜色的复合规则,例如 SRC_OVER 表示混合时新颜色(源色)应覆盖在已有颜色(目标色)之上; DST_OUT 表示混合时去除已有颜色; SRC_OUT 表示混合时去

除新颜色；DST_OVER 表示混合时用已有颜色覆盖新颜色。在设置混合颜色的同时，还可以设置颜色的透明度 alpha 值，它用百分比表示在颜色重叠时当前颜色的透明度，alpha 取值范围从 0.0（完全透明）~ 1.0（完全不透明）。

3.5.5 路径类

在 java.awt.geom 包中定义了几何图形类，包括点、直线、矩形、圆、椭圆、多边形，等等。该包中各类的层次结构如下：

```
|- java.lang.Object
  |- java.awt.geom.AffineTransform
  |- java.awt.geom.Area
  |- java.awt.geom.CubicCurve2D
    |- java.awt.geom.CubicCurve2D.Double
    |- java.awt.geom.CubicCurve2D.Float
  |- java.awt.geom.Dimension2D
  |- java.awt.geom.FlatteningPathIterator
  |- java.awt.geom.Line2D
    |- java.awt.geom.Line2D.Double
    |- java.awt.geom.Line2D.Float
  |- java.awt.geom.Path2D
    |- java.awt.geom.Path2D.Double
    |- java.awt.geom.Path2D.Float
    |- java.awt.geom.GeneralPath
  |- java.awt.geom.Point2D
    |- java.awt.geom.Point2D.Double
    |- java.awt.geom.Point2D.Float
    |- java.awt.geom.QuadCurve2D
      |- java.awt.geom.QuadCurve2D.Double
      |- java.awt.geom.QuadCurve2D.Float
  |- java.awt.geom.RectangularShape
    |- java.awt.geom.Arc2D
      |- java.awt.geom.Arc2D.Double
      |- java.awt.geom.Arc2D.Float
    |- java.awt.geom.Ellipse2D
      |- java.awt.geom.Ellipse2D.Double
      |- java.awt.geom.Ellipse2D.Float
    |- java.awt.geom.Rectangle2D
```

路径类用于构造直线、二次曲线和三次曲线的几何路径，它可以包含多个子路径。如上类层次结构所描述，Path2D 是基类（它是一个抽象类）；Path2D.Double 和 Path2D.Float 是其子类，它们分别以不同精度的坐标定义几何路径；GeneralPath 在 1.5 及其以前的版本中，是一个独立的最终类，在 1.6 版本中进行了调整与划分，其功能由 Path2D 替代，为了其兼容性，把它划为 Path2D.Float 派生的最终类。下面以 GeneralPath 类为例介绍路径类的功能与应用。

1. 构造方法

构造路径对象的方法如下。

（1）**GeneralPath(int rule)**：以 rule 指定缠绕规则构建对象。缠绕规则确定路径内部的方式。有两种方式的缠绕规则：Path2D.WIND_EVEN_ODD 用于确定路径内部的奇偶 (even-odd) 缠绕规则；Path2D.WIND_NON_ZERO 用于确定路径内部的非零 (non-zero) 缠绕规则。

（2）**GeneralPath()**：以默认的缠绕规则 Path2D.WIND_NON_ZERO 构建对象。

(3) **GeneralPath**(int rule, int initialCapacity): 以 rule 指定缠绕规则和 initialCapacity 指定的容量 (以存储路径坐标) 构建对象。

(4) **GeneralPath**(Shape s): 以 Shape 对象 s 构建对象。

2. 常用方法

路径对象常用的方法如下。

(1) void **append**(Shape s, boolean connect): 将指定 Shape 对象的几何形状追加到路径中, 也许会使用一条线段将新几何形状连接到现有的路径段。如果 connect 为 True 并且路径非空, 则被追加的 Shape 几何形状的初始 moveTo 操作将被转换为 lineTo 操作。

(2) void **closePath**(): 回到初始点使之形成闭合的路径。

(3) boolean **contains**(double x, double y): 测试指定坐标是否在当前绘制的边界内。

(4) void **curveTo**(float x1, float y1, float x2, float y2, float x3, float y3): 将 3 个新点定义的曲线段添加到路径中。

(5) Rectangle2D **getBounds2D**(): 获得路径的边界框。

(6) Point2D **getCurrentPoint**(): 获得当前添加到路径的坐标。

(7) int **getWindingRule**(): 获得缠绕规则。

(8) void **lineTo**(float x, float y): 绘制一条从当前坐标到 (x, y) 指定坐标的直线, 将 (x, y) 坐标添加到路径中。

(9) void **moveTo**(float x, float y): 从当前坐标位置移动到 (x, y) 指定位置, 将 (x, y) 添加到路径中。

(10) void **quadTo**(float x1, float y1, float x2, float y2): 将两个新点定义的曲线段添加到路径中。

(11) void **reset**(): 将路径重置为空。

(12) void **setWindingRule**(int rule): 设置缠绕规则。

(13) void **transform**(AffineTransform at): 使用指定的 AffineTransform 变换此路径的几何形状。

【例 3-8】 使用 GeneralPath 类绘制一个正三角形和一个倒三角形, 运行结果如图 3-14 所示。

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.JFrame;
public class AddPathExample extends JFrame{
    public AddPathExample() {
        super();
        setTitle("GeneralPath Demo");
        setSize(200,200);
        setVisible(true);
    }
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        GeneralPath myPath, myPath2;
        //myArray 包含正三角形的所有顶点
        Point[] myArray =
        {
            new Point(130,130), new Point(160,160),
            new Point(100,160), new Point(130,130)
        };
        myPath = new GeneralPath();
        myPath.moveTo(myArray[0].x, myArray[0].y);
```

//建立 GeneralPath 对象
//起始顶点

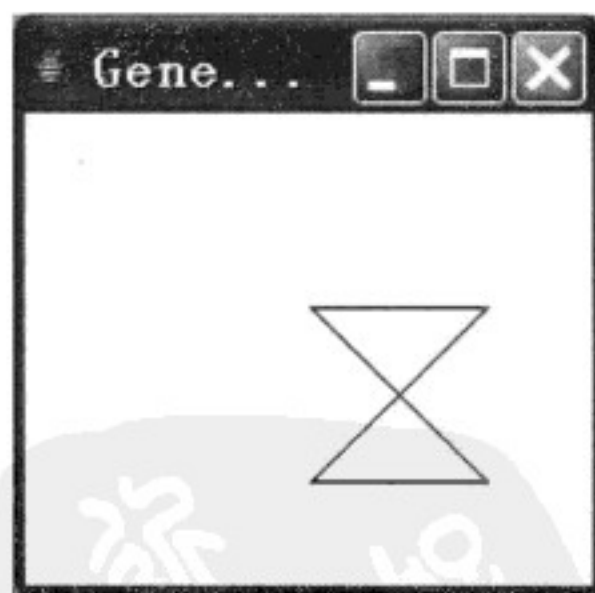


图 3-14 例 3-8 的运行结果


```

        for(int i=1; i<myArray.length; i++)                //创建正三角形的路径
            myPath.lineTo(myArray[i].x, myArray[i].y);      //绘制直线
        g2.draw(myPath);    //画正三角形
        //myArray2 包含倒三角形的所有顶点
        Point[] myArray2 =
        {
            new Point(130,130),    new Point(100,100),
            new Point(160,100),    new Point(130,130)
        };
        myPath2 = new GeneralPath();                        //建立 GeneralPath 对象
        myPath2.moveTo(myArray2[0].x, myArray2[0].y);        //起始顶点
        for(int i=1; i<myArray2.length; i++)                //创建倒三角形的路径
            myPath2.lineTo(myArray2[i].x, myArray2[i].y);    //绘制直线
        g2.draw(myPath2);    //画倒三角形
    }
    public static void main(String[] args) {
        AddPathExample gp = new AddPathExample();
        gp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

GeneralPath 还可以包含多个图形,其中每一个图形被称为子路径(figures)。如果在 GeneralPath 对象中使用了 closePath 函数,在这之后对路径所添加的线条都将构成一个新的子路径。

3.5.6 平移、缩放或旋转图形

需要平移、缩放或旋转一个图形,可以使用 AffineTransform 类来实现对图形的这些操作。

(1) 首先使用 AffineTransform 类创建一个对象:

```
AffineTransform trans=new AffineTransform();
```

对象 trans 具有最常用的 3 个方法来实现对图形变换的操作。

① translate(double a,double b): 将图形在 x 轴方向移动 a 个单位像素, y 轴方向移动 b 个单位像素。a 是正值时向右移动,负值是向左移动; b 是正值时是向下移动,负值是向上移动。

② scale(double a,double b): 将图形在 x 轴方向缩放 a 倍, y 轴方向缩放 b 倍。

③ rotate(double number,double x,double y): 将图形沿顺时针或逆时针以 (x, y) 为轴点旋转 number 个弧度。

(2) 进行需要的变换,比如要把一个矩形绕点 (100,100) 顺时针旋转 60°,那么就要先做好准备:

```
trans.rotate(60.0*3.1415927/180,100,100);
```

(3) 把 Graphics 对象,比如 g_2d 设置为具有 trans 这种功能的“画笔”:

```
g_2d.setTransform(trans);
```

假如 rect 是一个矩形对象,那么 g_2d.draw(rect)画的就是旋转后的矩形。

第 4 章

Java 游戏程序的基本框架

游戏开发是艺术与科学的结合，是策划、美术、程序三者间的协调以及创意与商业的平衡，集故事、音乐、动画等多元素于一身。无论是 2D 还是 3D 的动画或者程序设计，不仅要做出动画，更要做出互动的游戏。一些剧情游戏，游戏是根据互动的情况而有不同的故事结局，这些都可以点燃一名游戏开发人员的创作激情。目前 Java 不仅可以开发台式机游戏，而且随着智能手机的普及，目前支持 Java 的手机性能上已经接近第二代控制台游戏机，因此手机游戏大规模使用的时代已经来临，可见 Java 游戏开发市场是如此之广大。

4.1 动画的类型及帧频

动画的制作是游戏设计的基础，几乎所有的游戏都是在动画的基础上添加人机交互功能，以及增加剧情功能等延伸出来的。因此动画的制作是游戏开发人员必须了解的知识，也是游戏制作的基本元素。

4.1.1 动画类型

动画主要分为影视动画和游戏动画两种。影视动画就是使用专业动画软件编辑出来的效果很好的动画视频，当投影机以每秒 24 格的速度投射在银幕上，或者录像机以每秒 30 格的扫描方式在电视荧光屏上呈现影像时，它会把每格不同的画面连接起来，从而在我们脑海中产生物体在“运动”的印象，这就是“视觉暂留”现象。

游戏动画则不同于影视动画，是在屏幕上显示一系列连续动画画面的第一帧图形，然后在每隔很短的时间显示下一帧图像，如此反复，利用人眼的视觉暂留现象而感觉好像画面的物体在运动。显示的图形不一定是图片，也可能是其他绘图元素，例如正方体等。同时由于游戏动画的特殊性，一个游戏人物的动画往往只有几个简单的动作，因此可以循环绘制这几个简单的人物动画图片以达到动画效果。而游戏动画的背景则可以由很多相同的小图片以贴图的形式表现出来。

4.1.2 设置合理的帧频

FPS 顾名思义，就是每秒钟的帧数。每一帧就是一幅静态图像，电影的播放速度是 24FPS，但是通常游戏速度达到 10FPS 就能明显感觉到动画的效果了。由于屏幕上显示的图像越大，占用的内存越多，处理的速度就越慢，尤其是哪些需要大量动画的游戏，因此如果想使用较高的 FPS，就必须在显示大小上面做出牺牲。目前 PC 游戏往往分为 640×480 、 1024×768 等多种分辨率就是这个

道理。设计一款良好的手机游戏，要充分考虑到设备本身的限制，在游戏中设置好最佳的 FPS。

4.2 游戏动画的制作

4.2.1 绘制动画以及动画循环

既然动画就是将一连串的图像快速地循环播放，因此就需要使用循环语句控制图像连续播放。因为动画需要一定的播放速度，所以需要连续播放动画的同时能够控制动画的播放速度，最好使用线程中的暂停函数来实现。

无论你希望动画播放几次都可以通过一个循环语句来实现，例如希望动画无限制地播放的方法如下所示：

```
while(true){
    处理游戏功能;
    使用 repaint() 函数要求重画屏幕
    暂停一小段时间; //帧频 FPS 控制
}
```

在 Java 游戏程序中，通过 `repaint()` 函数请求屏幕的重画，可以请求重画全部屏幕，也可以请求重画部分屏幕。

下面举一个简单的例子，详细讲解如何使用线程（线程技术详见第 6 章）和循环实现一个简单的自由落体小球动画。由于自由落体的基本动画就是从屏幕的顶端往下自由降落小球，一个复杂的动画总是由一些基本的元素组成的，因此实现自由落体动画，首先设计一个自由降落的小球，同时控制降落的速度。控制速度就需要实现一个继承了 `Runnable` 线程接口和继承了 `JPanel` 类的 `TetrisPanel` 面板类。继承 `JPanel` 类是为了使用 `JPanel` 的 `Paint()` 方法来实现小球在屏幕上的绘制，继承 `Runnable` 线程接口可以实现动画的暂停控制。

`TetrisPanel` 类程序的整体结构如下所示：

```
class TetrisPanel extends JPanel implements Runnable {
    public TetrisPanel () {
    }
    public void paint(Graphics g) {
    }
    public void run() {
    }
}
```

在 `TetrisPanel()` 构造方法函数中创建线程，并启动该线程。一旦做好这些准备工作以后，当 `TetrisPanel` 对象第一次被显示时，就会创建线程对象的一个实例，并把“this”对象作为建构方法的参数，相当于开启一个本地的线程，之后就可以启动动画了。

```
public TetrisPanel()
{
    //创建一个新线程
    Thread t = new Thread(this);
    //启动线程
    t.start();
}
```


现在来实现线程的 `run()` 方法，它使用无限循环 `while (true)` 语句中每隔 30ms 重画动画场景。`Thread.sleep()` 这个方法很重要，如果在 `run()` 方法的循环语句中没有这部分，小球的重画动作将执行得很快，其他一些功能则不能完全执行，屏幕上完全看不出动画的效果，也即在屏幕上将看不到小球的显示。

```
public void run()//重载 run() 方法
{
    while(true)//线程中的无限循环
    {
        try{
            Thread.sleep(30); //线程休眠 30ms
        }catch(InterruptedException e){}
        ypos+=5; //修改小球左上角的纵坐标
        if(ypos>300) //小球离开窗口后重设左上角的纵坐标
            ypos=-80;
        repaint();//窗口重绘
    }
}
```

小球的重画将在 `paint()` 方法中实现，通过不断更改小球要显示的 `y` 坐标，同时清除上一次显示的小球，就可以看到小球的自由落体动画。由于是演示程序，并没有实现局部清除上一次显示的小球的方法，使用了清除全屏的简单方法，让读者把重点放在动画控制的程序流程中。具体代码如下所示：

```
public void paint(Graphics g) //重载绘图方法
{
    //super.paint(g);将面板上原来画的东西擦掉
    g.setColor(Color.RED); //设置小球颜色
    g.fillOval(90,ypos,80,80); //绘制小球
}
```

小球的下落则通过不断改变小球的显示 `y` 坐标达到目的。`TetrisPanel` 类完整的代码如例 4-1 所示：

【例 4-1】 TetrisPanel.java

```
import java.awt.*;
import javax.swing.JPanel;
public class TetrisPanel extends JPanel implements Runnable { //绘图线程类
    public int ypos=-80; //小球左上角的纵坐标
    //在类中添加如下两个私有成员
    private Image iBuffer;
    private Graphics gBuffer;
    public TetrisPanel()
    {
        //创建一个新线程
        Thread t = new Thread(this);
        //启动线程
        t.start();
    }
    public void run()//重载 run() 方法
    {
```



```

        while(true)//线程中的无限循环
        {
            try{
                Thread.sleep(30); //线程休眠 30ms
            }catch(InterruptedException e){}
            ypos+=5; //修改小球左上角的纵坐标
            if(ypos>300) //小球离开窗口后重设左上角的纵坐标
                ypos=-80;
            repaint();//窗口重绘
        }
    }
    public void paint(Graphics g) //重载绘图方法
    {
        super.paint(g);将面板上原来画的东西擦掉
        //先清屏幕，否则原来画的东西仍在
        g.clearRect(0, 0, this.getWidth(), this.getHeight());
        g.setColor(Color.RED); //设置小球颜色
        g.fillOval(90,ypos,80,80); //绘制小球
    }
}

```

通过不断改变坐标系统的 y 坐标 $ypos$ 达到红色小球向下移动的效果。屏幕的每次刷新都会根据改变后的 y 坐标 $ypos$ 重新绘制红色小球。

运行 TetrisPanel 类的主程序并没有添加其他的控制，仅仅为动画屏幕添加了一个窗口关闭处理方法，用来关闭程序。例 4-1 主程序的具体代码如下所示：

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MyWindow extends JFrame
{
    MyWindow()
    {
        //设置窗口标题
        this.setTitle("这是测试窗口");
        Container c=this.getContentPane();//获取面板容器
        c.add(new TetrisPanel());
        //设置窗口开始显示时距离屏幕左边 400 个像素点
        //距离屏幕上边 200 个像素点
        //窗口宽 300 个像素点，窗口高 300 个像素点
        this.setBounds(400,200,300,300);
        //设置窗口关闭按钮具有关闭整个程序的功能
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false); //设置窗口大小不会改变
        this.setVisible(true); //显示该窗口
    }
    public static void main(String args[])
    {
        //创建该窗口的实例 DB，开始整个程序
        MyWindow DB=new MyWindow(); //创建主类的对象
    }
}

```



```
DB.addWindowListener(new WindowAdapter()//添加窗口关闭处理方法
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
}
```

程序运行效果如图 4-1 所示，一个红色小球从上往下以一定的速度慢慢掉下来，当黑色小球移动到屏幕底端的时候，将循环从上开始继续下落，只要不关闭程序，将会无限制地重复下落的动画。

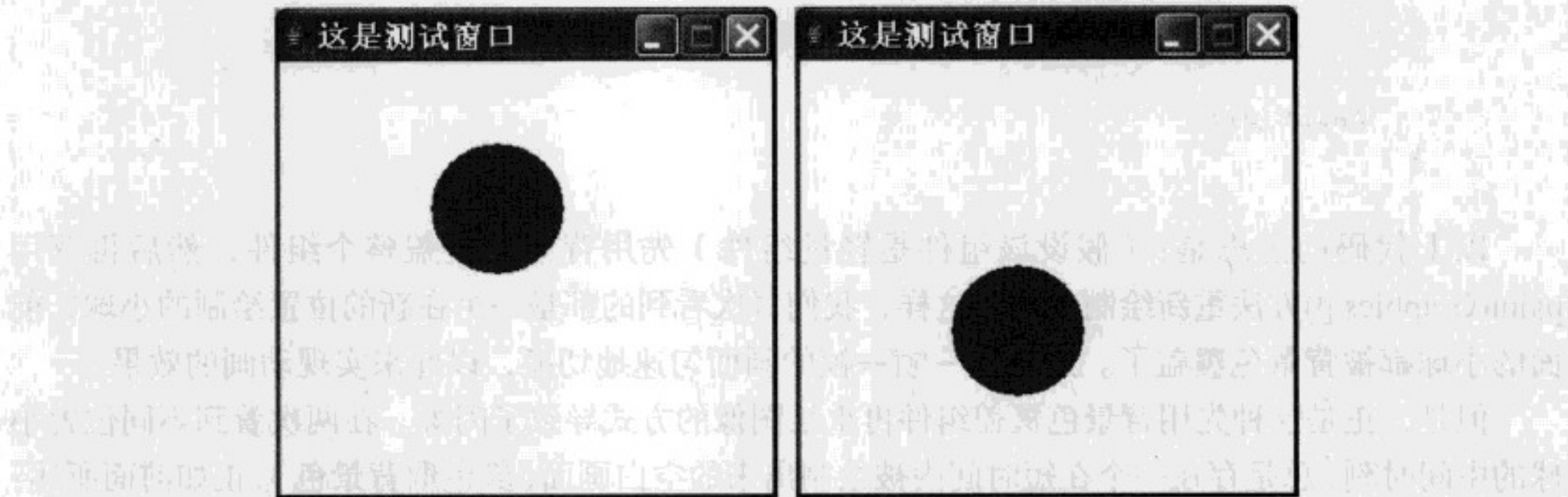


图 4-1 例 4-1 的运行结果

4.2.2 消除动画闪烁现象——双缓冲技术

一个动画在运行的时候，如果图像的切换是在屏幕上完成的，则可能会造成屏幕的闪烁，消除动画闪烁现象的最佳方法是使用双缓冲技术。

双缓冲技术在屏幕外做一个图像缓冲区，事先在这个缓冲区内绘制图像，然后再将这个图像送到屏幕上，虽然动画中的图像切换很频繁，但是双缓冲技术很好地避免了在屏幕上进行消除和刷新时候的处理工作所带来的屏幕闪烁情况。但是在屏幕外的缓冲区需要占用一部分的内存资源，特别是图像比较大的时候，内存占用非常严重，因此一般需要考虑动画的质量和运行速度之间的重要性，有选择性地进行开发。

1. 屏幕产生闪烁的原因

在 Java 游戏编程和动画编程中最常见的就是对于屏幕闪烁的处理。屏幕产生闪烁的原因是先用背景色覆盖组件再重绘图像的方式造成的。即使时间很短，如果重绘的面积较大则花费的时间也是比较可观的，这个时间甚至可以大到足以让闪烁严重到让人无法忍受的地步。就像以前课堂上老师用的旧式幻灯机，放完一张胶片，老师会将它拿下来，这个时候屏幕上一片空白，直到放上第二张，中间时间间隔较长。当然，这不是在放动画，但上述闪烁的产生原因与这很类似。

运行图 4-1 简单的小球下落动画程序后，我们会看到窗体中有一个从上至下匀速运动的小球，但仔细观察，你会发现小球会不时地被白色的不规则横纹隔开，即所谓的屏幕闪烁，这不是我们预期的结果。

这种闪烁是如何出现的呢？

首先我们分析一下这段代码。MyWindow 类的对象 DB 建立后，将显示窗口，程序首先自动调用重载后的 `paint(Graphics g)` 方法，在窗口上绘制了一个小球，绘图线程启动后，该线程每隔 30ms 修改一下小球的位置，然后调用 `repaint()` 方法。

注意，这个 `repaint()` 方法并不是我们重载的，而是从 `JPanel` 类继承而来的。它先调用 `update(Graphics g)` 方法，`update(Graphics g)` 方法再调用 `paint(Graphics g)` 方法。问题就出在 `update(Graphics g)` 方法上，我们来看看这个方法的源代码：

```
public void update(Graphics g)
{
    if (isShowing())
    {
        if (! (peer instanceof LightweightPeer))
        {
            g.clearRect(0, 0, width, height);
        }
        paint(g);
    }
}
```

以上代码的意思是：（假设该组件是轻量组件）先用背景色覆盖整个组件，然后再调用 `paint(Graphics g)` 方法重新绘制小球。这样，我们每次看到的都是一个在新的位置绘制的小球，前面的小球都被背景色覆盖了。这就像一帧一帧的画面匀速地切换，以此来实现动画的效果。

但是，正是这种先用背景色覆盖组件再重绘图像的方式导致了闪烁。在两次看到不同位置小球的中间时刻，总是存在一个在短时间内被绘制出来的空白画面（颜色取背景色）。正如前面所述：即使时间很短，如果重绘的面积较大则花费的时间也是比较可观的，这个时间甚至可以大到足以让闪烁严重到让人无法忍受的地步。

另外，用 `paint(Graphics g)` 方法在屏幕上直接绘图的时候，由于执行的语句比较多，程序不断地改变窗体中正在被绘制的图像，会造成绘制的缓慢，这也从一定程度上加剧了闪烁。知道了闪烁产生的原因，我们就有了更具针对性的解决闪烁的方案。

2. 双缓冲技术

所谓双缓冲，就是在内存中开辟一片区域，作为后台图像，程序对它进行更新、修改，绘制完成后再显示到屏幕上。

双缓冲技术的工作原理：先在内存中分配一个与我们动画窗口一样大的空间（在内存中的空间我们是看不到的），然后利用 `getGraphics()` 方法来获得双缓冲画笔，接着利用双缓冲画笔给空间描绘出我们想画的东西，最后将它全部一次性地显示到屏幕上。这样在咱们的动画窗口上面显示出来就非常流畅了，避免了上面的闪烁效果。

3. 双缓冲的使用

一般采用重载 `paint(Graphics g)` 方法实现双缓冲。这种方法要求我们将双缓冲的处理放在 `paint(Graphics g)` 方法中，那么具体该怎么实现呢？先看下面的代码（基于前面代码段修改）：

在 `TetrisPanel` 类中添加如下两个私有成员：

```
private Image iBuffer;
private Graphics gBuffer;
```

重载 `paint(Graphics g)` 方法：

```
public void paint(Graphics g)
{
    if (iBuffer==null)
```



```

{
    iBuffer=createImage(this.getSize().width,this.getSize().height);
    gBuffer=iBuffer.getGraphics();
}
gBuffer.setColor(getBackground());
gBuffer.fillRect(0,0,this.getSize().width,this.getSize().height);
gBuffer.setColor(Color.RED);
gBuffer.fillOval(90,ypos,80,80);
再将此缓冲图像一次性绘制到代表屏幕的 Graphics 对象上, 即该方法传入的 g 上
g.drawImage(iBuffer,0,0,this);
}

```

分析上述代码: 我们首先添加了两个成员变量 `iBuffer` 和 `gBuffer` 作为缓冲 (这就是所谓的双缓冲名称的来历)。在 `paint(Graphics g)` 方法中, 首先检测如果 `iBuffer` 为 `null`, 则创建一个与屏幕上的绘图区域大小一样的缓冲图像; 再取得 `iBuffer` 的 `Graphics` 类型的对象的引用, 并将其赋值给 `gBuffer`; 然后对 `gBuffer` 这个内存中的后台图像先用 `fillRect(int,int,int,int)` 清屏, 再进行绘制操作, 完成后将 `iBuffer` 直接绘制到屏幕上。

这段代码看似可以完美地完成双缓冲, 但是运行之后我们看到的还是严重的闪烁。什么原因呢? 问题还是出现在 `update(Graphics g)` 方法上。这段修改后的程序中的 `update(Graphics g)` 方法还是我们从父类继承的。在 `update(Graphics g)` 方法中, `clearRect(int,int,int,int)` 对前端屏幕进行了清屏操作, 而在 `paint(Graphics g)` 方法中, 对后台图像又进行了清屏操作。那么如果保留后台清屏, 去掉多余的前台清屏应该就会消除闪烁。因此, 我们只要重载 `update(Graphics g)` 方法即可:

```

public void update(Graphics g)
{
    paint(g);
}

```

这样就避开了对前端图像的清屏操作, 避免了屏幕的闪烁。

运行上述修改后的程序, 我们会看到完美的消除闪烁后的动画效果。就像在电影院看电影, 每张胶片都是在后台准备好的, 播放完一张胶片之后, 下一张很快就被播放到前台, 自然不会出现闪烁的情形。

为了让读者能对双缓冲技术有个全面的认识, 现将上述双缓冲的实现原理概括如下。

(1) 定义一个 `Graphics` 对象 `gBuffer` 和一个 `Image` 对象 `iBuffer`。按屏幕大小建立一个缓冲对象给 `iBuffer`。然后取得 `iBuffer` 的 `Graphics` 赋给 `gBuffer`。此处可以把 `gBuffer` 理解为逻辑上的缓冲屏幕, 而把 `iBuffer` 理解为缓冲屏幕上的图像。

(2) 在 `gBuffer` (逻辑上的屏幕) 上绘制图像。

(3) 将后台图像 `iBuffer` 全部一次性地绘制到我们的前台窗口。

以上就是一次双缓冲的过程。注意, 将这个过程联系起来的是 `repaint()` 方法。`paint(Graphics g)` 方法是一个系统调用语句, 不能由程序员手工调用, 只能通过 `repaint()` 方法调用。

上文提到的双缓冲的实现方法只是消除闪烁的方法中的一种。如果在 `Swing` 中, 组件本身就提供了双缓冲的功能, 我们只需要进行简单的方法调用就可以实现组件的双缓冲, 在 `awt` 中却没有提供此功能。另外, 一些硬件设备也可以实现双缓冲, 每次都是先把图像画在缓冲中, 然后再绘制在屏幕上, 而不是直接绘制在屏幕上, 基本原理与文中介绍的类似。还有其他用软件实现消除闪烁的方法, 但双缓冲是个简单的、值得推荐的方法。

4. 关于双缓冲的补充

双缓冲技术是编写 Java 游戏的关键技术之一。双缓冲付出的代价是较大的额外内存消耗, 但

现在节省内存已经不再是程序员考虑的最首要的问题了,游戏的画面在游戏制作中是至关重要的,因此以额外的内存消耗换取程序质量的提高还是值得肯定的。

有时动画中相邻的两幅画面只是有很少部分的不同,这就没必要每次对整个绘图区进行清屏。我们可以对文中的程序进行修改,使之每次只对部分屏幕清屏,这样既能节省内存,又能减少绘制图像的时间,使动画更加连贯。

4.3 使用定时器

定时器在游戏开发中是相当重要的,前面提到动画的实现就是通过显示时间的控制达到视觉暂留的效果,除了使用线程的暂停函数 `sleep()` 外,还有一个重要的计时工具,那就是 `Timer` 组件。

`Timer` 组件可以定时执行任务,这在游戏动画编程上非常有用。`Timer` 组件可以用 `javax.swing.Timer` 包中的 `Timer` 类来实现,该类的构造方法为:

```
Timer(int delay, ActionListener listener);
```

该构造方法用于建立一个 `Timer` 组件对象,参数 `listener` 用于指定一个接收该计时器操作事件的侦听器,指定所要触发的事件;而参数 `delay` 用于指定每一次触发事件的时间间隔。也就是说, `Timer` 组件会根据用户所指定的 `delay` 时间,周期性地触发 `ActionEvent` 事件。如果要处理这个事件,就必须实现 `ActionListener` 接口类,以及接口类中的 `actionPerformed()` 方法。

`Timer` 组件类中的主要方法如下。

- `void start()`: 激活 `Timer` 组件对象。
- `void stop()`: 停止 `Timer` 组件对象。
- `void restart()`: 重新激活 `Timer` 组件对象。

在游戏编程中,在组件内容更新时经常用到 `Timer` 组件,例如 `JPanel`、`JLabel` 等内容的更新。本书中俄罗斯方块游戏就采用定时器 `Timer` 实现控制方块的下落。

【例 4-2】 下面是一个简单的每隔 500ms 显示时间的 swing 程序,可以帮助读者加深对 `Timer` 组件使用的理解。

```
TimerTest.java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.*;
/**
 * 测试 swing 中 Timer 的使用, 一个显示时间的 GUI 程序
 */
public class TimerTest extends JFrame implements ActionListener {
    //一个显示时间的 JLabel
    private JLabel jlTime = new JLabel();
    private Timer timer;
    public TimerTest() {
        setTitle("Timer 测试");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(180, 80);
        add(jlTime);
```



```

        //设置 Timer 定时器并启动
        timer = new Timer(500, this);
        timer.start();
        setVisible(true);
    }
    /**
     * Timer 要执行的部分
     */
    public void actionPerformed(ActionEvent e) {
        DateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date date = new Date();
        jlTime.setText(format.format(date));
    }
    public static void main(String[] args) {
        new TimerTest();
    }
}

```

TimerTest 类实现了 ActionListener 接口，因此可以直接设置 timer = new Timer (500,this);使用 this 初始化计时器。

当计时器启动后 (timer.start()执行后)，每隔 500ms 执行一次实现的 ActionListener 接口中的 actionPerformed 的方法体。

4.4 设置游戏难度

一般来说，一款游戏要增强它的可玩性，就需要有合理的游戏难度，使玩家不容易感到厌烦，同时增加玩家的挑战欲望。例如，智力游戏可以在人工智能方面下工夫，但是由于人工智能在游戏里面很难真正地控制游戏本身的难度，因此往往通过其他手段实现游戏的难度控制，比如增加游戏进行的速度。例如，在俄罗斯方块游戏中，可以增加方块下落的速度，或者使用增加游戏关数、减少玩家的思考时间等。难度设置的开发需要根据具体的游戏情况而定，很难一概而论。

如果使用速度控制游戏难度，则可以把游戏设计成为具有很多个级别，每个级别游戏的运行速度都不一样，类似的代码如下所示：

```

public void level(){
    if(level ==3)
        speed = 1;//设置游戏速度为 1
    else if(level == 4)
        speed = 2;//设置游戏速度为 2
}

```

4.5 游戏与玩家的交互

对于游戏而言，交互性就是生命，优异的作品总是体现在人物与场景之间的高互动性、人物与人物（NPC）之间的高互动性以及玩家与玩家之间（多人模式）的高互动性上，这是真正能让玩家融入其中的动力所在，因此一款好的游戏必然拥有一个良好的游戏与玩家之间的互动方式。

游戏与玩家的交互都是通过键盘或者鼠标实现的，具体的方法在第 2 章中已经详细介绍了。

下面举一个简单的例子，在小球下落时可以通过键盘控制其左右移动。由于需要实现键盘监听事件，因此 TetrisPanel 加入了 KeyListener 接口，同时为控制 x 坐标增加了 xpos 变量。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.JPanel;

public class TetrisPanel extends JPanel implements Runnable, KeyListener {
    public int ypos=-80, xpos=90; //小球左上角的坐标
    public TetrisPanel()
    {
        Thread t = new Thread(this); //创建一个新线程
        t.start(); //启动线程
        //设定焦点在本面板并作为监听对象
        setFocusable(true);
        addKeyListener(this);
    }
    public void run()//重载 run()方法
    {
        while(true)//线程中的无限循环
        {
            try{
                Thread.sleep(30); //线程休眠 30ms
            }catch(InterruptedException e){}
            ypos+=5; //修改小球左上角的纵坐标
            if(ypos>300) //小球离开窗口后重设左上角的纵坐标
                ypos=-80;
            repaint();//窗口重绘
        }
    }
    public void paint(Graphics g) //重载绘图方法
    {
        //super.paint(g);将面板上原来画的东西擦掉
        //先清屏幕，否则原来画的东西仍在
        g.clearRect(0, 0, this.getWidth(), this.getHeight());
        g.setColor(Color.RED); //设置小球颜色
        g.fillOval(xpos,ypos,80,80); //绘制小球
    }
    public void keyPressed(KeyEvent e) {
        int keyCode = e.getKeyCode(); //获得按键编号
        switch (keyCode) { //通过 keyCode 识别用户的按键
            case KeyEvent.VK_LEFT : //当触发 Left 时
                xpos-=10;
                break;
            case KeyEvent.VK_RIGHT : //当触发 Right 时
                xpos+=10;
                break;
        }
        repaint();//重新绘制窗体图像
    }
    public void keyReleased(KeyEvent arg0) {
```



```

    }
    public void keyTyped(KeyEvent arg0) {
    }
}

```

这样在游戏过程中，玩家通过键盘就可以控制小球左右移动了。

4.6 游戏中的碰撞检测

我们在游戏开发中总会遇到这样那样的碰撞，并且会很频繁地去处理这些碰撞，这也是游戏开发中的一种基本算法。常见的碰撞算法是矩形碰撞、圆形碰撞和像素碰撞，矩形碰撞用得最多。

4.6.1 矩形碰撞

假如把游戏中的角色统称为一个一个的 Actor，并且把每个 Actor 框成一个与角色大小相等的矩形框，那么在游戏过程中的每次循环检查就是围绕每个 Actor 的矩形框之间是否发生了交错。为了简单起见，我们就拿一个主角与一个 Actor 来分析，其他情况与此类似。

一个主角与一个 Actor 的碰撞其实就成了两个矩形的检测，是否发生了交集。

第1种方法：

我们可以通过检测一个矩形的4个顶点是否在另外一个矩形的内部来完成。下面简单地设定一个 Actor 类：

```

public class Actor {
    int x,y,w,h;
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
    public int getActorWidth() {
        return w;
    }
    public int getActorHeight() {
        return h;
    }
}

```

检测的处理为：

```

public boolean isCollidingWith (int px ,int py){
    if(px > getX() && px < getX() + getActorWidth()
    && py > getY() && py < getY() + getActorHeight())
        return true;
    else
        return false;
}

public boolean isCollidingWith(Actor another) {
    if(isCollidingWith(another.getX(),another.getY())
    ||isCollidingWith(another.getX()+ another.getActorWidth(),another.getY())
    ||isCollidingWith(another.getX(),another.getY()+another.getActorHeight())
    ||isCollidingWith(another.getX()+
        another.getActorWidth(),another.getY()+another.getActorHeight())

```



```

    )
    return true;
else
    return false;
}

```

以上处理运行应该是没有什么问题的，但是没有考虑到运行速度，而游戏中需要大量的碰撞检测，因此要求碰撞检测要尽量地快。

第 2 种方法：

我们从相反的角度考虑，以前是处理什么时候相交，现在我们处理什么时候不会相交。可以处理 4 条边，左边 a 矩形的右边界在 b 矩形的左边界以外，同理，a 的上边界需要在 b 的下边界以外，4 条边都判断，则可以知道 a 是否与 b 相交。示意图如图 4-2 所示。



图 4-2 矩形检查

代码如下：

```

/**
 * ax : a 矩形左上角 x 坐标
 * ay : a 矩形左上角 y 坐标
 * aw : a 矩形宽度
 * ah : a 矩形高度
 * bx : b 矩形左上角 x 坐标
 * by : b 矩形左上角 y 坐标
 * bw : b 矩形宽度
 * bh : b 矩形高度
 */
public boolean isColliding(int ax,int ay,int aw,int ah, int bx, int by, int bw, int
bh)
{
    if(ay > by + bh || by > ay + ah
        || ax > bx + bw || bx > ax + aw)
        return false;
    else
        return true;
}

```

此方法比第 1 种方法简单且运行快。

第 3 种方法：

这种方法其实可以说是第 2 种方法的一个变异，我们可以保存两个矩形的左上和右下两个坐标的坐标值，然后对两个坐标的一个对比就可以得出两个矩形是否相交。这种方法应该比第 2 种方法更优越一点。

```

/*
 * rect1[0]: 矩形 1 左上角 x 坐标

```



```
* rect1[1]: 矩形 1 左上角 y 坐标
* rect1[2]: 矩形 1 右下角 x 坐标
* rect1[3]: 矩形 1 右上角 y 坐标
* rect2[0]: 矩形 2 左上角 x 坐标
* rect2[1]: 矩形 2 左上角 y 坐标
* rect2[2]: 矩形 2 右下角 x 坐标
* rect2[3]: 矩形 2 右上角 y 坐标
*/
static boolean IsRectCrossing (int rect1[], int rect2[])
{
    if (rect1[0] > rect2[2]) return false;
    if (rect1[2] < rect2[0]) return false;
    if (rect1[1] > rect2[3]) return false;
    if (rect1[3] < rect2[1]) return false;
    return true;
}
```

这种方法的速度应该很快了，推荐使用这种方法。

4.6.2 圆形碰撞

下面介绍一种测试两个对象边界是否重叠的方法。可以通过比较两个对象间的距离和两个对象半径的和的大小，很快实现这种检测。如果它们之间的距离小于半径的和，就说明产生了碰撞。为了计算半径，可以取高度或者宽度的一半作为半径的值。

代码如下：

```
public static boolean isColliding(int ax,int ay,int aw, int ah, int bx, int by, int
bw, int bh)
{
    int r1 = (Math.max(aw, ah)/2 + 1);
    int r2 = (Math.max(bw, bh)/2 + 1);
    int rSquard = r1 * r1;
    int anrSquard = r2* r2;
    int disX = ax - bx;
    int disY = ay - by;
    if((disX * disX) + (disY * disY) < (rSquard + anrSquard))
        return true;
    else
        return false;
}
```

这种方法类似于圆形碰撞检测，处理两个圆的碰撞时就可以用这种方法。

4.6.3 像素碰撞

由于游戏中的角色的大小往往是以一个刚好能够将其包围的矩形区域来表示的，如图 4-3 所示，虽然两个卡通人物并没有发生真正的碰撞，但是矩形碰撞检查的结果却是它们发生了碰撞。如果使用像素检查，就通常把精灵的背景颜色设置为相同的颜色而且是最后图片里面很少用到的颜色，然后碰撞检查的时候就只判断两张图片除了背景色外的其他像素区域是否发生了重叠的情况。如图 4-4 所示，虽然两张图片的矩形发生了碰撞，但是两个卡通人物并没有发生真正的碰撞，这就是像素检查的好处，但缺点是计算复杂，消耗了大量的系统资源，因此一般没有特殊要求，应尽量使用矩形检查碰撞。



图 4-3 矩形检查



图 4-4 像素检查

以上只是总结了几种简单的方法，当然在游戏中熟练的运用才是最好的。在 Java 中掌握以上几种方法就可以了，它不需要太精密的算法，当然可能有些时候需要比以上情况更复杂。例如如果一个对象速度足够得快，可能只经历一步就穿越了一个本该和它发生碰撞的对象，如果要考虑这种情况，就要根据它的运动路径来处理。还有可能碰到不同的边界发生不同行为的情况，这就要对碰撞行为进行具体的解剖，然后再做处理。

4.7 游戏中图像的绘制

Graphics 类中提供了很多绘制图形的方法，但对于复杂图形，大部分都事先利用专用的绘图软件绘制好，或者是用其他截取图像的工具（如扫描仪、视效卡等）获取图像的数据信息，再将它们按一定的格式存入图像文件。Java 程序运行时，将它装载到内存，然后在适当的时机将它显示在屏幕上。

4.7.1 图像文件的装载

Java 目前所支持的图像文件格式通常有 GIF、PNG 和 JPEG 格式（带有.gif、.jpg、.jpeg 后缀名的文件），具体描述如下。

（1）GIF（Graphics Interchange Format，图像互换格式）图像文件的数据是经过压缩的，其压缩率一般在 50% 左右。此外，在一个 GIF 文件中可以存放多幅彩色图像，如果把它们逐幅读出并显示到屏幕上，就可形成一组简单的动画。

（2）JPG 的全名是 JPEG（Joint Photographic Experts Group，联合图像专家组），它是与平台无关的格式，支持最高级别的压缩，但是这种压缩是有损耗的。

（3）PNG（Portable Network Graphic，流式网络图形）是一种位图文件存储格式。PNG 用来存储灰度图像时，灰度图像的深度可多到 16 位；存储彩色图像时，彩色图像的深度可多到 48 位，并且还可存储多到 16 位的 a 通道数据。PNG 使用无损数据压缩算法，压缩比高，生成文件的容量小。

Java 特别提供了 java.awt.Image 包来管理与图像文件有关的信息，因此执行与图像文件有关的操作时需要 import 包。java.awt.image 包提供可用于创建、操纵和观察图像的接口和类。每一个图像都用一个 java.awt.Image 对象表示。除了 Image 类外，java.awt 包还提供了其他基本的图像支持，例如 Graphics 类的 drawImage() 方法、Toolkit 对象的 getImage() 方法及 MediaTracker 类。

Toolkit 类提供了两个 getImage() 方法来加载图像。

- Image getImage(URL url)。
- Image getImage(String filename)。

Toolkit 是一个组件类，取得 Toolkit 的方法如下：

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
```

对于继承了 Frame 的类来说，可以直接使用下面的方法取得：

```
Toolkit toolkit = getToolkit();
```

下面是两个加载图片的实例：

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
```

```
Image image1 = toolkit.getImage("imageFile.gif");
```

```
Image image2 = toolkit.getImage(new URL("http://java.sun.com/graphics/people.gif"));
```

在 Java 中获取一个图像文件，可以调用 Toolkit 类提供的 getImage() 方法。但是 getImage() 方法会在调用后马上返回，如果此时马上使用由 getImage() 方法获取的 Image 对象，但这时 Image 对象并没有真正装载或者装载完成。因此，我们在使用图像文件时，使用 java.awt 包中的 MediaTracker 跟踪一个 Image 对象的装载，可以保证所有图片都加载完毕。

使用 MediaTracker 需要如下 3 个步骤。

(1) 实例化一个 MediaTracker，注意要将显示图片的 Component 对象作为参数传入。

```
MediaTracker tracker = new MediaTracker(Jpanel1);
```

(2) 将要装载的 Image 对象加入 MediaTracker。

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
```

```
Image[] pics=new Image[10];
```

```
pics[i] = toolkit.getImage("imageFile"+i+".jpg");
```

```
tracker.addImage(pics[i], 0);
```

(3) 调用 MediaTracker 的 checkAll() 方法，等待装载过程的结束。

```
tracker.checkAll(true);
```

下面讲解图片显示的方法，并通过两个实例演示显示图片和缩放的过程。

4.7.2 图像文件的显示

getImage() 方法仅仅是将图像文件装载进来，交由 Image 对象管理，而把得到的 Image 对象中的图像显示在屏幕上，则通过传递到 paint() 方法的 Graphics 对象可以很容易地显示图像。具体显示过程需要调用 Graphics 类的 drawImage() 方法，它能完成将 Image 对象中的图像显示在屏幕的特定位置上，就像显示文本一样方便。drawImage() 方法的常见调用格式如下：

```
boolean drawImage(Image img, int x, int y, ImageObserver observer)
```

```
boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)
```

```
boolean drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)
```

```
boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)
```

这里介绍常用的情况：

```
boolean drawImage(Image img, int x, int y, ImageObserver observer)
```

其中，参数 img 就是要显示的 Image 对象，参数 x 和 y 是该图像左上角的坐标值，参数 observer 则是一个 ImageObserver 接口 (interface)，它用来跟踪图像文件装载是否已经完成的情况，通常我们都将参数置为 this，即传递本对象的引用去实现这个接口。组件可以指定 this 作为图像观察者的原因是，Component 类实现了 ImageObserver 接口。当图像数据被加载时，它的实现会调用 repaint() 方法。

例如，下面的代码在组件区域的左上角 (0, 0) 以原始大小显示一幅图像：


```
g.drawImage(myImage, 0, 0, this);
```

除了将图像文件照原样输出以外,drawImage()方法的另外一种调用格式还能指定图像显示的区域大小:

```
boolean drawImage(Image img, int x,int y,int width,int height,ImageObserver observer)
```

这种格式比第一种格式多了两个参数 width 和 height,即表示图像显示的宽度和高度。如果实际图像的宽度和高度与这两个参数值不一样时,Java 系统会自动将实际图像进行缩放,以适合我们所设定的矩形区域。

下面的代码在坐标(90,0)处显示一个被缩放为宽 300 像素高 62 像素的图像:

```
g.drawImage(myImage, 90, 0, 300, 62, this);
```

有时,我们为了不使图像因缩放而变形失真,可以将原图的宽和高均按相同的比例进行缩小或放大。调用 Image 类中的两个方法就可以分别得到原图的宽度和高度,它们的调用格式如下:

```
int getWidth(ImageObserver observer)
```

```
int getHeight(ImageObserver observer)
```

与 drawImage()方法一样,我们通常用 this 作为 observer 的参数值。

【例 4-3】 下面的程序段给出了一个显示图像文件的例子,程序运行结果如图 4-5 所示。

```
ShowImage.java
import java.awt.*;
import javax.swing.JFrame;
public class ShowImage extends JFrame{
    String filename;
    public ShowImage(String filename){
        setSize(570, 350);
        setVisible(true);
        this.filename = filename;
    }
    public void paint(Graphics g){
        Image img = getToolkit().getImage(filename);//获取 Image 对象,加载图像
        int w=img.getWidth(this);//获取图像的宽度
        int h=img.getHeight(this); //获取图像的高度
        g.drawImage(img,20,80,this);//原图
        g.drawImage(img,200,80,w/2,h/2,this); //缩小一半
        g.drawImage(img,280,80,w*2,h/3,this); //宽扁图
        g.drawImage(img,500,80,w/2,h*2,this); //瘦高图
    }
    public static void main(String args[]) {
        new ShowImage("C:/test.jpg");
    }
}
```

窗体类继承自 JFrame,因此可以使用 Toolkit.getDefaultToolkit()方法来取得 Toolkit 对象,然后使用 getImage()方法取得一张本地图片文件,最后在 paint()方法中使用 Graphics 的 drawImage()方法即可显示该图像并缩放显示,运行结果如图 4-5 所示。

通过 getImage()方法取得的是 java.awt.Image 类型的对象,也可以使用 javax.imageio.ImageIO 类的 read()方法取得一幅图像,返回的是 BufferedImage 对象。

调用格式如下:

```
BufferedImage ImageIO.read(Url);
```

BufferedImage 是 Image 的子类,它描述了具有可访问图像数据缓存区的 Image,通过该类即可实现图片的缩放。

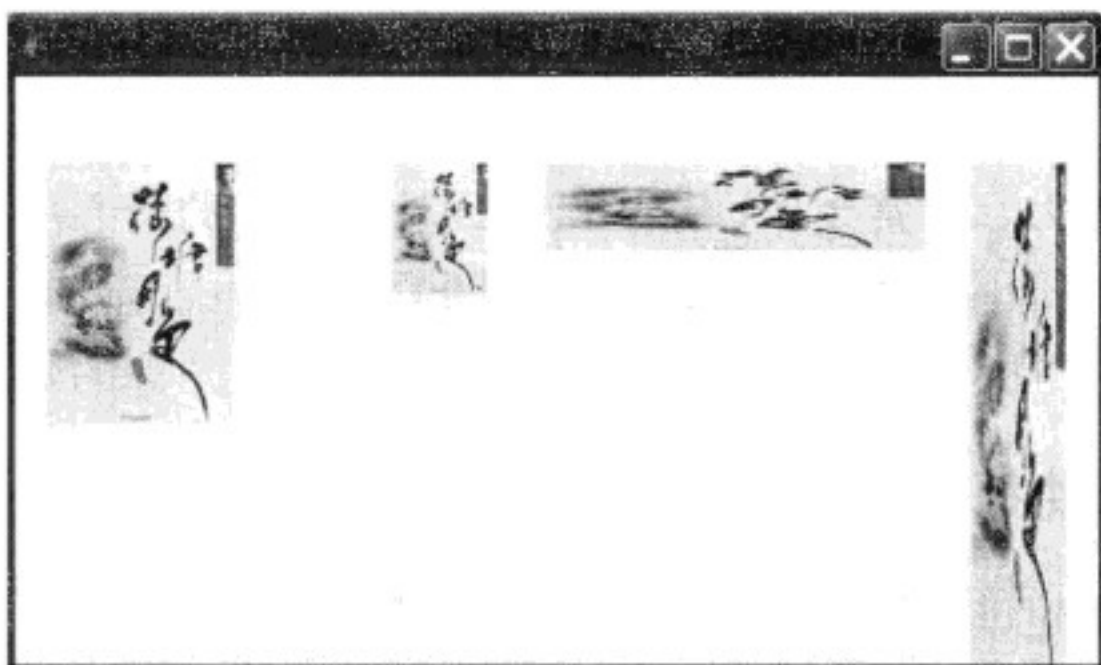


图 4-5 例 4-3 的运行效果图

例如：

```
Image im=ImageIO.read( getClass().getResource("ball.gif") );
g.drawImage(im, 0, 0, null); //图像的绘制方式
```

【例 4-4】 下面的程序首先读入一个图片文件，然后根据 Image 的 getWidth()和 getHeight()方法取得图片的宽度和高度，按照该宽度和高度的一半构造新的图片对象 BufferedImage，并将原有的图片写入该实例中，即可实现图片的缩小，最后通过 JPEG 编码保存图片。

```
ZoomImage.java
import java.io.*;
import java.awt.Image;
import java.awt.image.BufferedImage;
import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageEncoder;
public class ZoomImage {
    public void zoom(String file1, String file2) {
        try {
            //读入图片文件
            File _file = new File(file1);
            Image src = javax.imageio.ImageIO.read(_file); //构造 Image 对象
            int width = src.getWidth(null); //得到图宽
            int height = src.getHeight(null); //得到图长
            //图片缩放
            BufferedImage tag = new BufferedImage(width / 2, height / 2, BufferedImage.
TYPE_INT_RGB);
            //绘制缩小后的图片
            tag.getGraphics().drawImage(src, 0, 0, width / 2, height / 2, null);
            //写入图片
            FileOutputStream out = new FileOutputStream(file2);
            //输出到文件流，进行 JPEG 编码
            JPEGImageEncoder encoder =
                JPEGCodec.createJPEGEncoder(out); encoder.encode(tag);
            out.close();
        } catch (Exception e) {}
    }
    public static void main(String args[]) {
        String file1 = "C:/test.jpg";
        String file2 = "C:/testzoom.jpg";
        new ZoomImage().zoom(file1, file2);
    }
}
```


运行该程序，即可生成缩小后的 testzoom.jpg。

游戏中的场景图像分为两种，分别叫作卷轴型图像（ribbon）和砖块型图像（tile）。卷轴型图像的特点是内容多、面积大，常常用作远景，例如设计游戏中的蓝天白云图像，一般不与用户交互。砖块型图像通常面积很小，往往由多块这样的图像共同组成游戏的前景，并且作为障碍物与游戏角色进行交互，图 4-7 所示即为推箱子游戏中绘制砖块型图像的效果图。

4.7.3 绘制卷轴型图像

卷轴型图像通常都要超过程序窗口的尺寸，如果采用上一小节介绍的图像绘制方法来绘制卷轴型图像，则只能显示图像的一部分。

事实上，在绘制卷轴型图像时往往都需要让其滚动显示以制造移动效果，让图像的不同部分依次从程序窗口中“经过”，就如同坐火车的情形，卷轴型图像好比风景，程序窗口则好比车窗。下面具体介绍如何绘制卷轴型图像。

用程序实现这样的滚动显示效果，则需要将卷轴型图像一段一段地显示在程序窗口中，而这又涉及从图像坐标系到程序窗口坐标系的变换问题。如图 4-6 所示，左边为程序窗口坐标系，原点在窗口左上角；右边为图像坐标系，原点在卷轴型图像区域的左上角。

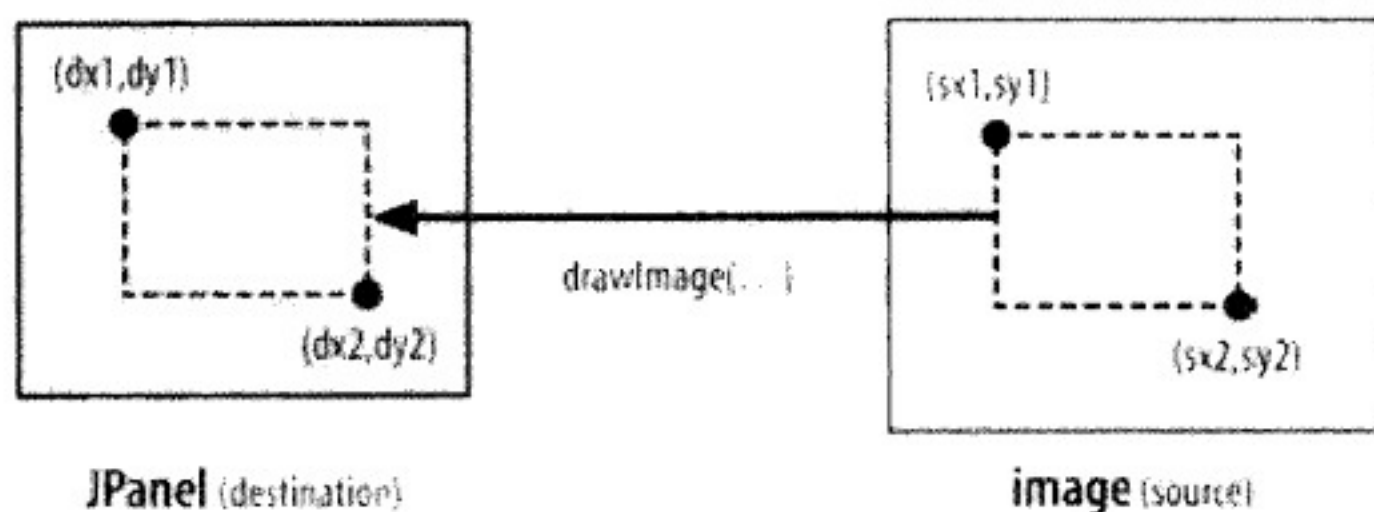


图 4-6 图像坐标系到程序窗口坐标系的变换

图中的坐标变换可通过调用程序窗口的 Graphics 对象的另一个 drawImage() 方法来实现，该方法有 10 个参数，具体定义如下：

```
drawImage(im, t dx1, f dyl, dx2, dy2, : sx1, sylr sx2, sy2, observer);
```

其中，第 1 个参数表示源图像，第 2 个至第 9 个参数的含义如图 4-6 所示。

dx1 和 dyl 为目标区域左上角坐标；dx2 和 dy2 为目标区域右下角坐标；

sx1 和 syl 为源区域左上角坐标；sx2 和 sy2 为源区域右下角坐标。

为了简化起见，这里仅讨论水平方向的场景滚动，因此 dyl 和 dy2、syl 和 sy2 的值无需改变，可依据窗口的尺寸设置为固定值。于是可以另写一个 drawRibbon() 方法来封装 drawImage() 方法，程序代码如下：

```
Private void drawRibbon(Graphics g, Image im, int dx1, int dx2, int sx1, int sx2) {
    g.drawImage (im, dx1, 0, dx2, pHeight, sx1, 0, sx2, pHeight, null); // pHeight 为窗口的高度
}
```

本书第 6 章介绍的模拟雷电的飞机射击游戏中的背景采用了类似绘制卷轴型图像，不过是采用两幅图片（不是一幅图片）循环移动出现在窗口中不停地绘制来实现的。

4.7.4 绘制砖块型图像

砖块型图像的尺寸通常较小，其绘制过程类似于在程序窗口“贴瓷砖”，即将窗口区域按砖块

型图像的尺寸划分为许多小方格，然后在相应的方格内绘制图像，如图 4-7 所示。

用多个砖块型图像来绘制窗口中的不同区域，需要使用砖块地图（Tile Map）。砖块地图可以简单地使用一个文本文件或者二维数组来保存，记录某个位置显示的图像可以通过图像代号来表示。在游戏初始时由程序载入砖块地图文件或者二维数组，并对文件中的信息或者二维数组逐行地进行分析，然后根据不同的图像代号来分别读取不同种类的砖块型图像。

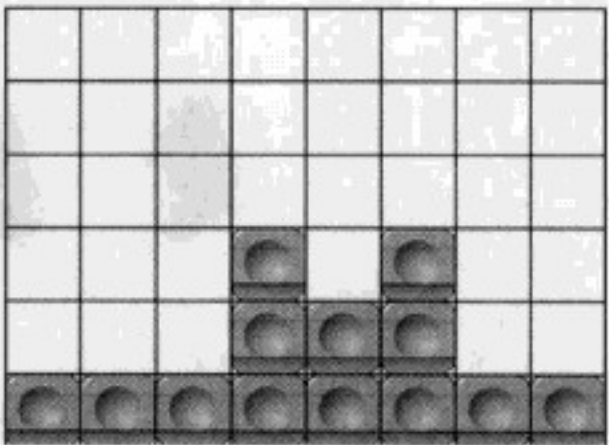


图 4-7 砖块型图像

例如，将推箱子游戏每关的地图信息放入二维数组中，可以和网格对应。其中存储 1 代表 pic1.jpg 砖块型图像，存储 2 代表 pic2.jpg 砖块型图像，依此类推。存储 0 代表此处不绘制砖块型图像。

游戏中的图片资源如图 4-8 所示，推箱子游戏中绘制砖块型图像的效果如图 4-9 所示。

```
static byte map[][]={
    { 0, 0, 1, 1, 1, 0, 0, 0 },
    { 0, 0, 1, 4, 1, 0, 0, 0 },
    { 0, 0, 1, 9, 1, 1, 1, 1 },
    { 1, 1, 1, 2, 9, 2, 4, 1 },
    { 1, 4, 9, 2, 5, 1, 1, 1 },
    { 1, 1, 1, 1, 2, 1, 0, 0 },
    { 0, 0, 0, 1, 4, 1, 0, 0 },
    { 0, 0, 0, 1, 1, 1, 0, 0 }}
```



图 4-8 推箱子游戏的图片资源

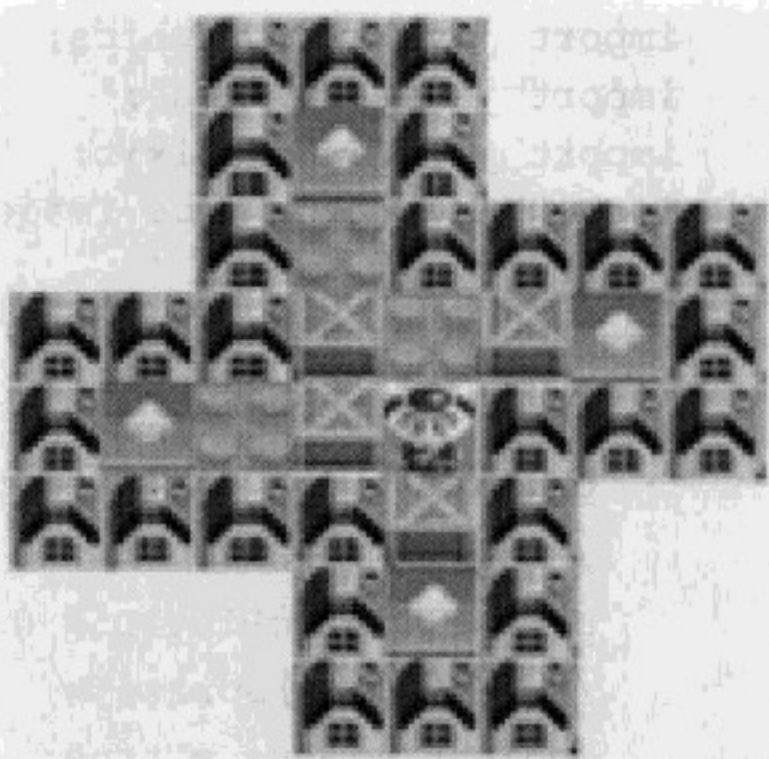


图 4-9 推箱子游戏中绘制砖块型图像的效果图

本书中的许多游戏，例如 RGP 走迷宫游戏、连连看游戏、推箱子游戏等都采用了砖块型图像技术来实现。

4.8 游戏角色开发

游戏角色也叫作游戏精灵（Sprite），是指游戏中可移动的物体，主要包括玩家控制的角色和电脑控制的角色，如怪物、敌机等。在游戏中精灵通常可以活动，往往需要通过连续地绘制多幅静态图像来表现其运动效果，如图 4-10 所示。

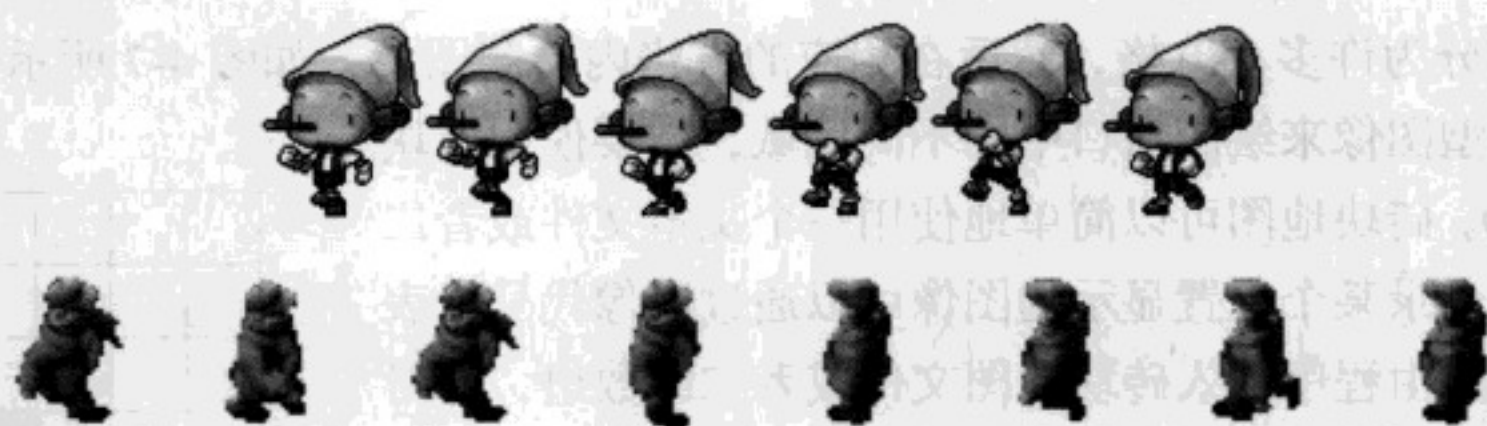


图 4-10 精灵运动效果

这里我们设计一个 `Sprite` 类，主要用来实现游戏里的人物动画和移动的效果。使用 `Sprite` 类可以读取一张张小图片，并且把它们按照一定的顺序存储在数组中，然后在屏幕上显示出其中的一张小图片，如果连续地更换显示的小图片，则屏幕上表现为一个完整的动画效果。

假如 `Sprite` 类中的人物动画由 4 帧状态图片（见图 4-11）构成，在绘制人物时，根据 `mPlayID` 的值绘制相应状态的图片。



图 4-11 精灵向右行走的图片

```
//Sprite 类
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.image.ImageObserver;
public class Sprite {
    /** Sprite 类的 x, y 坐标**/
    public int m_posX = 0, m_posY = 0;
    private Image pic[] = null; // Sprite 类的图片数组
    /**当前帧的 ID **/
    private int mPlayID = 0;
    /**是否更新绘制 Sprite**/
    boolean mFacus = true;
    public Sprite () {
        pic = new Image[4];
        for (int i = 0; i < 4; i++)
            pic[i] = Toolkit.getDefaultToolkit().getImage(
                "images\\d" + i + ".png");
    }
    /**初始化坐标**/
    public void init(int x, int y) {
        m_posX = x;
        m_posY = y;
    }
    /**设置坐标**/
    public void set(int x, int y) {
        m_posX = x;
        m_posY = y;
    }
}
```



```

/**绘制精灵**/
public void DrawSprite (Graphics g,JPanel i)
{
    g.drawImage(pic[mPlayID],m_posX,m_posY,(ImageObserver)i);
    mPlayID++; //下一帧图像
    if(mPlayID==4) mPlayID=0;
}

```

Sprite 类中的坐标更新主要修改 x 坐标（水平方向）值，每次 15 个像素。当然也可以修改 y 坐标（垂直方向）值，这里为了简化起见没修改 y 坐标值（垂直方向）。

```

/**更新精灵的坐标点**/
public void UpdateSprite () {
    if (mFacus==true)
        m_posX += 15;
    if(m_posX==300) //如果达到窗口右边缘
        m_posX=0;
}
}

```

下面的例子展示了如何显示一个精灵走动的动画，一个精灵从屏幕的左方往右方走动，显示精灵走动的 SpritePanel 类的代码如下所示：

```

import java.awt.*;
import javax.swing.JPanel;
public class SpritePanel extends JPanel implements Runnable { //绘图线程类
    private Sprite player;
    public SpritePanel()
    {
        player=new Sprite(); //创建角色精灵
        Thread t = new Thread(this); //创建一个新线程
        t.start(); //启动线程
    }
    public void run() //重载 run() 方法
    {
        while(true) //线程中的无限循环
        {
            player.UpdateSprite(); //更新角色 Sprite 类的 x, y 坐标
            try{
                Thread.sleep(500); //线程休眠 500ms
            }catch(InterruptedException e){}
            repaint(); //窗口重绘
        }
    }
    public void paint(Graphics g) //重载绘图方法
    {
        super.paint(g); //将面板上原来画的东西擦掉
        //先清屏幕，否则原来画的东西仍存在
        g.clearRect(0, 0, this.getWidth(), this.getHeight());
        player.DrawSprite(g, this); //绘制精灵
    }
}

```

仍然使用例 4-4 的主程序，其中做如下修改：


```
//c.add(new TetrisPanel());
c.add(new SpritePanel());
```

程序运行效果如图 4-12 所示，一个精灵从屏幕的左方往右方走动，如果超出屏幕边界，则从左方重新开始走动，如此循环一直到程序关闭为止。

当然在游戏中人物角色 Sprite 的移动是玩家控制的，这就需要在键盘事件中判断用户按键的方向，从而更新角色 Sprite 的 x, y 坐标即可。



图 4-12 精灵移动效果

4.9 游戏声音效果的设定

游戏中的音乐能够丰富游戏的内涵，同时增强游戏的可玩性。游戏中的声音效果一般分为两类，分别是动作音效和场景音乐。前者用于动作的配音，以便增强游戏角色的真实感；后者用于烘托游戏气氛，通过为不同的背景配备相对应的音乐来表达特定的情感。

Java 提供了丰富的 API 用于对声音进行处理和播放，其中最常用的是一个使用数字化样本文件工作的包，专门用于载入声音文件并通过音频混合器进行播放。该包叫作 `javax.sound.sampled`，其所支持的声音文件格式有以下几种：AIFF、AU 和 WAV 3 种。

Java 中播放声音文件与显示图像文件一样方便，同样只需要先将声音文件装载进来，然后播放即可。下面以 WAV 文件的播放为例来进行说明。首先需要打开声音文件并读取其中的信息，主要包括如下几个步骤。

- (1) 新建一个文件对象获取 WAV 文件数据。

```
File file = new File("sound.wav");
```

- (2) 将 WAV 文件转换为音频输入流。

```
AudioInputStream stream = AudioSystem.getAudioInputStream(file);
```

- (3) 获取音频格式。

```
AudioFormat format = stream.getFormat();
```

众所周知，Java 通过流对象（Stream）来统一处理输入与输出数据，声音数据也不例外，处理声音数据输入的流类叫作 `AudioInputStream`，它是具有指定音频格式和长度的输入流。除此之外，还需要使用 `AudioSystem` 类，该类用于充当取样音频系统资源的入口点，提供许多在不同格式间转换音频数据的方法，以及在音频文件和流之间进行转换的方法。

`AudioFormat` 类是在声音流中指定特定数据安排的类。通过检查以音频格式存储的信息，可以发现现在二进制声音数据中解释位的方式。

获取声音文件信息之后接下来就能够对声音数据进行播放了，主要包括如下几个步骤。

- (1) 设置音频行信息。

```
DataLine.Info info = new DataLine.Info(Clip.class, format);
```

- (2) 建立音频行。

```
Clip clip = (Clip)AudioSystem.getLine(info);
```

- (3) 将音频数据流读入音频行。

```
clip.open(stream);
```

- (4) 播放音频行。

```
clip.start();
```


其中涉及的类包括如下几个。

Line:Line 接口表示单声道或多声道音频供给。

DataLine:包括一些音频传输控制方法,这些方法可以启动、停止、消耗和刷新通过数据行传入的音频数据。

DataLine.Info:提供音频数据行的信息。包括受数据行支持的音频格式、其内部缓冲区的最小和最大值等。

Clip 接口表示特殊种类的数据行,该数据行的音频数据可以在回放前加载,而不是实时流出。音频剪辑的回放可以使用 `start()`和 `stop()`方法开始和终止。这些方法不重新设置介质的位置, `start()`方法的功能是从回放最后停止的位置继续回放。

下面建立 **SoundPlayer** 类来播放声音效果,代码如下:

```
import javax.sound.sampled.*;
import java.io.*;
public class SoundPlayer {
    File file;    AudioInputStream stream;
    AudioFormat format;    DataLine.Info info;
    Clip clip;
    SoundPlayer() {
    }

    public void loadSound(String fileName) { //打开声音文件
        file = new File(fileName);
        try {
            stream = AudioSystem.getAudioInputStream(file);
        } catch (UnsupportedAudioFileException ex) {
        } catch (IOException ex) {
        }
        format = stream.getFormat();
    }

    public void playSound() { //播放声音
        info = new DataLine.Info(Clip.class, format);
        try {
            clip = (Clip) AudioSystem.getLine(info);
        } catch (LineUnavailableException ex) {
        }
        try {
            clip.open(stream);
        } catch (LineUnavailableException ex) {
        }
        catch (IOException ex) {
        }
        clip.start();
    }
}
```

给角色动作添加动作音乐,在用户按空格键跳跃时播放音效,代码如下:

```
public class GamePanel extends Panel implements Runnable, KeyListener {
    public SoundPlayer sound = new SoundPlayer();
    public void keyPressed(KeyEvent e) {
        int keycode = e.getKeyCode();
        switch (keycode) {
            case KeyEvent.VK_DOWN:
                break;
            case KeyEvent.VK_SPACE:
                sound.loadSound("Sounds/jump.wav");
                sound.playSound();
        }
    }
}
```


第 5 章

推箱子游戏

5.1 推箱子游戏介绍

经典的推箱子游戏是一款来自日本的古老游戏，目的是训练玩家的逻辑思考能力。在一个狭小的仓库中，要求把木箱放到指定的位置，稍不小心就会出现箱子无法移动或者通道被堵住的情况，因此要求玩家巧妙地利用有限的空间和通道，合理安排移动的次序和位置，才能顺利地完成任务。

推箱子游戏的功能如下所述。

运行游戏载入相应的地图，屏幕中出现一名推箱子的工人^❶，其周围是围墙^❷、人可以走的通道^❸、几个可以移动的箱子^❹和箱子放置的目的地^❺。玩家通过按上下左右键控制工人推箱子，当所有箱子都推到了目的地后出现过关信息，并显示下一关。如果推错了，玩家通过单击鼠标右键可以撤销上次的移动操作，还可以按空格键重新玩这一关，直到通过全部关卡。

本章开发推箱子游戏，推箱子游戏的界面如图 5-1 所示。

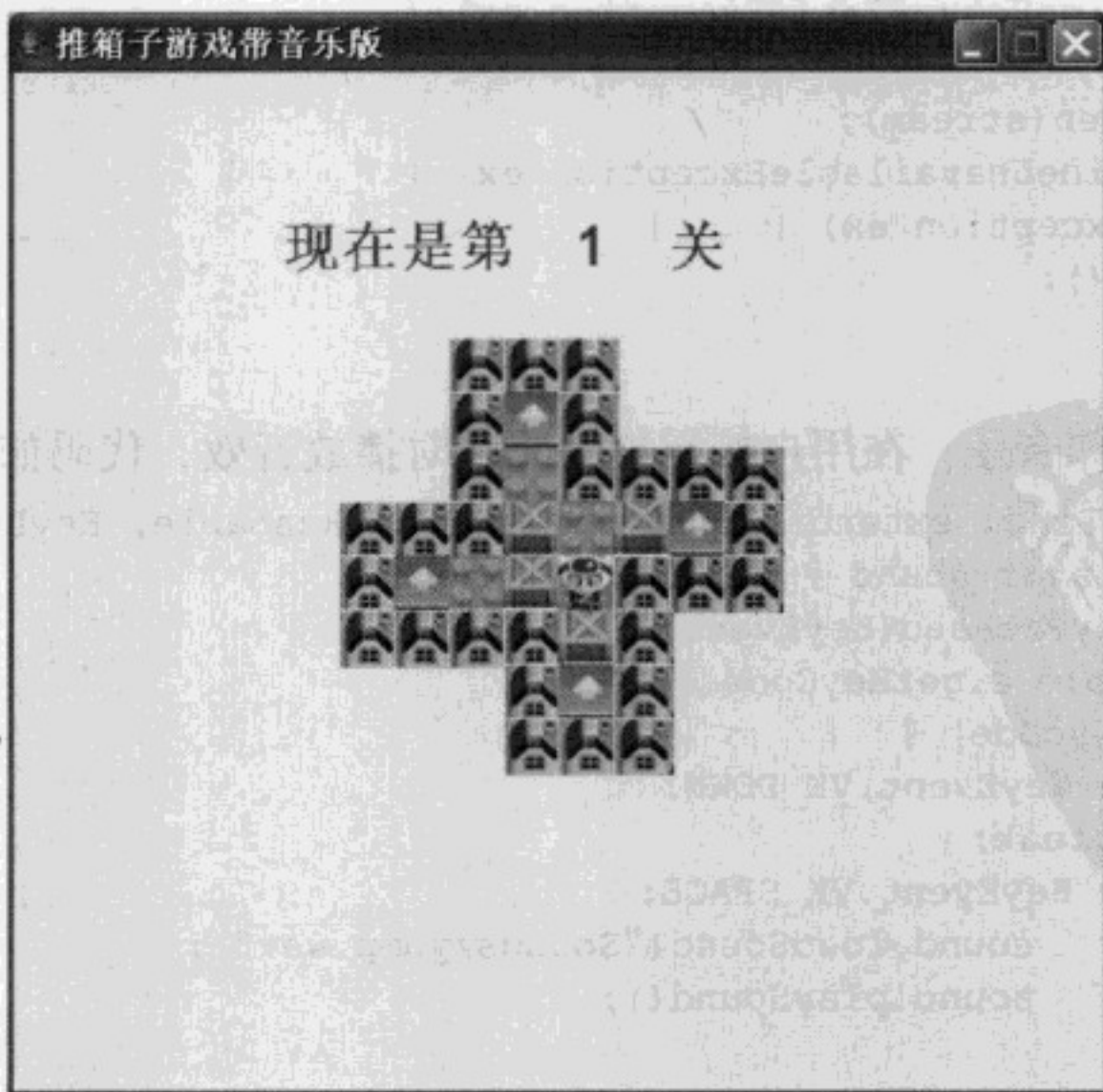


图 5-1 推箱子游戏界面

游戏中用到的图片资源如下。

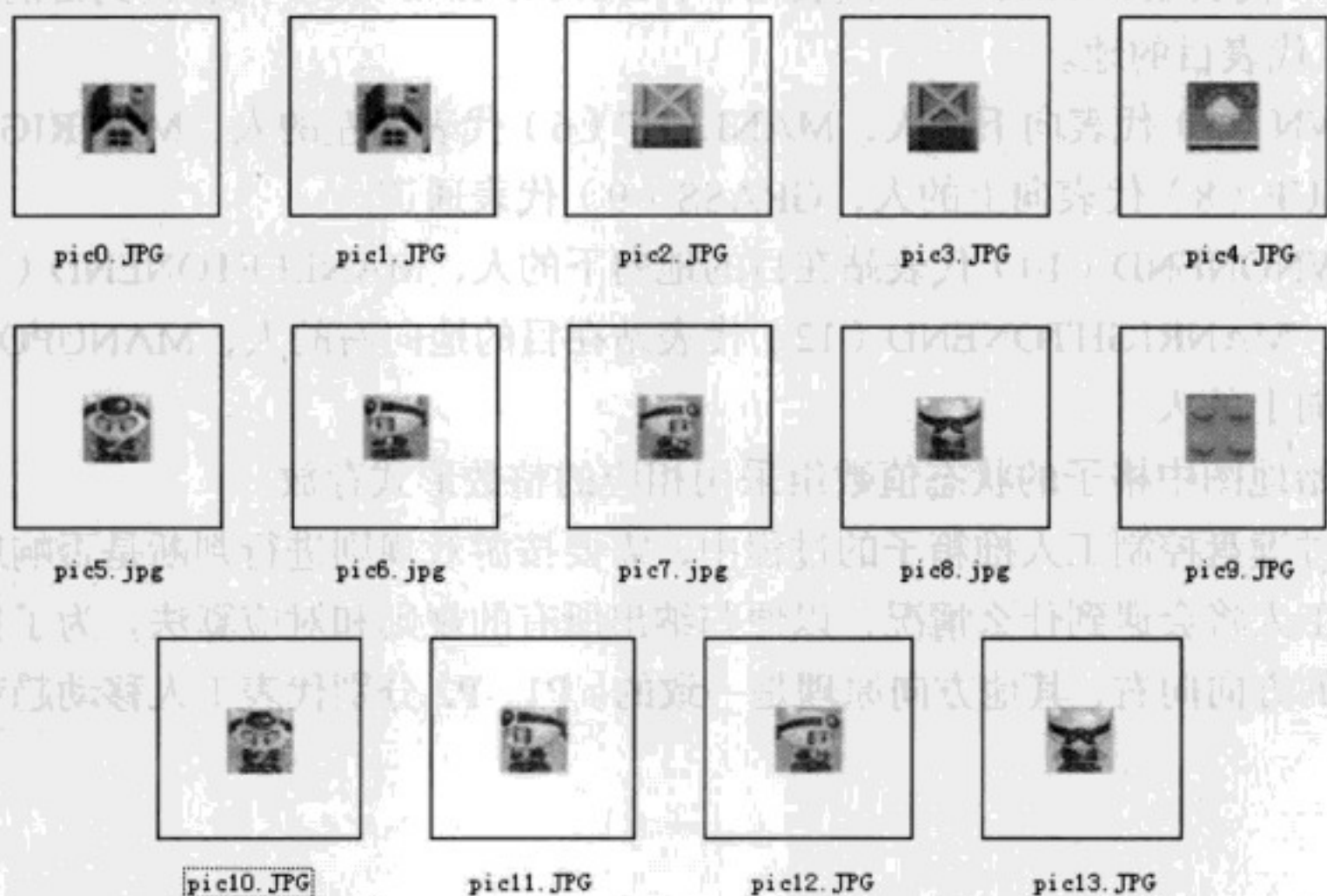


图 5-2 推箱子游戏用到的图片资源

图片资源说明如下。

- pic1：墙；pic2：箱子；pic3：箱子在目的地上；pic4：目的地；
- pic5：向下的人；pic6：向左的人；pic7：向右的人；pic8：向上的人；pic9：通道；
- pic10：站在目的地向下的人；pic11：站在目的地向左的人；
- pic12：站在目的地向右的人；pic13：站在目的地向上的人。

5.2 程序设计的思路

首先我们来确定一下游戏开发的难点。对工人的操作很简单，就是 4 个方向的移动，工人移动，箱子也随之移动，因此对按键的处理也比较简单。当箱子到达目的地位置时，就会产生游戏过关事件，需要一个逻辑判断。那么我们仔细想一下，所有这些事件都发生在一张地图中。这张地图包括箱子的初始化位置、箱子最终放置的位置和围墙障碍等。每一关地图都要更换，这些位置也要变。因此我们发现每关的地图数据是最关键的，它决定每关的不同场景和物体位置。那么我们就重点分析一下地图。

我们把地图想象成一个网格，每个格子就是工人每次移动的步长（这里为 30 像素），也是箱子移动的距离，这样问题就简化多了。首先我们设计一个 `mapRow×mapColumn` 的二维数组 `map`。按照这样的框架来思考，对于格子的 (x, y) 两个屏幕像素坐标，可以由二维数组下标 (i, j) 换算得到。

换算公式为： $\text{left}x + j \times 30, \text{left}Y + i \times 30$

每个格子状态值分别用枚举类型值：

//定义一些常量，对应地图的元素

```
final byte WALL = 1, BOX = 2, BOXONEND = 3, END = 4, MANDOWN = 5,
MANLEFT = 6, MANRIGHT = 7, MANUP = 8, GRASS = 9,
MANDOWNONEND = 10, MANLEFTONEND = 11,
```


MANRIGHTONEND = 12, MANUPONEND = 13;

WALL (1) 代表墙, BOX (2) 代表箱子, BOXONEND (3) 代表放到目的地的箱子, END (4) 代表目的地。

MANDOWN (5) 代表向下的人, MANLEFT (6) 代表向左的人, MANRIGHT (7) 代表向右的人, MANUP (8) 代表向上的人, GRASS (9) 代表通道。

MANDOWNONEND (10) 代表站在目的地向下的人, MANLEFTONEND (11) 代表站在目的地向左的人, MANRIGHTONEND (12) 代表站在目的地向右的人, MANUPONEND (13) 代表站在目的地向上的人。

存储的原始地图中格子的状态值数组采用相应的整数形式存放。

在玩家通过键盘控制工人推箱子的过程中, 需要按游戏规则进行判断是否响应该按键指示。下面分析一下工人将会遇到什么情况, 以便归纳出所有的规则 and 对应算法。为了描述方便, 假设工人移动趋势的方向向右, 其他方向原理是一致的。P1、P2 分别代表工人移动趋势方向前的两个方格。



P1

P2

(1) 前方 P1 是围墙

如果工人前方是围墙 (即阻挡工人的路线)

```
{
    退出规则判断, 布局不做任何改变;
}
```

(2) 前方 P1 是通道 (GRASS) 或目的地 (END)

如果工人前方是通道或目的地

```
{
    工人可以进到 P1 方格; 修改相关位置格子的状态值。
}
```

(3) 前方 P1 是箱子



P2

在前面两种情况中, 只要根据前方 P1 处的物体就可以判断出工人是否可以移动; 而在第 3 种情况中, 需要判断箱子前方 P2 处的物体才能判断出工人是否可以移动。此时有以下可能:

- ① P1 处为箱子 (BOX) 或者放到目的地的箱子 (BOXONEND), P2 处为通道 (GRASS); 工人可以进到 P1 方格; P2 方格状态为箱子。修改相关位置格子的状态值。
- ② P1 处为箱子 (BOX) 或者放到目的地的箱子 (BOXONEND), P2 处为目的地 (END);

工人可以进到 P1 方格；P2 方格状态为放置好的箱子。修改相关位置格子的状态值。

③ P1 处为箱子 (BOX)，P2 处为墙 (WALL)。

退出规则判断，布局不做任何改变。

综合前面的分析，可以设计出整个游戏的实现流程。

整个游戏的源文件说明如下：

GameFrame.java：游戏界面视图；

Map.java：封装游戏的当前状态；

MapFactory.java：提供的地图数据。

5.3 程序设计的步骤

5.3.1 设计地图数据类 (MapFactory.java)

地图数据类保存所有关卡的原始地图数据，每关数据为一个二维数组，因此此处 map 是一个三维数组。

```
import java.io.InputStream;
public class MapFactory {
    static byte map[][][]={
        {
            { 0, 0, 1, 1, 1, 0, 0, 0 },
            { 0, 0, 1, 4, 1, 0, 0, 0 },
            { 0, 0, 1, 9, 1, 1, 1, 1 },
            { 1, 1, 1, 2, 9, 2, 4, 1 },
            { 1, 4, 9, 2, 5, 1, 1, 1 },
            { 1, 1, 1, 1, 2, 1, 0, 0 },
            { 0, 0, 0, 1, 4, 1, 0, 0 },
            { 0, 0, 0, 1, 1, 1, 0, 0 }
        },
        {
            { 1, 1, 1, 1, 1, 0, 0, 0, 0 },
            { 1, 9, 9, 5, 1, 0, 0, 0, 0 },
            { 1, 9, 2, 2, 1, 0, 1, 1, 1 },
            { 1, 9, 2, 9, 1, 0, 1, 4, 1 },
            { 1, 1, 1, 9, 1, 1, 1, 4, 1 },
            { 0, 1, 1, 9, 9, 9, 9, 4, 1 },
            { 0, 1, 9, 9, 9, 1, 9, 9, 1 },
            { 0, 1, 9, 9, 9, 1, 1, 1, 1 },
            { 0, 1, 1, 1, 1, 1, 0, 0, 0 }
        },
        ...//省略其余关卡数据
    };
    static int count=map.length;
    public static byte[][] getMap(int grade)
    {byte temp[][];
        if(grade>=0 && grade<count)
            temp=map[grade];
        else
            temp=map[0];
    }
```



```

        int row=temp.length;
        int column=temp[0].length;
        byte[][] result=new byte[row][column];
        for(int i=0;i<row;i++)
            for(int j=0;j<column;j++)
                result[i][j]=temp[i][j];
        return result;
    }
    public static int getCount()
    {
        return count;
    }
}

```

5.3.2 设计地图类 (Map.java)

由于每移动一步，需要保存当前的游戏状态，所以此处定义此地图类，保存人的位置和游戏的地图的当前状态。撤销移动时，恢复地图的操作通过此类获取需要的人的位置、地图的当前状态和关卡数来实现。

```

public class Map {
    int manX=0;
    int manY=0;
    byte map[][];
    int grade;
    //此构造方法用于撤销操作
    //撤销操作只需要人的位置和地图的当前状态
    public Map(int manX,int manY,byte [][]map)
    {
        this.manX=manX;
        this.manY=manY;
        int row=map.length;
        int column=map[0].length;
        byte temp[][]=new byte[row][column];
        for(int i=0;i<row;i++)
            for(int j=0;j<column;j++)
                temp[i][j]=map[i][j];
        this.map=temp;
    }

    //此构造方法用于保存操作
    //恢复地图时需要人的位置、地图的当前状态和关卡数（关卡切换时此为基数）
    public Map(int manX,int manY,byte [][]map,int grade)
    {
        this(manX,manY,map);
        this.grade=grade;
    }

    public int getManX() {
        return manX;
    }
    public int getManY() {
        return manY;
    }
    public byte[][] getMap() {

```



```

        return map;
    }
    public int getGrade() {
        return grade;
    }
}

```

5.3.3 设计游戏面板类 (GameFrame.java)

游戏面板类完成游戏的界面刷新显示, 以及相应鼠标键盘的相关事件。

```

//推箱子带音乐版
//右键单击——悔棋功能
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.util.ArrayList;
import javax.sound.midi.*;
import javax.swing.*;

public class GameFrame extends JFrame implements ActionListener, MouseListener,
KeyListener{
    //主面板类
    private int grade = 0;
    // row, column 记载人的行号、列号
    // leftX, leftY 记载左上角图片的位置, 避免图片从 (0, 0) 坐标开始
    private int row = 7, column = 7, leftX = 0, leftY = 0;
    //记载地图的行列数
    private int mapRow = 0, mapColumn = 0;
    //width, height 记载屏幕的大小
    private int width = 0, height = 0;
    private boolean acceptKey = true;
    //程序所用到的图片
    private Image pic[] = null;
    private byte[][] map = null;
    private ArrayList list = new ArrayList();
    Sound sound;
}

```

关于格子状态值的常量, 对应地图的元素。

```

final byte WALL = 1, BOX = 2, BOXONEND = 3, END = 4, MANDOWN = 5,
MANLEFT = 6, MANRIGHT = 7, MANUP = 8, GRASS = 9,
MANDOWNONEND = 10, MANLEFTONEND = 11,
MANRIGHTONEND = 12, MANUPONEND = 13;

```

在构造方法 GameFrame() 中, 调用 initMap() 方法来初始化本关 grade 游戏地图, 清空悔棋信息列表 list, 同时播放 MIDI 背景音乐。

```

public GameFrame() {
    super("推箱子游戏带音乐版");
    setSize(600, 600);
    setVisible(true);
    setResizable(false);
    setLocation(300, 20);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container cont = getContentPane();
    cont.setLayout(null);
    cont.setBackground(Color.black);
}

```



```

//最初始的 13 张图片
getPic();
width = this.getWidth();
height = this.getHeight();
this.setFocusable(true);
initMap();    //初始化本关 grade 游戏地图, 清空悔棋信息列表 list
this.addKeyListener(this);
this.addMouseListener(this);
sound=new Sound();
sound.loadSound();    //播放 MIDI 背景音乐
}

```

initMap()方法的作用是初始化本关 grade 游戏地图, 清空悔棋信息列表 list。调用 getMapSizeAndPosition()方法获取游戏区域大小及显示游戏的左上角位置 (leftX, leftY)。

```

public void initMap() {
    map = getMap(grade);
    list.clear();
    getMapSizeAndPosition();
    getManPosition();
}

```

getManPosition()方法的作用是获取工人的当前位置 (row, column)。

```

public void getManPosition() {
    for (int i = 0; i < map.length; i++)
        for (int j = 0; j < map[0].length; j++)
            if (map[i][j] == MANDOWN || map[i][j] == MANDOWNONEND
                || map[i][j] == MANUP || map[i][j] == MANUPONEND
                || map[i][j] == MANLEFT || map[i][j] == MANLEFTONEND
                || map[i][j] == MANRIGHT || map[i][j] == MANRIGHTONEND)
            {
                row = i;
                column = j;
                break;
            }
}

```

getManPositionzeAndPosition()方法用来获取游戏区域大小及显示游戏的左上角位置 (leftX, leftY)。

```

private void getMapSizeAndPosition() {
    //TODO Auto-generated method stub
    mapRow = map.length;
    mapColumn = map[0].length;
    leftX = (width - map[0].length * 30) / 2;
    leftY = (height - map.length * 30) / 2;
    System.out.println(leftX);
    System.out.println(leftY);
    System.out.println(mapRow);
    System.out.println(mapColumn);
}

```

getPic()方法用来加载要显示的图片。

```

public void getPic() {
    pic = new Image[14];
    for(int i=0; i<13; i++)
    {
        pic[i] = Toolkit.getDefaultToolkit().getImage("images\\pic"+i+".JPG");
    }
}

```


grassOrEnd(byte man)方法判断人所在的位置是通道 GRASS 还是目的地 END。

```
public byte grassOrEnd(byte man) {
    byte result = GRASS;
    if (man == MANDOWNONEND || man == MANLEFTONEND || man == MANRIGHTONEND || man ==
MANUPONEND)
        result = END;
    return result;
}
```

游戏逻辑主要包括人物移动。人物可以向 4 个方向移动,向上移动(向下、向左、向右类似)。推箱子游戏的特殊性,决定了人物移动涉及 3 个位置,即人物的当前位置、人物将要移动到的位置和箱子要到达的位置。下面以人物向上移动为例进行说明。

向上移动涉及的 3 个位置为 人物当前位置 (map[row][column])、人物上一位的位置 p1 (map[row-1][column]) 和上上一位的位置 p2 (map[row-2][column])。

首先判断 p1 处的图像类型,这里的图像类型可能为 BOX、BOXONEND、WALL、GRASS 或 END。

(1) 如果前方 P1 是围墙 WALL,因为不能移动,所以不用处理直接返回即可。

(2) 如果 p1 为 BOX 或 BOXONEND,则判断 p2 处的图像类型,这里需要处理的图像类型为 END 或 GRASS,如果 p2 的图像类型是这两种类型时,就进行如下处理:

保存当前全部游戏的地图信息到 ArrayList 类型的 list 中,用于撤销动作。

```
Map currMap = new Map(row, column, map);
list.add(currMap);
```

判断 p2 处是否为目的地,如果是目的地则 P2 方格状态为放置好的箱子,否则为箱子。

```
byte boxTemp = map[row-2][column] == END?BOXONEND: BOX;
map[row - 2][column] = boxTemp;
```

人往上走一步,需要判断 p1 处是 BOX 还是 BOXONEND,是 BOX 则 P1 方格状态改为 MANUP (人在 p1 位置),否则改为 MANUPONEND (人在目的地)。

```
byte manTemp = map[row - 1][column] == BOX ? MANUP : MANUPONEND;
map[row - 1][column] = manTemp;
```

将人刚才站的地方变成 GRASS 或者 END。

```
map[row][column] = grassOrEnd(map[row][column])
```

修改人的位置在 map 数组中的行坐标 row,做自减操作。

(3) 如果 p1 的图像类型为 GRASS 或者 END,则只需要做如下处理。

保存当前全部游戏的地图信息到 ArrayList 类型的 list 中,用于撤销动作。

```
Map currMap = new Map(row, column, map);
list.add(currMap);
```

判断 p1 处是否为目的地,如果是目的地则 P1 方格状态改为 MANUPONEND (人在目的地),否则 P1 方格状态改为 MANUP (人在 p1 位置)。

```
byte temp = map[row-1][column]== END? MANUPONEND : MANUP;
map[row - 1][column] = temp;
```

将人刚才站的地方变成 GRASS 或者 END。

```
map[row][column] = grassOrEnd(map[row][column])
```

修改人的位置在 map 数组中的行坐标 row,做自减操作。

具体向上移动的代码如下所示:

```
private void moveUp() { //向上
    //上一位 p1 为 WALL
```



```

    if (map[row - 1][column] == WALL)
        return;
    //上一位 p1 为 BOX 或 BOXONEND, 需考虑 P2
    if (map[row - 1][column] == BOX || map[row - 1][column] == BOXONEND) {
        //上上一位 p2 为 END 或 GRASS, 则向上一步, 其他情况不用处理
        if (map[row - 2][column] == END || map[row - 2][column] == GRASS) {
            Map currMap = new Map(row, column, map);
            list.add(currMap);
            byte boxTemp = map[row - 2][column] == END ? BOXONEND : BOX;
            byte manTemp = map[row - 1][column] == BOX ?
                MANUP : MANUPONEND;
            //箱子变成 temp, 箱子往前一步
            map[row - 2][column] = boxTemp;
            //人变成 MANUP, 往上走一步
            map[row - 1][column] = manTemp;
            //将人刚才站的地方变成 GRASS 或者 END
            map[row][column] = grassOrEnd(map[row][column]);
            //人离开后修改人的坐标
            row--;
        }
    } else {
        //上一位为 GRASS 或 END, 无需考虑 P2, 其他情况不用处理
        if (map[row - 1][column] == GRASS || map[row - 1][column] == END) {
            Map currMap = new Map(row, column, map);
            list.add(currMap);
            byte temp = map[row - 1][column] == END ? MANUPONEND : MANUP;
            //人变成 temp, 人往上走一步
            map[row - 1][column] = temp;
            //人刚才站的地方变成 GRASS 或者 END
            map[row][column] = grassOrEnd(map[row][column]);
            //人离开后修改人的坐标
            row--;
        }
    }
}

```

向下、向左、向右移动与向上类似, 这里不再赘述。

```

private void moveDown() { //向下
    ...//略
}
private void moveLeft() { //向左
    //左一位 p1 为 WALL
    if (map[row][column - 1] == WALL)
        return;
    //左一位 p1 为 BOX 或 BOXONEND
    if (map[row][column - 1] == BOX || map[row][column - 1] == BOXONEND) {
        //左左一位 p2 为 END, GRASS, 则向左一步, 其他情况不用处理
        if (map[row][column - 2] == END || map[row][column - 2] == GRASS) {
            Map currMap = new Map(row, column, map);
            list.add(currMap);
            byte boxTemp = map[row][column - 2] == END ? BOXONEND : BOX;
            byte manTemp = map[row][column - 1] == BOX ? MANLEFT

```



```

        : MANLEFTONEND;
        //箱子变成 boxTemp, 箱子往左一步
        map[row][column - 2] = boxTemp;
        //人变成 manTemp, 往左走一步
        map[row][column - 1] = manTemp;
        //人刚才站的地方变成 grassOrEnd(map[row][column])
        map[row][column] = grassOrEnd(map[row][column]);
        column--;
    }
} else {
    //左一位为 GRASS 或 END, 其他情况不用处理
    if (map[row][column - 1] == GRASS || map[row][column - 1] == END) {
        Map currMap = new Map(row, column, map);
        list.add(currMap);
        byte temp = map[row][column - 1] == END ? MANLEFTONEND
            : MANLEFT;
        //人变成 temp, 人往左走一步
        map[row][column - 1] = temp;
        //人刚才站的地方变成 grassOrEnd(map[row][column])
        map[row][column] = grassOrEnd(map[row][column]);
        column--;
    }
}
}
private void moveRight() { //向右
    ...//略
}

```

isFinished()方法验证玩家是否过关。如果有目的地 END 值或人在目的地则没有成功。

```

public boolean isFinished() {
    for (int i = 0; i < mapRow; i++)
        for (int j = 0; j < mapColumn; j++)
            if (map[i][j] == END || map[i][j] == MANDOWNONEND
                || map[i][j] == MANUPONEND || map[i][j] == MANLEFTONEND
                || map[i][j] == MANRIGHTONEND)
                return false;
    return true;
}

```

paint(Graphics g)方法绘制整个游戏区域的图形。

```

public void paint(Graphics g) //绘图
{
    for (int i = 0; i < mapRow; i++)
        for (int j = 0; j < mapColumn; j++) {
            //画出地图, i 代表行数, j 代表列数
            if (map[i][j] != 0)
                g.drawImage(pic[map[i][j]], leftX + j * 30, leftY + i * 30, this);
        }
    g.setColor(Color.RED);
    g.setFont(new Font("楷体_2312", Font.BOLD, 30));
    g.drawString("现在是第", 150, 140);
    g.drawString(String.valueOf(grade+1), 310, 140);
}

```



```
g.drawString("关", 360, 140);
```

```
}
```

getManX()、getManY()方法返回人的位置。

```
public int getManX() {
    return row;
```

```
}
```

```
public int getManY() {
    return column;
```

```
}
```

getGrade()方法返回当前关卡数。

```
public int getGrade() {
    return grade;
```

```
}
```

getMap(int grade)方法返回当前关卡的地图信息。

```
public byte[][] getMap(int grade) {
    return MapFactory.getMap(grade);
```

```
}
```

DisplayToast(String str)方法用来显示提示信息对话框。

```
/* 显示提示信息对话框 */
```

```
public void DisplayToast(String str) {
    JOptionPane.showMessageDialog(null, str, "提示",
        JOptionPane.ERROR_MESSAGE);
```

```
}
```

undo()方法的作用是撤销移动操作。

```
public void undo() {
    if (acceptKey) {
        // 撤销
        if (list.size() > 0) {
            //如果撤销则必须走过
            Map priorMap = (Map) list.get(list.size() - 1);
            map = priorMap.getMap();
            row = priorMap.getManX();
            column = priorMap.getManY();
            repaint();
            list.remove(list.size() - 1);
        } else
            DisplayToast("不能再撤销!");
    } else {
        DisplayToast("此关已完成，不能撤销!");
    }
}
```

nextGrade()方法实现下一关初始化，并调用 **repaint()**方法显示游戏界面。

```
public void nextGrade() {
    if (grade >= MapFactory.getCount() - 1) {
        DisplayToast("恭喜你完成所有关卡!");
        acceptKey = false;
    } else {
        grade++;
        initMap();
        repaint();
    }
}
```



```

        acceptKey = true;
    }
}

```

priorGrade()方法实现上一关初始化并调用 repaint()方法显示游戏界面。

```

public void priorGrade() {
    grade--;
    acceptKey = true;
    if (grade < 0)
        grade = 0;
    initMap();
    repaint();
}

```

键盘相关事件如下。窗体的 keyPressed 按键事件中根据用户的按键消息，分别调用 4 个方向移动的方法。

```

public void keyPressed(KeyEvent e) // 键盘事件
{
    if (e.getKeyCode() == KeyEvent.VK_UP) {
        // 向上
        moveUp();
    }
    if (e.getKeyCode() == KeyEvent.VK_DOWN) {
        // 向下
        moveDown();
    }
    if (e.getKeyCode() == KeyEvent.VK_LEFT) { // 向左
        moveLeft();
    }
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) { // 向右
        moveRight();
    }
    repaint();
    if (isFinished()) {
        // 禁用按键
        acceptKey = false;
    }
    if (grade == 10) { JOptionPane.showMessageDialog(this, "恭喜通过最后一关"); }
    else
    {
        // 提示进入下一关
        String msg = "恭喜您通过第" + grade + "关!!!\n 是否要进入下一关? ";
        int type = JOptionPane.YES_NO_OPTION;
        String title = "过关";
        int choice = 0;
        choice = JOptionPane.showConfirmDialog(null, msg, title, type);
        if (choice == 1) System.exit(0);
        else if (choice == 0)
        {
            // 进入下一关
            acceptKey = true;
            nextGrade();
        }
    }
}
}

```



```

}
public void actionPerformed(ActionEvent arg0) {
    //TODO Auto-generated method stub
}
public void keyReleased(KeyEvent arg0) {
    //TODO Auto-generated method stub
}
public void keyTyped(KeyEvent arg0) {
    //TODO Auto-generated method stub
}

```

鼠标相关事件如下：

```

public void mouseClicked(MouseEvent e) {
    // TODO Auto-generated method stub
    if (e.getButton() == MouseEvent.BUTTON3) //右键撤销移动
    {
        undo(); //撤销移动
    }
}

```

程序入口 Main 方法实例化一个 GameFrame 窗口。

```

public static void main(String[] args)
{
    new GameFrame();
}

```

5.3.4 播放背景音乐类 (Sound.java)

```

import javax.sound.midi.*;
import java.io.File;
class Sound//播放背景音乐
{
    String path=new String("musics\\");
    String file=new String("nor.mid");
    Sequence seq;
    Sequencer midi;
    boolean sign;
    void loadSound()
    {
        try {
            seq=MidiSystem.getSequence(new File(path+file));
            midi=MidiSystem.getSequencer();
            midi.open();
            midi.setSequence(seq);
            midi.start();
            midi.setLoopCount(Sequencer.LOOP_CONTINUOUSLY);
        }
        catch (Exception ex) {ex.printStackTrace();}
        sign=true;
    }
    void mystop(){midi.stop();midi.close();sign=false;}
    boolean isplay(){return sign;}
    void setMusic(String e){file=e;}
}

```


游戏动作音效播放通常比较短暂，其播放时间最多只有1~2秒，而游戏背景音乐则需要持续比较长的时间，少则十几秒，多则几分钟甚至更长时间。Java 支持的音乐文件格式主要有：CD、MP3 和 MIDI。这里采用的是 MIDI 文件格式。

MIDI (Musical Instrument Digital Interface, 乐器数字接口) 是 20 世纪 80 年代初为解决电声乐器之间的通信问题而提出的。MIDI 传输的不是声音信号，而是音符、控制参数等指令，它指示 MIDI 设备要做什么以及怎么做。例如演奏哪个音符、多大音量等。它们被统一表示成 MIDI 消息 (MIDI Message)。

Java 提供了专门的包用来处理和播放 MIDI 音乐，包名为 javax.sound.midi，其中包括了与 MIDI 相关的各个类及其方法。读取 MIDI 文件音频序列，其主要步骤如下。

(1) 打开 MIDI 文件。

```
Sequence sequence = MidiSystem.getSequence(new File(filename));
```

(2) 建立音频序列。

```
Sequencer sequencer = MidiSystem.getSequencer();
```

(3) 打开音频序列。

```
sequencer.open();
```

读取 MIDI 音频序列并初始化音频序列器之后，接下来便可以播放 MIDI 音乐了，播放 MIDI 音乐的步骤如下。

(1) 读取即将播放的音频序列。

```
sequencer.setSequence(sequence);
```

(2) 播放音频序列。

```
sequencer.start();
```

如果要循环地播放 MIDI 音乐，则可以使用 Sequencer 的 isRunning() 方法来进行判断，若该方法返回值为 False，则说明音乐已经播放完毕，此时可以再次调用 Start() 方法重新开始播放。

第 6 章

雷电飞机射击游戏

6.1 雷电游戏介绍

雷电游戏因其操作简单、节奏明快，作为纵轴射击的经典之作，“雷电”系列受到了广大玩家的喜爱，可以说是老少咸宜的游戏了。

本章开发模拟雷电游戏的飞机射击游戏，下方是玩家的飞机，能自动不断地发射子弹，上方是随机出现的敌方飞机（数量不超过 5 架）。玩家可以通过键盘的方向键控制自己飞机的移动，当玩家飞机的子弹碰到敌方飞机时，敌方飞机会出现爆炸效果。游戏运行界面如图 6-1 所示。

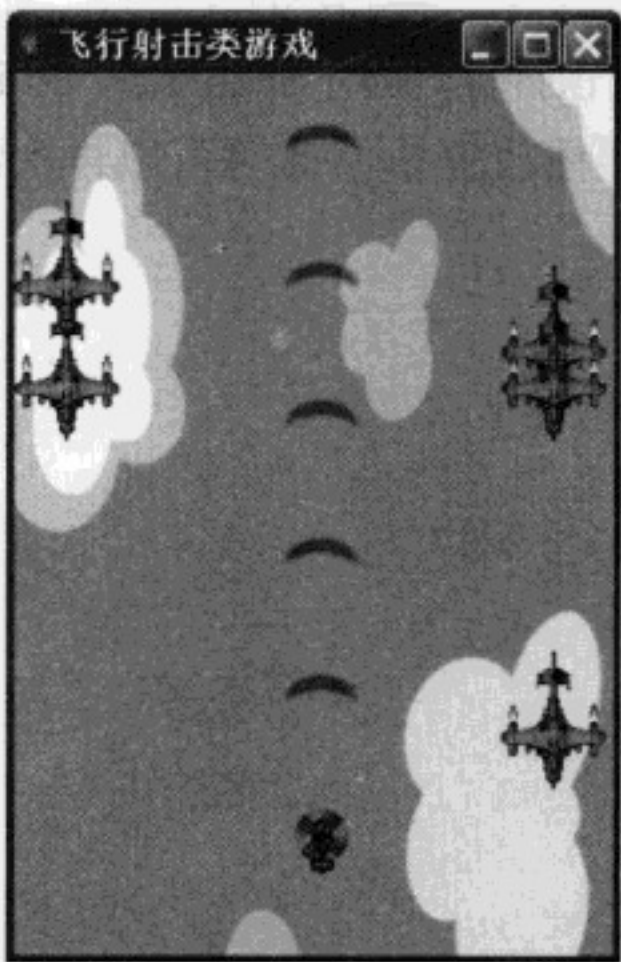


图 6-1 飞机射击游戏运行界面

6.2 程序设计的思路

6.2.1 游戏素材

游戏程序中用到敌方飞机、我方飞机、子弹、敌机被击中的爆炸图片等，分别使用图 6-2 所

示的图片来表示。

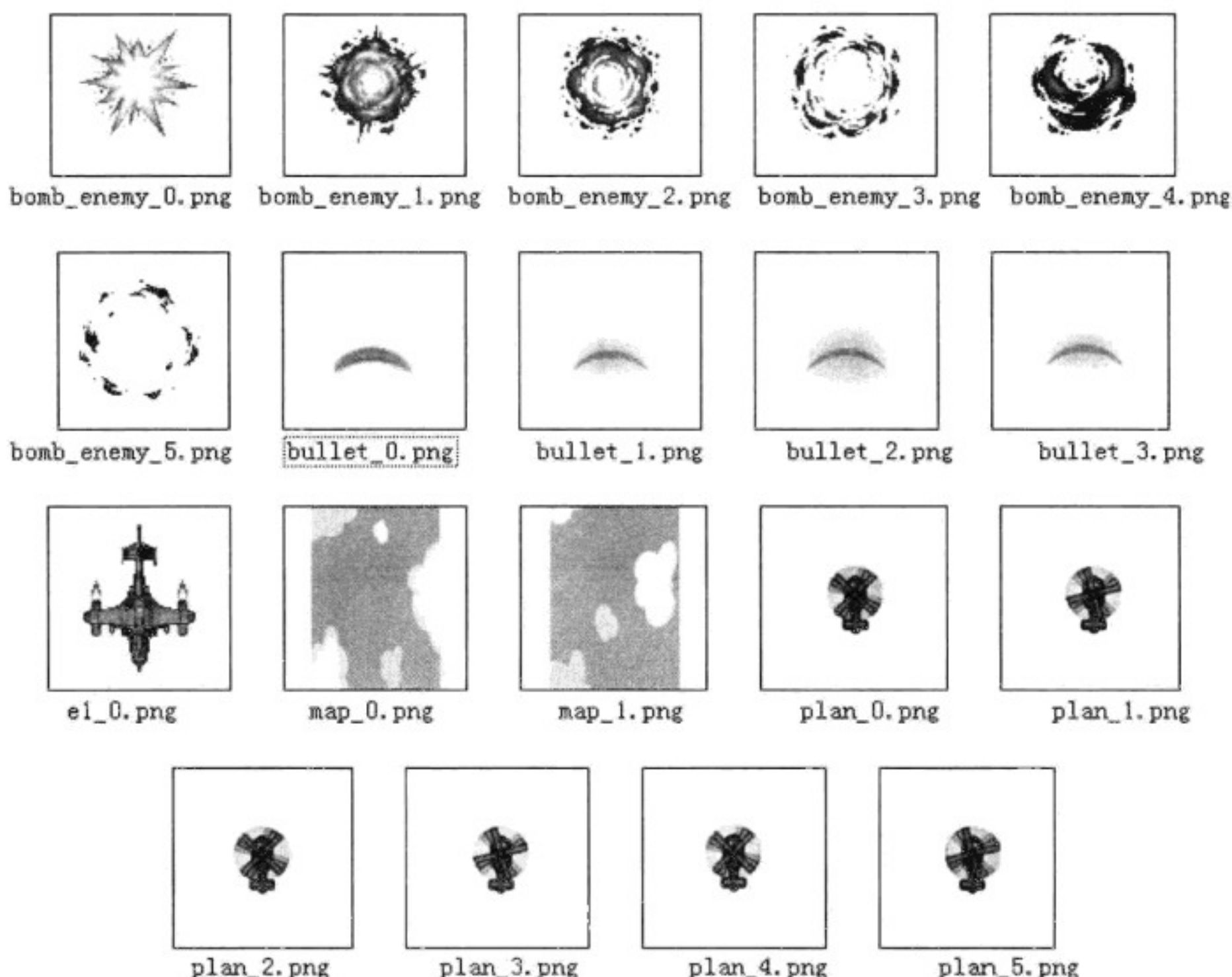


图 6-2 相关图片素材

6.2.2 地图滚动的原理实现

举个简单的例子，大家都坐过火车，坐火车的时候都遇到过这种情况：自己乘坐的火车明明是静止的，但是旁边铁轨上的火车在向后行驶，这样就会有一种错觉感觉自己的火车在向前行驶。飞行射击类游戏的地图原理和这种情况完全一样。玩家在控制飞机在屏幕中飞行位置的同时，背景图片一直在向后滚动，从而给玩家一种错觉感觉自己控制的飞机在向前飞行。如图 6-3 所示，两张地图图片（map_0.png 和 map_1.png）在屏幕背后交替滚动，这样就会给玩家产生自己控制的飞机在向前移动的错觉。

地图滚动的相关代码如下：

```
private void updateBg() {
    /** 更新游戏背景图片实现向下滚动效果 */
    mBitposY0 += 10;    //第一张地图 map_0.png 的纵坐标下移 10 个像素
    mBitposY1 += 10;    //第二张地图 map_1.png 的纵坐标下移 10 个像素
    if (mBitposY0 == mScreenHeight) { //超过游戏屏幕的底边
        mBitposY0 = -mScreenHeight; //回到屏幕上方
    }
    if (mBitposY1 == mScreenHeight) { //超过游戏屏幕的底边
        mBitposY1 = -mScreenHeight; //回到屏幕上方
    }
    ...
}
```

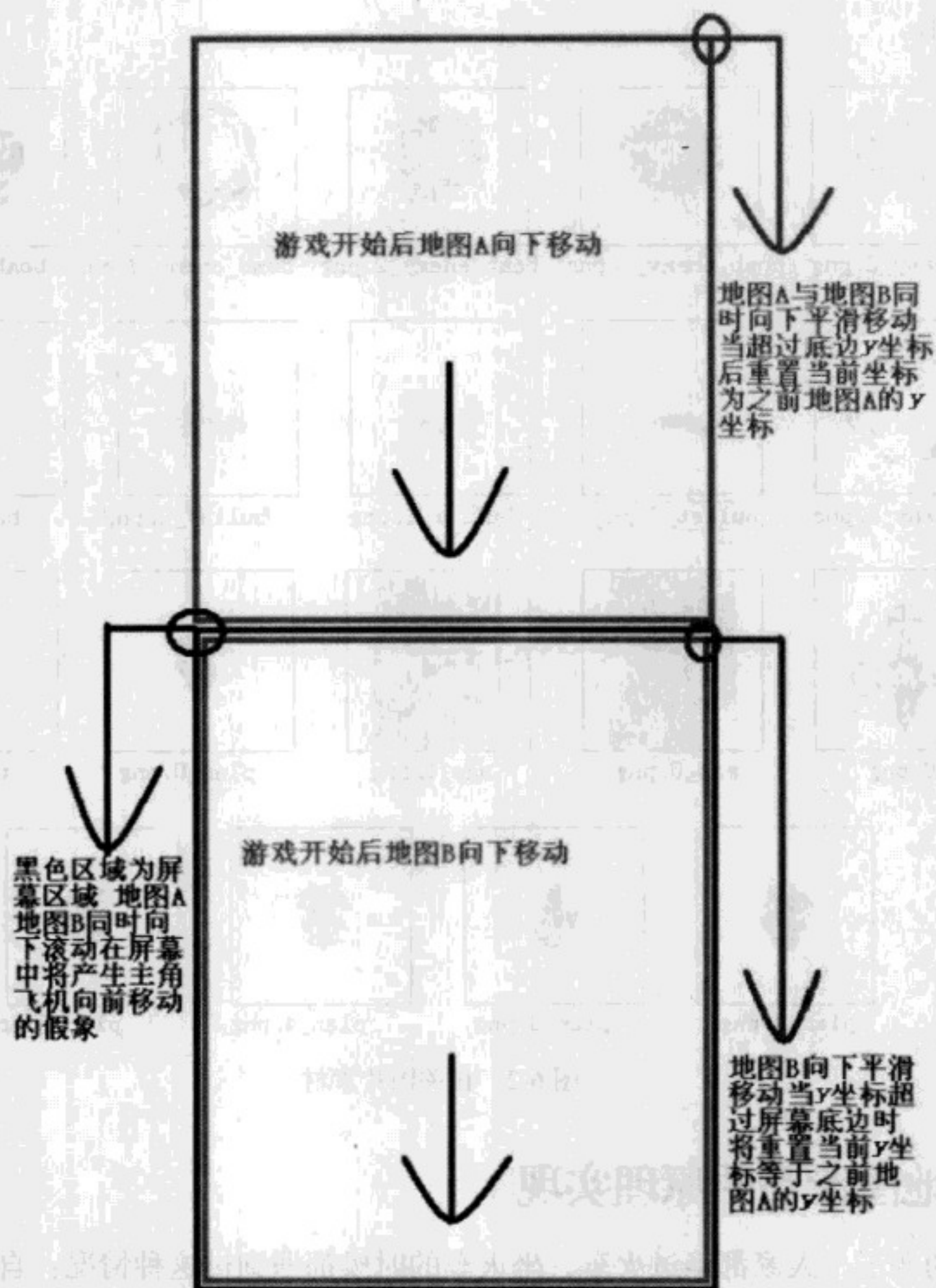



图 6-3 地图滚动的原理

6.2.3 飞机和子弹的实现

游戏中使用到的飞机、子弹均采用对应的类实现。因为子弹的数量很多，敌机的数量也会很多，所以每一颗子弹需要用一个对象来记录当前子弹在屏幕中的绘制区域的 (x,y) 坐标。每一架敌机也是一个对象，也记录了它在屏幕中的绘制区域的 (x,y) 坐标。这样在处理碰撞的时候其实就是每一颗子弹的矩形区域与每一架敌机的矩形区域的碰撞。通过遍历子弹对象与敌机对象就可以计算出碰撞的结果，从而得到碰撞的敌机对象播放死亡爆炸的动画。

如果按照这样的思路将会频繁地创建子弹对象与敌机对象，这样会造成内存泄漏等严重的问题。仔细想一下屏幕中需要绘制的子弹数量与敌机数量应该是有限的，我们可以初始化固定数量的子弹对象与敌机对象，只对这些对象进行更新逻辑与绘制。

举个例子，当前游戏屏幕中最多需要 5 架敌机，代码中就只产生 5 个敌机对象，分别检测这些对象，如果被子弹打中或者向下超过屏幕底边，这时候可以对这个对象进行属性的重置，让这架飞机重新出现在上方的战场上，这样就实现了在不增加飞机对象的情况下让玩家感觉有打不完

的飞机的情形, 子弹对象同理。

游戏开始时将所有飞机、子弹对象全部初始化, 也就是说游戏中不会再分配新对象内存。

游戏过程中, 需要不断更新游戏背景图片位置, 下移 10 个像素, 实现向下滚动的效果。更新子弹位置, 每次 15 个像素。更新敌机位置, 每次 5 个像素, 敌机死亡且爆炸动画结束或者敌机超过屏幕还未死亡则重置敌机坐标位置。同时每隔 500ms 添加一发子弹并初始化其位置坐标在玩家飞机前方 (注意 15 个子弹对象虽然已经存在, 但是被初始化后在屏幕外不能显示出来)。最后调用 Collision() 方法检测子弹与敌机的碰撞。

```
private void updateBg() {
    /**更新游戏背景图片实现向下滚动的效果* */
    ...
    /**更新每发子弹的位置坐标, 上移 15 个像素* */
    for (int i = 0; i < BULLET_POOL_COUNT; i++) {
        mBuilet[i].UpdateBullet();
    }
    /**更新敌机位置坐标* */
    for (int i = 0; i < ENEMY_POOL_COUNT; i++) {
        mEnemy[i].UpdateEnemy();
        /*敌机死亡且爆炸动画结束或者敌机超过屏幕还未死亡则重置坐标*/
        if (mEnemy[i].mAnimState == Enemy.ENEMY_DEATH_STATE
            && mEnemy[i].mPlayID == 6 || mEnemy[i].m_posY >= mScreenHeight) {
            mEnemy[i].init(UtilRandom(0, ENEMY_POOL_COUNT) * ENEMY_POS_OFF, 0);
        }
    }
    /**根据时间初始化发射的子弹位置* */
    if (mSendId < BULLET_POOL_COUNT) {
        long now = System.currentTimeMillis();
        if (now - mSendTime >= PLAN_TIME) {
            mBuilet[mSendId].init(mAirPosX - BULLET_LEFT_OFFSET, mAirPosY - BULLET_UP_OFFSET);
            mSendTime = now;
            mSendId++;
        }
    } else {
        mSendId = 0;
    }
    //子弹与敌人的碰撞检测
    Collision();
}
```

6.2.4 主角飞机子弹与敌机的碰撞检测

将所有子弹对象的矩形区域与敌机对象的矩形区域逐一检测, 如果重叠则说明子弹与敌机发生了碰撞。

```
public void Collision() {
    //子弹与敌人的碰撞检测
    for (int i = 0; i < BULLET_POOL_COUNT; i++) {
        for (int j = 0; j < ENEMY_POOL_COUNT; j++) {
            if (mBuilet[i].m_posX >= mEnemy[j].m_posX
                && mBuilet[i].m_posX <= mEnemy[j].m_posX + 30
                && mBuilet[i].m_posY >= mEnemy[j].m_posY
                && mBuilet[i].m_posY <= mEnemy[j].m_posY + 30
```



```

        ) //发生碰撞敌人的状态修改为死亡
        {
            mEnemy[j].mAnimState = Enemy.ENEMY_DEATH_STATE;
        }
    }
}

```

6.3 关键技术

6.3.1 多线程

多线程编程可以使程序具有两个或两个以上的并发执行任务，就像日常工作中由多人同时合作完成一项任务。这在很多情况下可以改善程序的响应性能，提高资源的利用效率，在多核 CPU 年代，这显得尤为重要。

Java 提供的多线程功能使得在一个程序里可同时执行多个小任务，CPU 在线程间的切换非常迅速，使人们感觉到所有线程好像是在同时进行似的。多线程带来的更大好处是更好的交互性能和实时控制性能，当然，实时控制性能还取决于操作系统本身。

每个 Java 程序都有一个默认的主线程，对于 Application，主线程是 main()方法执行的代码。要想实现多线程，必须在主线程中创建新的线程对象。Java 语言使用 Thread 类及其子类对象来表示线程，新建的线程在它的一个完整生命周期中通常要经历以下 5 种状态。

1. 新建

当一个 Thread 类或其子类的对象被声明并创建时，新生的线程对象处于新建状态。此时它已经有了相应的内存空间和其他资源，并已被初始化。

2. 就绪

处于新建状态的线程被启动后，将进入线程队列排队等待 CPU 时间片，此时它已经具备了运行的条件，一旦轮到它来享用 CPU 资源时，就可以脱离创建它的主线程独立开始自己的生命周期了。另外，原来处于阻塞状态的线程被解除阻塞后也将进入就绪状态。

3. 运行

当就绪状态的线程被调度并获得处理器资源时，便进入运行状态。每一个 Thread 类及其子类的对象都有一个重要的 run()方法，当线程对象被调度执行时，它将自动调用本对象的 run()方法，从第一句开始顺序执行。run()方法定义了这一类线程的操作和功能。

4. 阻塞

一个正在执行的线程如果在某些特殊情况下，如被人为挂起或者需要执行费时的输入输出操作时，将让出 CPU 并暂时终止自己的执行，进入阻塞状态。阻塞时它不能进入排队队列，只有当引起阻塞的原因被消除时，线程才可以转入就绪状态，重新进到线程队列中排队等待 CPU 资源，以便从原来终止处开始继续执行。

5. 死亡

处于死亡状态的线程不具有继续运行的能力。线程死亡的原因有两个：一个是正常运行的线程完成了它的全部工作，即执行完了 run()方法的最后一条语句并退出；另一个原因是线程被提前

强制性的终止，如通过执行 `stop()` 或 `destroy()` 方法终止线程。

由于线程与进程一样是一个动态的概念，所以它也像进程一样有一个从产生到消亡的生命周期，如图 6-4 所示。

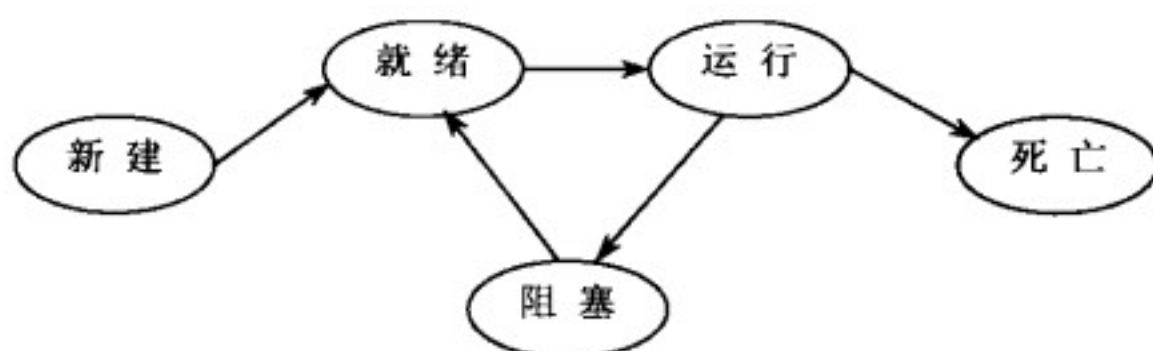


图 6-4 线程状态的变化

线程在各个状态之间的转换及线程生命周期的演进是由系统运行的状况、同时存在的其他线程和线程本身的算法所共同决定的。在创建和使用线程时应注意利用线程的方法宏观地控制这个过程。

6.3.2 Java 的 Thread 类和 Runnable 接口

Java 中编程实现多线程的应用有两种途径：一种是创建用户自己的 Thread 线程子类，一种是在用户自己的类中实现 Runnable 接口。

1. Thread 类

Thread 类是一个具体的类，该类封装了线程的属性和行为。

Thread 类的构造函数有多个，较常用的有如下几个。

(1) `public Thread()`:

这个方法创建了一个默认的线程类的对象。

(2) `public Thread(Runnable target)`:

这个方法在上一个构造函数的基础上，利用一个实现了 Runnable 接口参数对象 Target 中所定义的 `run()` 方法，以便初始化或覆盖新创建的线程对象的 `run()` 方法。

(3) `public Thread(String name)`:

这个方法在第一个构造函数创建一个线程的基础上，利用一个 String 类的对象 name 为所创建的线程对象指定了一个字符串名称供以后使用。

Thread 类的主要方法如下。

(1) 启动线程的 `start()` 方法: `public void start()`

`start()` 方法将启动线程对象，使之从新建状态转入就绪状态并进入就绪队列排队。

(2) 定义线程操作的 `run()` 方法: `public void run()`

Thread 类的 `run()` 方法是用来定义线程对象被调用之后所执行的操作，都是系统自动调用而用户程序不能引用的方法。系统的 Thread 类中，`run()` 方法没有具体内容，因此用户程序需要创建自己的 Thread 类的子类，并定义新的 `run()` 方法来覆盖原来的 `run()` 方法。

`run()` 方法将运行线程，使之从就绪队列状态转入运行状态。

(3) 使线程暂时休眠的 `sleep()` 方法: `public static void sleep(long millis) throws InterruptedException` `millis` 是以 ms 为单位的休眠时间。

线程的调度执行是按照其优先级的高低顺序进行的，当高级线程未完成即未死亡时，低级线程没有机会获得处理器。优先级高的线程可以在它的 `run()` 方法中调用 `sleep()` 方法来使自己放弃处

理器资源，休眠一段时间。休眠时间的长短由 `sleep()` 方法的参数决定。进入休眠的线程仍处于活动状态，但不被调度运行，直到休眠期满它可以被另一个线程用中断唤醒。如果被另一个线程唤醒，则会抛出 `InterruptedException` 异常。

(4) 终止线程的 `stop()` 方法：

```
public final void stop();
public final void stop(Throwable obj)
```

程序中需要强制终止某线程的生命周期时可以使用 `stop()` 方法。`stop()` 方法可以由线程在自己的 `run()` 方法中调用，也可以由其他线程在其执行过程中调用。

`stop()` 方法将会使线程由其他状态进入死亡状态。

(5) 判断线程是否未消亡的 `isAlive()` 方法：`public final native Boolean isAlive()`

在调用 `stop()` 方法终止一个线程之前，最好先用 `isAlive()` 方法检查一下该线程是否仍然存活，终止不存在的线程可能会造成系统错误。

如果一个类直接或间接继承自 `Thread` 类，则该类对象便具有了线程的能力。这是最简单的开发自己线程的方式，采用此方式最重要的是重写继承的 `run()` 方法。其实，`run()` 方法中的代码就是线程所要执行任务的描述，这种方式的基本语法如下：

```
class <类名> extends Thread
{
    public void run()
    {
        //线程所要执行任务的代码
    }
}
```

上述格式中，`run()` 方法中编写的是线程所要执行任务的代码，一旦线程启动，`run()` 方法中的代码将成为一项独立的执行任务。

使用 `Thread` 类的子类创建一个线程，程序员必须创建一个从 `Thread` 类导出的新类，并且必须覆盖 `Thread` 类的 `run()` 方法来完成所需要的工作。用户并不直接调用此 `run()` 方法，而是必须调用 `Thread` 类的 `start()` 方法，该方法再调用 `run()` 方法。

下面是用于显示时间的多线程程序。

```
import java.util.*;

class TimePrinter extends Thread {    //定义了 Thread 类的子类 TimePrinter 类
    int pauseTime;
    String name;
    public TimePrinter(int x, String n) {    //构造函数
        pauseTime = x;
        name = n;
    }
    public void run() {    //用户重载了 run() 方法，定义了线程的任务
        while(true) {
            try {
                System.out.println(name + ":" + new
                    Date(System.currentTimeMillis()));
                Thread.sleep(pauseTime);
            } catch (Exception e) {    //有可能抛出线程休眠被中断的异常
                System.out.println(e);
            }
        }
    }
}
```



```

    }
}
public static void main(String args[]) {
    TimePrinter tp1 = new TimePrinter(1000, "Fast Guy");//线程的创建
    tp1.start(); //线程的启动
    TimePrinter tp2 = new TimePrinter(3000, "Slow Guy");
    tp2.start();
}
}

```

这个程序是 Java Application，其中定义了一个 Thread 类的子类 TimePrinter。在 TimePrinter 类中重载了 Thread 类中的 run() 方法，用来显示当前时间，并且休眠一段时间。为了防止在休眠的时候被打断，用了一个 try-catch 块进行了异常处理。在 TimePrinter 类中的 main() 方法根据不同的参数创建了两个新的线程 Fast Guy 和 Slow Guy 并分别启动它们，则这两个线程将轮流运行。

2. Runnable 接口

Runnable 接口只有一个方法 run()，所有实现 Runnable 接口的用户类都必须具体实现这个 run() 方法，为它书写方法体并定义具体操作。当线程转入运行状态时，它所执行的就是 run() 方法中规定的操作。

下面给出了一个实现了 Runnable 接口的类，代码如下：

```

//实现了 Runnable 接口的类
class MyRunnable implements Runnable
{
    //重写 run() 方法
    public void run()
    {
        //线程所要执行任务的代码
    }
}

```

可以通过实现 Runnable 接口的方法来定义用户线程的操作。Runnable 接口只有一个方法 run()，实现这个接口，就必须定义 run() 方法的具体内容，用户新建线程的操作也由这个方法来决定。实现 Runnable 接口的类来创建线程的过程如下：

```

MyRunnable mr=new MyRunnable();//创建 Runnable 接口实现类的对象
Thread t=new Thread(mr); //创建 Thread 对象
t.run();//调用 Thread 对象中的 run() 方法

```

下面是采用 Runnable 接口的方法，实现显示时间的多线程程序。

```

import java.util.*;
class TimePrinter implements Runnable{//定义实现了 Runnable 接口的子类
    int pauseTime;
    String name;
    public TimePrinter(int x, String n) { //构造函数
        pauseTime = x;
        name = n;
    }
    public void run() { //用户重载了 run() 方法，定义了线程的任务
        while(true) {
            try {
                System.out.println(name + ":" + new

```



```

        Date(System.currentTimeMillis()));
        Thread.sleep(pauseTime);
    } catch (Exception e) { //有可能抛出线程休眠被中断的异常
        System.out.println(e);
    }
}

static public void main(String args[]) {
    Thread t1 = new Thread(new TimePrinter(1000, "Fast Guy"));
    t1.start();
    Thread t2 = new Thread(new TimePrinter(3000, "Slow Guy"));
    t2.start(); //线程的启动
}
}

```

这个程序实现了前面程序的相同功能，其他方面都是相同的，唯一不同的地方是前面程序中使用了继承 Thread 类的方法，而这个程序中使用了实现 Runnable 接口的方式，它们最后运行的效果也是完全一样的。可见用这两种方式实现多线程的程序效果是相同的。

3. 两种方式的比较

无论使用哪种方式，都可以通过一定的操作得到一项独立的执行任务，然而两者之间不是完全相同的，下面对两者之间的异同进行了比较。

(1) 继承 Thread 类的方式虽然简单，但继承了该类就不能继承其他类，这在有些情况下会严重影响开发。其实，很多情况下开发人员只是希望自己的类具有线程的能力，能扮演线程的角色，而自己的类还需要继承其他类。

(2) 实现 Runnable 接口既不影响继承其他类，也不影响实现其他接口，只是实现 Runnable 接口的类多扮演了一种角色，多了一种能力而已，灵活性更好。

实际开发中继承 Thread 类的情况没有实现 Runnable 接口的多，因为后者具有更大的灵活性，可扩展性强。因此本章雷电飞机游戏采用 Runnable 接口实现多线程。

6.4 雷电飞机游戏设计的步骤

6.4.1 设计子弹类 (Bullet.java)

在项目中创建一个 Bullet 类，用于表示子弹，实现子弹坐标更新以及绘制功能。

导入包及相关类：

```

import java.awt.*;
import java.awt.image.ImageObserver;
import javax.swing.JFrame;
import javax.swing.JPanel;

```

子弹类的构造方法中，加载了子弹的 4 帧状态图片（如图 6-5 所示），在绘制子弹时，根据 mPlayID 的值绘制相应状态的图片。

```

//子弹类
public class Bullet {
    /**子弹的 x 轴速度**/
    static final int BULLET_STEP_X = 3;

```



```

/**子弹的 y 轴速度**/
static final int BULLET_STEP_Y = 15;
/**子弹图片的宽度**/
static final int BULLET_WIDTH = 40;
/**子弹的 (x,y) 坐标**/

```

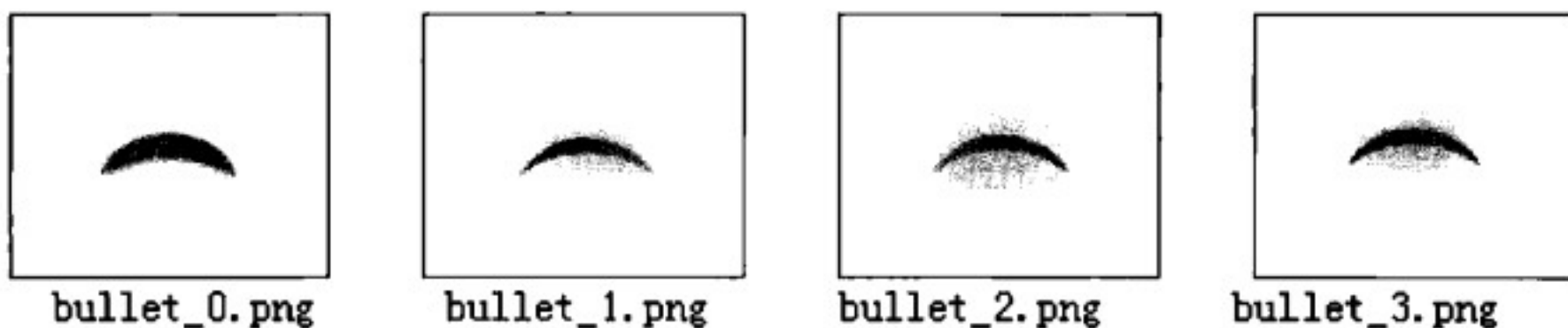


图 6-5 子弹图片

```

public int m_posX = 0;
public int m_posY = -20;    //初始时子弹在游戏屏幕外
/**是否更新绘制子弹**/
boolean mFacus = true;
private Image pic[] = null; //子弹图片数组
/**当前帧的 ID **/
private int mPlayID = 0;
public Bullet() {
    pic = new Image[4];
    for (int i = 0; i < 4; i++)
        pic[i] = Toolkit.getDefaultToolkit().getImage(
            "images\\bullet_" + i + ".png");
}
/**初始化坐标**/
public void init(int x, int y) {
    m_posX = x;
    m_posY = y;
    mFacus = true;
}
/**绘制子弹**/
public void DrawBullet(Graphics g, JPanel i)
{
    g.drawImage(pic[mPlayID++], m_posX, m_posY, (ImageObserver) i);
    if(mPlayID==4) mPlayID=0;
}

```

子弹坐标更新主要修改 y 坐标（垂直方向）值，每次 15 个像素。当然也可以修改 x 坐标（水平方向）值，这里为了简单起见没修改 x 坐标值（水平方向）。

```

/**更新子弹的坐标点**/
public void UpdateBullet() {
    if (mFacus)
        m_posY -= BULLET_STEP_Y;
}
}

```

6.4.2 设计敌机类（Enemy.java）

在项目中创建一个 Enemy 类，用于表示敌机，实现子弹坐标更新以及绘制功能，其功能与子

弹类相似。

导入包及相关类：

```
import java.awt.*;
import java.awt.image.ImageObserver;
import java.io.File;
import javax.swing.JPanel;
```

敌机类中定义的一些数据成员：

```
public class Enemy { //敌机类
    /**敌机存活状态*/
    public static final int ENEMY_ALIVE_STATE = 0;
    /**敌机死亡状态*/
    public static final int ENEMY_DEATH_STATE = 1;
    /**敌机行走的 y 轴速度*/
    static final int ENEMY_STEP_Y = 5;
    /**敌机的 (x,y) 坐标 */
    public int m_posX = 0;
    public int m_posY = 0;
    /**敌机状态*/
    public int mAnimState = ENEMY_ALIVE_STATE; //敌机最初为存活状态
    private Image enemyExplorePic[] = new Image[6]; //敌机爆炸图片数组
    /**当前帧的 ID */
    public int mPlayID = 0;
```

敌机类的构造方法中，加载敌机爆炸时的 6 帧状态图片，在绘制爆炸时，根据 mPlayID 的值绘制相应爆炸状态的图片（如图 6-6 所示）。敌机本身为存活状态时仅有一种状态图片，不需要切换。

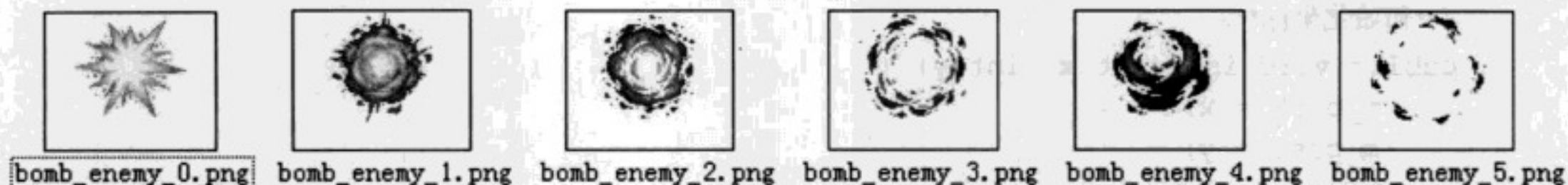


图 6-6 敌机爆炸时的 6 帧图片

```
public Enemy() {
    for (int i = 0; i < 6; i++)
        enemyExplorePic[i] = Toolkit.getDefaultToolkit().getImage(
            "images\\bomb_enemy_" + i + ".png");
}
/**初始化坐标*/
public void init(int x, int y) {
    m_posX = x;
    m_posY = y;
    mAnimState = ENEMY_ALIVE_STATE;
    mPlayID = 0;
}
/**绘制敌机*/
public void DrawEnemy(Graphics g, JPanel i)
{
    //当敌机状态为死亡且死亡动画播放完毕，则不再绘制敌机
    if(mAnimState == ENEMY_DEATH_STATE && mPlayID<6) {
        g.drawImage(enemyExplorePic[mPlayID],m_posX,m_posY,(ImageObserver)i);
        mPlayID++;
    }
```



```

        return;
    }
    //当敌机状态为存活状态
    Image pic = Toolkit.getDefaultToolkit().getImage("images/el_0.png");
    g.drawImage(pic,m_posX,m_posY,(ImageObserver)i);
}

```

敌机坐标更新主要修改 y 坐标（垂直方向）值，每次 5 个像素。当然也可以修改 x 坐标（水平方向）值，这里为了简单起见没修改敌机 x 坐标值（水平方向）。

```

/**更新敌机坐标*/
public void UpdateEnemy() {
    m_posY += ENEMY_STEP_Y;
}
}

```

6.4.3 设计游戏界面类（GamePanel.java）

在项目中创建一个继承 JPanel 的 GamePanel 类，用于实现游戏界面，完成子弹发射、敌机移动、碰撞检测等功能。

导入包及相关类：

```

import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.File;
import java.io.IOException;
import java.util.Random;
import javax.imageio.ImageIO;
import javax.swing.*;

```

继承 JPanel 的 GamePanel 类实现了 KeyListener 接口，实现了键盘事件监听；同时实现了 Runnable 接口，实现多线程来更新游戏画面。

```

public class GamePanel extends JPanel implements Runnable, KeyListener {
    /**屏幕的宽高*/
    private int mScreenWidth = 320;
    private int mScreenHeight = 480;
    /**游戏主菜单状态*/
    private static final int STATE_GAME = 0;
    /**游戏状态*/
    private int mState = STATE_GAME;
    /**游戏背景资源：两张图片进行切换让屏幕滚动起来*/
    private Image mBitMenuBG0 = null;
    private Image mBitMenuBG1 = null;
    /**记录两张背景图片更新时的 y 坐标*/
    private int mBitposY0 = 0;
    private int mBitposY1 = 0;
    /**子弹对象的数量*/
    final static int BULLET_POOL_COUNT = 15;
    /**飞机移动步长*/
    final static int PLAN_STEP = 10;
    /**每 500ms 发射一颗子弹*/
    final static int PLAN_TIME = 500;
    /**敌人对象的数量*/

```



```

final static int ENEMY_POOL_COUNT = 5;
/**敌人飞机偏移量**/
final static int ENEMY_POS_OFF = 65;
/**游戏主线程**/
private Thread mThread = null;
/**线程循环标志**/
private boolean mIsRunning = false;
/**飞机在屏幕中的坐标**/
public int mAirPosX = 0;
public int mAirPosY = 0;

/**敌机对象数组**/
Enemy mEnemy[] = null;
/**子弹对象数组**/
Bullet mBULLET[] = null;
/**初始化发射子弹 ID**/
public int mSendId = 0;
/**上一颗子弹发射的时间**/
public Long mSendTime = 0L;
Image myPlanePic[];    /**玩家飞机的所有图片**/
public int myPlaneID = 0; /**玩家飞机当前的帧号**/

```

GamePanel 类构造方法设置游戏屏幕区域为 320×480 像素大小,调用 init()方法初始化各种对象,最后启动游戏线程。

```

/**
 * 构造方法
 */
public GamePanel() {
    setPreferredSize(new Dimension(mScreenWidth, mScreenHeight));
    //设定焦点在本窗体并赋予监听对象
    setFocusable(true);
    addKeyListener(this);
    init();
    setGameState(STATE_GAME);
    mIsRunning = true;
    mThread = new Thread(this); //实例线程
    /**启动游戏线程**/
    mThread.start();
    setVisible(true);
}

```

游戏线程主要执行 run()方法,每延时 0.1s 后调用 Draw()方法刷新游戏屏幕。

```

public void run() { //重写 run()方法
    while (mIsRunning) {
        /**刷新屏幕**/
        Draw();
        //延时 0.1s
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```



```

    }
}

```

init()方法初始化各种对象,包括两张背景图片 mBitMenuBG0 和 mBitMenuBG1,通过这两张背景图片的切换实现游戏背景动态移动的效果。初始化玩家飞机的坐标在(150,400)点处。创建5个敌方飞机对象 mEnemy 数组,同时创建15个子弹类对象 mBulet 数组。

```

private void init() {
    /**游戏背景* */
    try {
        mBitMenuBG0 = Toolkit.getDefaultToolkit().getImage("images\\map_0.png");
        mBitMenuBG1 = Toolkit.getDefaultToolkit().getImage("images\\map_1.png");
    } catch (IOException e) {
        e.printStackTrace();
    }
    /**第一张图片紧贴在屏幕(0,0)点处,第二张图片在第一张图片上方* */
    mBitposY0 = 0;
    mBitposY1 = -mScreenHeight;
    /**初始化玩家飞机的坐标* */
    mAirPosX = 150;
    mAirPosY = 400;
    /**初始化玩家飞机相关的6张图片对象* */
    myPlanePic = new Image[6];
    for (int i = 0; i < 6; i++)
        myPlanePic[i] = Toolkit.getDefaultToolkit().getImage(
            "images\\plan_" + i + ".png");
    /**创建敌机对象* */
    mEnemy = new Enemy[ENEMY_POOL_COUNT];
    for (int i = 0; i < ENEMY_POOL_COUNT; i++) {
        mEnemy[i] = new Enemy();
        mEnemy[i].init(i * ENEMY_POS_OFF, i * ENEMY_POS_OFF-300);
    }
    /**创建子弹类对象* */
    mBulet = new Bullet[BULLET_POOL_COUNT];
    for (int i = 0; i < BULLET_POOL_COUNT; i++) {
        mBulet[i] = new Bullet();
    }
    mSendTime = System.currentTimeMillis();
}

```

Draw()方法绘制游戏界面(包括背景、敌我飞机、子弹),更新游戏逻辑。

```

protected void Draw() {
    switch (mState) {
        case STATE_GAME:
            renderBg();//绘制游戏界面(包括背景、敌我飞机、子弹)
            updateBg();//更新游戏逻辑
            break;
    }
}
private void setGameState(int newState) {
    mState = newState;
}

```

renderBg()方法首先更新玩家飞机帧号,绘制游戏地图(由两幅图片组成),按帧号绘制玩家自己飞机和所有子弹的动画及所有敌方飞机的动画。玩家飞机的6帧图片如图6-7所示。

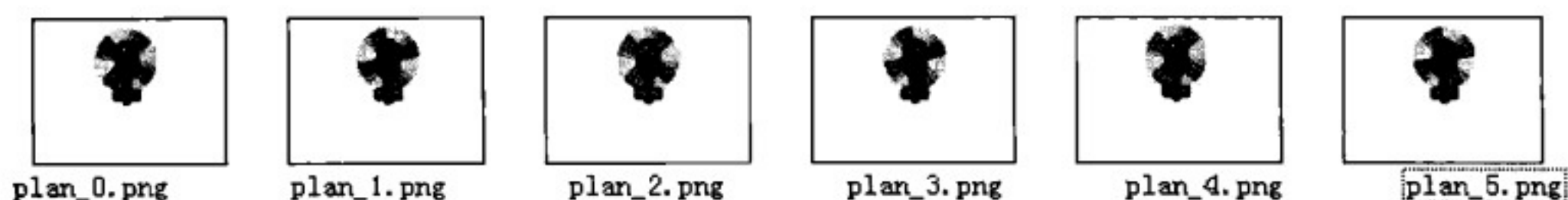


图 6-7 玩家飞机状态图

```

public void renderBg() {
    myPlaneID++;
    if (myPlaneID == 6)
        myPlaneID = 0;
    repaint();
}

public void paint(Graphics g) {
    /**绘制游戏地图**/
    g.drawImage(mBitMenuBG0, 0, mBitposY0, this);
    g.drawImage(mBitMenuBG1, 0, mBitposY1, this);
    /**绘制自己飞机动画**/
    g.drawImage(myPlanePic[myPlaneID], mAirPosX, mAirPosY, this);
    /**绘制子弹动画**/
    for (int i = 0; i < BULLET_POOL_COUNT; i++)
        mBuilet[i].DrawBullet(g, this);
    /**绘制敌机动画**/
    for (int i = 0; i < ENEMY_POOL_COUNT; i++)
        mEnemy[i].DrawEnemy(g, this);
}

```

`updateBg()`方法更新游戏逻辑。更新游戏背景图片位置,下移 10 个像素,实现向下滚动效果。更新子弹位置,每次 15 个像素。更新敌机位置,每次 5 个像素,敌机死亡或者敌机超过屏幕还未死亡,则重置敌机坐标位置。同时每隔 500ms 添加一发子弹并初始化其位置坐标。最后调用 `Collision()`方法检测子弹与敌机的碰撞。

```

private void updateBg() {
    /**更新游戏背景图片实现向下滚动效果**/
    mBitposY0 += 10;
    mBitposY1 += 10;
    if (mBitposY0 == mScreenHeight) {
        mBitposY0 = -mScreenHeight;
    }
    if (mBitposY1 == mScreenHeight) {
        mBitposY1 = -mScreenHeight;
    }
    /**更新子弹位置**/
    for (int i = 0; i < BULLET_POOL_COUNT; i++) {
        mBuilet[i].UpdateBullet();
    }
    /**更新敌机位置**/
    for (int i = 0; i < ENEMY_POOL_COUNT; i++) {
        mEnemy[i].UpdateEnemy();
        /**敌机死亡或者敌机超过屏幕还未死亡,则重置坐标* */
        if (mEnemy[i].mAnimState == Enemy.ENEMY_DEATH_STATE
            && mEnemy[i].mPlayID == 6 || mEnemy[i].m_posY >= mScreenHeight) {
            mEnemy[i].init(UtilRandom(0, ENEMY_POOL_COUNT) * ENEMY_POS_OFF, 0);
        }
    }
}

```



```

    }
    /**根据时间初始化将要发射的子弹位置在玩家飞机前方*/
    if (mSendId < BULLET_POOL_COUNT) {
        long now = System.currentTimeMillis();
        if (now - mSendTime >= PLAN_TIME) {
            //每过 500ms 发射一颗子弹, 此子弹位置在玩家飞机前方
            mBuilet[mSendId].init(mAirPosX - BULLET_LEFT_OFFSET, mAirPosY -
BULLET_UP_OFFSET);
            mSendTime = now;
            mSendId++;
        }
    } else {
        mSendId = 0;
    }
    Collision(); //子弹与敌机的碰撞检测
}

```

Collision()方法检测子弹与敌机的碰撞。

```

public void Collision() { //子弹与敌机的碰撞检测
    for (int i = 0; i < BULLET_POOL_COUNT; i++) {
        for (int j = 0; j < ENEMY_POOL_COUNT; j++) {
            if (mBuilet[i].m_posX >= mEnemy[j].m_posX
                && mBuilet[i].m_posX <= mEnemy[j].m_posX + 30
                && mBuilet[i].m_posY >= mEnemy[j].m_posY
                && mBuilet[i].m_posY <= mEnemy[j].m_posY + 30
            ) {
                mEnemy[j].mAnimState = Enemy.ENEMY_DEATH_STATE;
            }
        }
    }
}

```

UtilRandom(int botton, int top)方法返回 (botton, top) 区间的一个随机数。

```

private int UtilRandom(int botton, int top) {
    return ((Math.abs(new Random().nextInt()) % (top - botton)) + botton);
}

```

keyPressed(KeyEvent e)事件响应用户的按键操作, 修改玩家自己飞机的坐标 (mAirPosX, mAirPosY)。keyPressed 事件检测什么键被按下, 假如向上键被按下则玩家自己飞机垂直坐标 mAirPosY 减少 PLAN_STEP (10 个像素), 其他方向同理。同时控制玩家飞机不能超出游戏区域的左右边界, 如果超出左边界则 mAirPosX = 0; 如果超出右边界则 mAirPosX = mScreenWidth - 30。

```

public void keyPressed(KeyEvent e) {
    int key = e.getKeyCode();
    System.out.println(key);
    if (key == KeyEvent.VK_UP) //假如向上键被按下
        mAirPosY -= PLAN_STEP;
    if (key == KeyEvent.VK_DOWN) //假如向下键被按下
        mAirPosY += PLAN_STEP;
    if (key == KeyEvent.VK_LEFT) //假如向左键被按下
    {
        mAirPosX -= PLAN_STEP;
        if (mAirPosX < 0) //超出左边界
            mAirPosX = 0;
    }
}

```



```
    }
    if (key == KeyEvent.VK_RIGHT)//假如向右键被按下
    {
        mAirPosX += PLAN_STEP;
        if (mAirPosX > mScreenWidth - 30)//超出右边界
            mAirPosX = mScreenWidth - 30;
        System.out.println(mAirPosX + ":" + mAirPosY);
    }
}
```

6.4.4 设计游戏窗口类 (planeFrame.java)

在项目中创建一个继承 JFrame 的 planeFrame 类，用于显示自定义游戏面板界面。

```
import java.awt.Container;
import javax.swing.JFrame;
public class planeFrame extends JFrame {
    public planeFrame() {
        setTitle("飞行射击类游戏"); //窗体标题
        //获得自定义面板的实例
        GamePanel panel = new GamePanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
        pack();
    }
    public static void main(String[] args) {
        planeFrame e1 = new planeFrame();
        //设定允许窗体关闭操作
        e1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //显示窗体
        e1.setVisible(true);
    }
}
```


第 7 章

21 点扑克牌游戏

7.1 21 点扑克牌游戏介绍

21 点游戏是玩家要取得比庄家更大的点数总和，但点数超过 21 点即为输牌。J、Q、K 算 10 点，A 可算 1 点，其余按牌面值计点数。开始时每人发两张牌，一张明，一张暗，凡点数不足 21 点，可选择继续要牌。

本章开发 21 点扑克牌游戏，游戏运行界面如图 7-1 所示。为简化起见，游戏有两方，一方为 Dealer（庄家），另一方为 Player（玩家），都将发牌。Dealer（庄家）要牌过程由程序自动实现。程序能够判断玩家输赢。



图 7-1 21 点扑克牌游戏运行界面

7.2 关键技术

扑克牌游戏编程关键有两点：一是扑克牌面的绘制；二是扑克游戏规则的算法实现。初学扑克牌游戏编程的爱好者可从一些简单的游戏，借用一些现有资源开始。21 点扑克牌游戏用到了图 7-2 所示的图片素材。

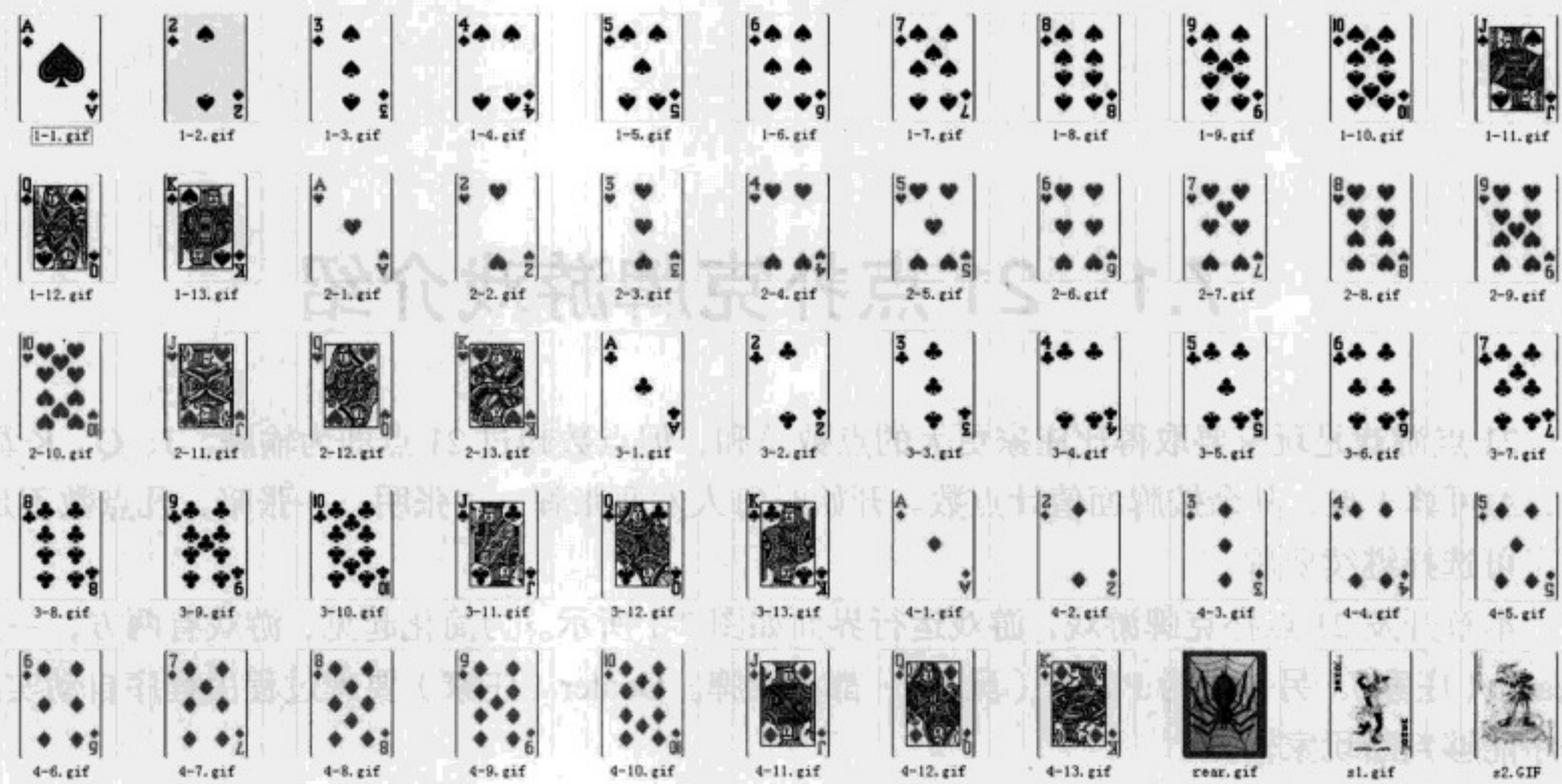


图 7-2 图片素材

7.2.1 扑克牌面绘制

`g.drawImage()`方法绘制扑克牌面。

```
private Image cardImage; //图片
//画牌面
protected void paint(Graphics g)
{
    g.drawImage(cardImage,x,y,null);
}
//画牌面
protected void paint(Graphics g,JPanel i)
{
    g.drawImage(cardImage,x,y, (ImageObserver)i);
}
```

7.2.2 识别牌的点数

如何识别牌的点数，这里采用 `count` 表示。游戏中采用以下代码计算点数：

```
if(value=="J"||value=="Q"||value=="K")
    this.count=10;
else if(value=="A")
    this.count=1;
```



```
else
    this.count=Integer.parseInt(value);
```

7.2.3 Dealer（庄家）要牌的智能实现

Dealer（庄家）要牌过程是电脑自动实现的，这里设计得比较简单，仅判断点数和是否已超过17，如果超过则不要牌了。当然读者可以设计更复杂的游戏逻辑。

7.2.4 游戏规则算法实现

游戏开始时，我们首先要取一副牌（Poker类实现），然后将牌洗好，用xipai()方法洗牌时，应取得2个54以内的随机数，将对应的Card顺序对调，重复500次即可达到洗牌效果。

发牌时从第1张牌开始发起。两家的牌就保存在ArrayList列表中，因此“开始”时首先清空ArrayList列表，同时清空积分（点数和）和输赢标志。得分（点数和）及输赢判断的过程为：calcComputerScore()方法用来计算电脑得分（点数和）。通过遍历存储于电脑中牌的ArrayList列表Computercards，获取每张牌的点数c.count从而得到电脑得分（点数和）。

```
public void calcComputerScore()
{
    public void calcComputerScore() {
        Computerscore = 0;
        for (int i = 0; i < Computercards.size(); i++) {
            Card c = (Card) Computercards.get(i);
            Computerscore += c.count;
        }
    }
}
```

calcComputerScore()方法的作用是计算玩家得分（点数和）。通过遍历存储于玩家手中牌的ArrayList列表Mycards，获取每张牌的点数c.count从而得到玩家得分（点数和）。

```
public void calcMyScore() {
    Myscore = 0;
    for (int i = 0; i < Mycards.size(); i++) {
        Card c = (Card) Mycards.get(i);
        Myscore += c.count;
    }
}
```

shuying()方法用来计算两家的得分，在不超过21点的情况下比较点数判断输赢。

```
public void shuying() {
    calcComputerScore();
    calcMyScore();
    if (Computerlose == false) //超过21点情况下
        JOptionPane.showMessageDialog(null, "电脑输了", "提示",
            JOptionPane.ERROR_MESSAGE);
    if (Mylose == false) //超过21点情况下
        JOptionPane.showMessageDialog(null, "玩家输了", "提示",
            JOptionPane.ERROR_MESSAGE);
    if (Myscore > Computerscore)
        JOptionPane.showMessageDialog(null, "玩家赢了", "提示",
            JOptionPane.ERROR_MESSAGE);
    else
        JOptionPane.showMessageDialog(null, "电脑赢了", "提示",
            JOptionPane.ERROR_MESSAGE);
}
```


7.3 程序设计的步骤

7.3.1 设计扑克牌类 (Card.java)

21 点游戏中, 一张牌只有 4 个属性说明: value (牌面大小)、color (牌面花色), count (点数) 和 cardImage (牌面图片)。因此, 这里用 Card 类来实现。

```
import java.awt.Graphics;
import java.awt.Image;
public class Card {
    private String color; //花色 (梅花、方块、红桃、黑桃)
    private String value; //牌面大小 (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K)
    private Image cardImage; //图片
    public int count; //点数
    private int x=100, y=100; //牌所在位置
    //定义一个构造方法用于初始化点数 2~A 的牌
    public Card(String color, String value, Image bmpcard) {
        this.color = color;
        this.value = value;
        this.cardImage=bmpcard;
        if(value=="J"||value=="Q"||value=="K")
            this.count=10;
        else if(value=="A")
            this.count=1;
        else
            this.count=Integer.parseInt(value);
    }
    //定义一个构造方法用于初始化大王和小王
    public Card(String value, Image bmpcard) {
        this.color = "王";
        this.value = value;
        this.cardImage=bmpcard;
        this.count=0; //大王和小王点数为 0
    }
    //设置牌要显示的位置
    protected void setPosition(int x, int y)
    {
        this.x=x;
        this.y=y;
    }
    //画牌面
    protected void paint(Graphics g)
    {
        g.drawImage(cardImage, x, y, null);
    }
    //在指定的面板上画牌面
    protected void paint(Graphics g, JPanel i)
```



```

{
    g.drawImage(cardImage,x,y, (ImageObserver)i);
}
//取一张牌的花色
public String getcolor() {
    return color;
}
//取一张牌面的大小
public String getvalue() {
    return value;
}
public void print() {
    System.out.print(color);
    System.out.print(value);
}
}

```

7.3.2 设计一副扑克类 (Poker.java)

一副扑克有 54 张牌 (Card)，因此 Poker 类中 Card[] cards 是保存 54 张牌 (Card)。构造方法 Poker() 用于生成每张牌从而初始化这副扑克。游戏开始时，我们首先要取一副牌，然后将牌洗好。用 xipai() 方法洗牌时，应取得 2 个 54 以内的随机数，将对应的 Card 顺序对调。

```

import java.awt.Image;
import java.awt.Toolkit;
public class Poker {
    static Card[] cards = new Card[54];
    static String[] colors = {"黑桃", "红桃", "梅花", "方块"};
    static String values[] = { "A", "2", "3", "4", "5", "6", "7", "8", "9", "10",
        "J", "Q", "K" };
    private Image pic[] = null; //扑克牌图片数组
    public void getPic() {
        pic = new Image[54];
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 13; j++) {
                pic[i * 13 + j] = Toolkit.getDefaultToolkit().getImage(
                    "images\\" + (i + 1) + "-" + (j + 1) + ".gif");
                pic[52] = Toolkit.getDefaultToolkit().
                    getImage("images\\s1.gif"); // 小王
                pic[53] = Toolkit.getDefaultToolkit().
                    getImage("images\\s2.gif"); // 大王
            }
        }
    }
    //构造方法 Poker() 用于初始化这副扑克
    public Poker() {
        getPic();
        for (int i = 0; i < colors.length; i++) {
            for (int j = 0; j < values.length; j++) { //生成每张牌
                cards[i * 13 + j] = new Card(colors[i], values[j], pic[i*13+j]);
            }
        }
        cards[52] = new Card("小王", pic[52]);
    }
}

```



```

        cards[53] = new Card("大王",pic[53]);
    }

    //方法 getCard()用于获取所有牌
    public Card[] getCard() {
        return Poker.cards;
    }

    //方法 getCard(int n )用于获取一张牌
    public Card getCard(int n) {
        return Poker.cards[n-1];
    }

    //方法 Show()用于显示一副新的扑克
    public void Show() {
        for (int i = 0; i < 54; i++) {
            cards[i].print();
        }
        System.out.println();
    }

    public void xipai() { //洗牌
        int i,j;
        Card tc;
        for (int k = 1; k <= 500; k++)
        {
            i = (int) (Math.random() * 54);
            j = (int) (Math.random() * 54);
            tc = cards[i];
            cards[i] = cards[j];
            cards[j] = tc;
        }
    }
}

```

7.3.3 设计游戏面板类 (PokerPanel.java)

导入包及相关的类:

```

import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.net.URL;
import java.util.ArrayList;
import javax.swing.*;

public class PokerPanel extends JPanel {
    Poker p = new Poker();
    int n = 1;
    ArrayList Mycards = new ArrayList(); //电脑得到的牌
    ArrayList Computercards = new ArrayList(); //玩家得到的牌
    int Myscore = 0; //电脑得分
    int Computerscore = 0; //玩家得分
    public boolean ComputerContinue = true;
    public boolean Computerlose = false; //电脑输了的标志
    public boolean Mylose = false; //玩家输了的标志
}

```


构造方法 `PokerPanel()` 用于游戏开始时将一副牌 (p) 洗好。

```
public PokerPanel() {
    p.xipai();
    p.Show();
    this.setVisible(true);
    repaint();
}
```

`faCard()` 方法给玩家发牌。玩家得到一张牌后, 如果玩家超过 21 点则提示“你输了, 超过 21 点”。

```
public void faCard() {
    Mycards.add(p.getCard(n)); // 玩家得到一张牌
    calcMyScore();
    n++;
    repaint();
    if (Myscore > 21) {
        Mylose = true;
        /* 显示提示信息对话框 */
        JOptionPane.showMessageDialog(null, "你输了, 超过 21 点", "提示",
            JOptionPane.ERROR_MESSAGE);
    }
}
```

`faCardToComputer()` 方法的作用是给电脑发牌。如果电脑超过 21 点则提示“超过 21 点, 电脑输了”。如果电脑得分小于 17 点则电脑要到一张牌, 否则电脑不再要牌。

```
public void faCardToComputer() {
    calcComputerScore();
    if (Computerscore > 21) {
        JOptionPane.showMessageDialog(null, "超过 21 点, 电脑输了", "提示",
            JOptionPane.ERROR_MESSAGE);
        ComputerContinue = false;
        Computerlose = true; // 电脑输了
    }
    if (Computerscore < 17) {
        Computercards.add(p.getCard(n)); // 电脑得到一张牌
        n++;
        repaint();
    } else {
        /* 显示提示信息对话框 */
        JOptionPane.showMessageDialog(null, "电脑不再要牌", "提示",
            JOptionPane.ERROR_MESSAGE);
        ComputerContinue = false;
    }
}
```

`calcComputerScore()` 方法的作用是计算电脑得分 (点数和)。

```
public void calcComputerScore() {
    public void calcComputerScore() {
        Computerscore = 0;
        for (int i = 0; i < Computercards.size(); i++) {
            Card c = (Card) Computercards.get(i);
            Computerscore += c.count;
        }
    }
}
```


calcComputerScore()方法用于计算玩家得分（点数和）。

```
public void calcMyScore() {
    Myscore = 0;
    for (int i = 0; i < Mycards.size(); i++) {
        Card c = (Card) Mycards.get(i);
        Myscore += c.count;
    }
}
```

shuying()方法用于计算两家的得分，判断输赢。

```
public void shuying() {
    calcComputerScore();
    calcMyScore();
    if (Computerlose == false)
        JOptionPane.showMessageDialog(null, "电脑输了", "提示",
            JOptionPane.ERROR_MESSAGE);
    if (Mylose == false)
        JOptionPane.showMessageDialog(null, "玩家输了", "提示",
            JOptionPane.ERROR_MESSAGE);
    if (Myscore > Computerscore)
        JOptionPane.showMessageDialog(null, "玩家赢了", "提示",
            JOptionPane.ERROR_MESSAGE);
    else
        JOptionPane.showMessageDialog(null, "电脑赢了", "提示",
            JOptionPane.ERROR_MESSAGE);
}
```

paint(Graphics g)事件用来绘制两家的牌面。

```
/**
 * 游戏绘图
 */
public void paint(Graphics g) {
    g.clearRect(0, 0, this.getWidth(), this.getHeight());
    g.drawString("玩家牌", 400, 250);
    // 玩家牌在下方
    for (int i = 0; i < Mycards.size(); i++) {
        Card c = (Card) Mycards.get(i);
        c.setPosition(50 * i, 200);
        c.print();
        c.paint(g, this);
    }
    System.out.println();
    g.drawString("电脑牌", 400, 100);
    // 电脑牌在上方
    for (int i = 0; i < Computercards.size(); i++) {
        Card c = (Card) Computercards.get(i);
        c.setPosition(50 * i, 50);
        c.print();
        c.paint(g, this);
    }
    System.out.println();
}
```

newGame()方法用于重新开始游戏。两家的牌就保存在 ArrayList 列表中，因此首先清空

ArrayList 列表，同时清空积分和输赢标志。

```
public void newGame() {
    // TODO Auto-generated method stub
    Mycards.clear();
    Computercards.clear();
    Myscore = 0;
    Computerscore = 0;
    ComputerContinue = true;
    Computerlose = false; //电脑输了的标志
    Mylose = false; //玩家输了的标志
    //给两家各发两张牌
    for(int i=1;i<=2;i++){
        faCard();
        faCardToComputer();
        repaint();
    }
}
}
```

7.3.4 设计游戏主窗口类 (Pai.java)

游戏界面中，需要添加 3 个命令按钮，button1 为“玩家要牌”、button2 为“玩家停牌”、button3 为“重新开始”。

导入包及相关的类：

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.*;

public class Pai extends JFrame{
    PokerPanel panel2=new PokerPanel();
    JButton button1=new JButton("玩家要牌");
    JButton button2=new JButton("玩家停牌");
    JButton button3=new JButton("重新开始");

    public Pai() {
        JPanel panel=new JPanel(new BorderLayout());
        JPanel panel3=new JPanel(new BorderLayout());

        String urlString="C://rear.gif";//背面牌 (rear.gif)
        JLabel label=new JLabel(new ImageIcon(urlString));

        panel.add(label, BorderLayout.CENTER);
        panel2.setLayout(new BorderLayout());
        panel3.setLayout(new FlowLayout());
        panel3.add(button1);
        panel3.add(button2);
        panel3.add(button3);
        this.getContentPane().setLayout(new BorderLayout());
        this.getContentPane().add(panel, BorderLayout.NORTH);
        this.getContentPane().add(panel2, BorderLayout.CENTER);
        this.getContentPane().add(panel3, BorderLayout.SOUTH);
        this.setSize(500, 500);
    }
}
```



```

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("21 点扑克牌游戏");
        this.setVisible(true);
        button1.setEnabled(false);
        button2.setEnabled(false);
        /**
         * 鼠标事件监听
         */
        button1.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                panel2.faCard();
                if (panel2.ComputerContinue)
                    panel2.faCardToComputer();
            }
        });
        button2.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                //电脑不超过 17 点则继续要牌
                while (panel2.ComputerContinue)
                    panel2.faCardToComputer();
                panel2.shuying();
            }
        });
        button3.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                panel2.newGame();
                button1.setEnabled(true);
                button2.setEnabled(true);
            }
        });
    }
    public static void main(String[] args) {
        //TODO Auto-generated method stub
        Pai showImage=new Pai();
    }
}

```

上述编程中，我们用 Card 类来描述牌，对 Card 的 value 取值为“A”，“2”，“3”，“4”，“5”，“6”，“7”，“8”，“9”，“10”，“J”，“Q”，“K”；对 color 取值为“黑桃”，“红桃”，“梅花”，“方块”。游戏规则也作了简化，只有两个玩家，也未对玩家属性（例如财富、下注、所持牌、持牌点数等）进行描述。实践表明，用 Card 和 Poker 类可以较好地描述游戏逻辑。

第 8 章

连连看游戏

8.1 连连看游戏介绍

“连连看”是一款源自台湾的桌面小游戏，自从流入大陆以来风靡一时，因为它是不分男女老少，适合大众的，集休闲、趣味、益智和娱乐于一体的经典小游戏。

“连连看”游戏考验的是玩家的眼力，在有限的时间内，只要把所有能连接的相同图案，两个一对地找出来，每找出一对，它们就会自动消失，只要把所有的图案全部消完即可获得胜利。所谓能够连接，是指无论横向或者纵向，从一个图案到另一个图案之间的连线不能超过两个弯（中间的直线不超过 3 根），其中，连线不能从尚未消去的图案上经过。

本章开发连连看游戏，游戏运行界面如图 8-1 所示。游戏具有统计消去的方块个数功能，这里由于是 10 行 10 列，因此方块总数为 100 个。玩家时间是 200s，如果玩家无法通关，则可以重新开始新的一局游戏。



图 8-1 连连看游戏运行界面

玩家第一次使用鼠标单击游戏界面中的动物方块,该方块此时为被选中状态,以特殊方式(红色方块)显示;再次单击其他方块,如果第二个方块与被选中的方块图案相同,且把第一个方块到第二个方块连起来,中间的直线不超过3根,则消掉这一对方块;否则第一个方块恢复成未被选中状态,而第二个方块变成被选中状态。

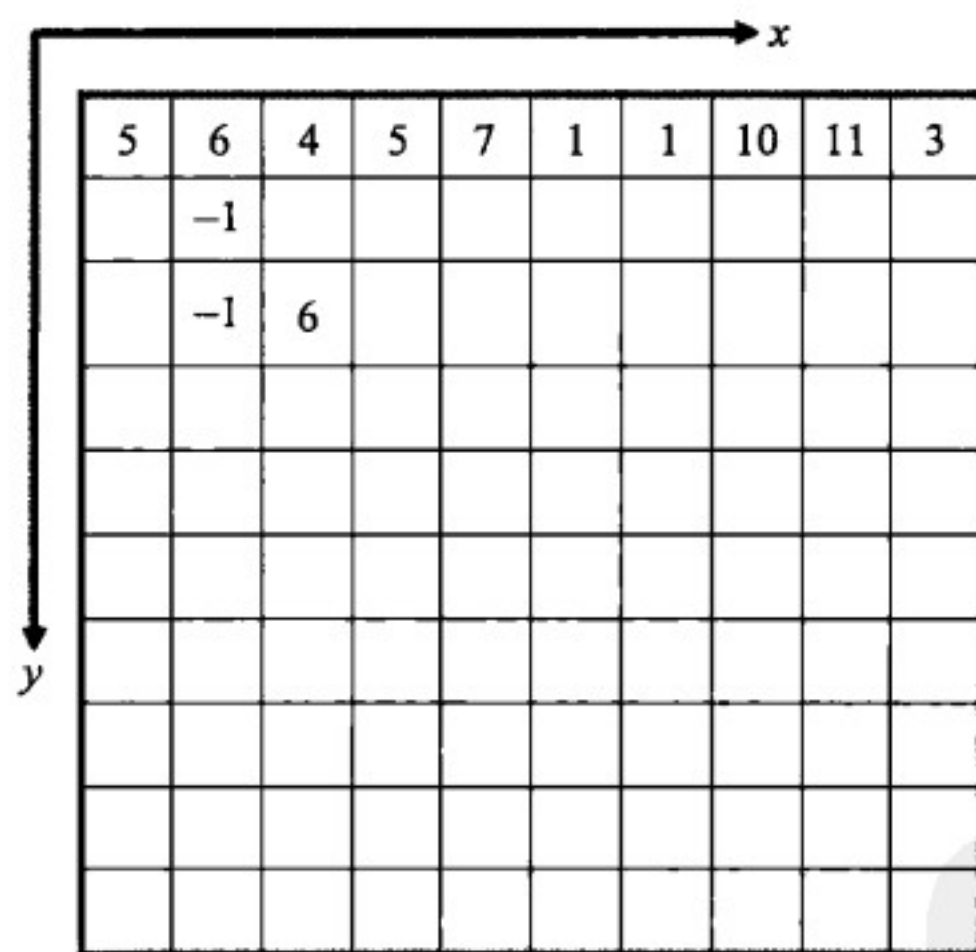
本游戏增加了智能查找功能,当玩家自己无法找到时,可以用鼠标右键单击画面,则会出现可以消去的两个方块(被加上蓝色边框线)的提示。

8.2 程序设计的思路

8.2.1 连连看游戏数据模型

对于游戏玩家而言,游戏界面上看到的“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,只不过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。

连连看的游戏界面是一个 $N \times M$ 的“网格”地图,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组来定义即可,而每个网格上所显示的图片,对于底层的数据模型来说,不同的图片对应不同的数值即可。图 8-2 所示为连连看游戏数据模型的示意图。



5	6	4	5	7	1	1	10	11	3
-1									
-1	6								

图 8-2 连连看游戏的数据模型

对于图 8-2 所示的数据模型,只要让数值为-1 (BLANK_STATE) 的网格上不绘制动物图片,其他数值是动物方块的图像的 ID;非-1 (BLANK_STATE) 的网格则绘制相应的动物图片,就可显示出连连看的游戏界面。本程序实际上并不是直接使用整型二维数组来保存游戏的状态数据,而是采用一维数组 `m_map`。对地图中的行列数的表达,用一个转换法则即可。

例如,点 $(x1,y2)$ 对应的数组元素为 `m_map(y2 × m_nCol + x1)`,其中 `m_nCol` 是总列数。当然数组元素下标也可换算出在“网格”地图中的坐标点。

8.2.2 动物方块布局

连连看游戏数据模型设计完成之后,在游戏开始前,如何对它进行初始化呢?下面我们一起来进行分析。由于方块需要成对地出现,因此在做地图初始化的时候,不应该仅对动物方块的图像 ID 做简单的随机取数,然后将该随机选出来的物件放到地图区域中就算完成;而是需要成对地对动物方块的图像进行选取,就是说地图区域中的小方块必须是偶数个才行。但是,怎样使方块图案成对出现呢?这里需要引入一个临时地图 tmpMap,该临时地图的大小与实际地图 m_map 的大小一致。首先添置好 4 组完全一样的图像类型的 ID 数据 ($0 \sim m_nCol \times m_nRow / 4$),也就是说每种图像方块有 4 个。

下面,我们可以先按顺序把每种动物方块(实际上就是标号 ID)排好放入 ArrayList 列表 tmpMap(临时地图)中,然后再随机从 tmpMap(临时地图)中取一个动物方块放入地图 m_map 中。实际上程序内部无需识别动物方块的图像,只需要用一个 ID 来表示,运行界面上画出来的动物图像是根据地图中的 ID 取资源里的图片画的。如果 ID 的值为-1 (BLANK_STATE),则说明此处已经被消除了。

```
private void StartNewGame()
{
    //初始化地图,将地图中所有方块区域的位置置为空方块状态
    for(int iNum=0;iNum<(m_nCol*m_nRow);iNum++)
    {
        m_map[iNum] = BLANK_STATE;
    }
    Random r = new Random();
    //生成随机地图
    //将所有匹配成对的动物物种放进一个临时地图中
    ArrayList tmpMap=new ArrayList ();
    for(int i=0;i<(m_nCol*m_nRow)/4;i++)
        for(int j=0;j<4;j++)
            tmpMap.add(i);

    //每次从上面的临时地图中取走(获取后并在临时地图中删除)
    //将一个动物放到地图的空方块上
    for (int i = 0; i < m_nRow * m_nCol; i++)
    {
        //随机挑选一个位置
        int nIndex = r.nextInt(tmpMap.size()) ;
        //获取该选定物件并放到地图的空方块中
        m_map[i]=(Integer)tmpMap.get(nIndex);
        //在临时地图中消除该动物
        tmpMap.remove(nIndex);
    }
}
```

在完成图案方块的摆放以及初始化后,下面将对整个游戏实现的关键算法,即图案方块的连通算法进行分析。

8.2.3 连通算法

我们分析一下连接的情况就可以看出,可以分 3 种情况,如图 8-3 所示。

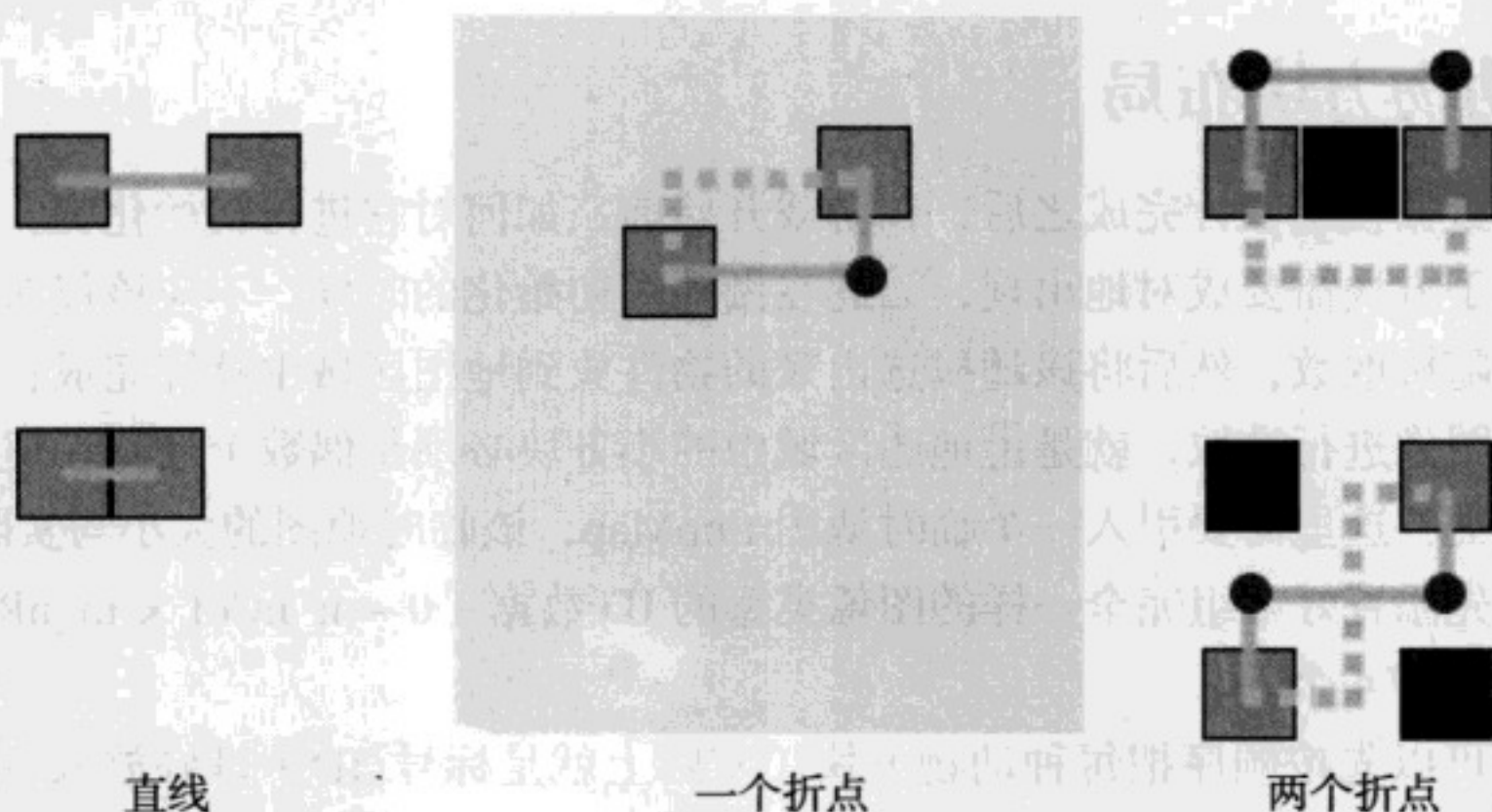


图 8-3 两个选中的方块之间连接线示意图

(1) 直连方式

在直连方式中, 要求两个选中的方块 x 或 y 相同, 即在一条直线上, 并且之间没有其他任何图案的方块。直连方式在 3 种连接方式中最简单。

(2) 一个折点

相当于两个方块之间划出一个矩形, 这两个方块是一对对角顶点, 另外两个顶点中某个顶点 (即折点) 如果可以同时和这两个方块直连, 那就说明可以“一折连通”。

(3) 两个折点

这种方式的两个折点 ($z1, z2$) 必定在两个目标点 (两个选中的方块) $p1$ 和 $p2$ 所在的 x 方向或 y 方向的直线上。

按 $p1(x1, y1)$ 点向 4 个方向探测, 例如向右探测, 每次 $x1+1$, 判断 $z1(x1+1, y1)$ 与 $p2(x2, y2)$ 点可否形成一个折点连通性, 如果可以形成连通, 则两个折点连通, 否则直到超过图形右边界区域。假如超过图形右边界区域, 则还需判断两个折点是否在选中方块的右侧, 且两个折点在图案区域之外的连通情况是否存在。此时判断可以简化为判断 $p2$ 点 ($x2, y2$) 是否可以水平直通到边界。

经过上面的分析, 对两个方块是否可以抵消的算法流程图如图 8-4 所示。根据图 8-4 所示的流程图, 对选中的两个方块 (分别在 $(x1, y1)$ 、 $(x2, y2)$ 位置) 是否可以抵消的判断按如下方式实现。把该功能封装在 `IsLink()` 方法里面, 其代码如下:

```
//
// 判断选中的两个方块是否可以消除
//
boolean IsLink(int x1, int y1, int x2, int y2)
{
    //x 直连方式即垂直方向连通
    if(x1==x2)
    {
        if(X_Link(x1, y1, y2))
        { LType=LinkType.LineType ; return true;}
    }
    //y 直连方式即水平方向连通
    else if(y1==y2)
    {
        if(Y_Link(x1, x2, y1))
        { LType = LinkType.LineType; return true; }
    }
}
```



```

}
//一个转弯（折点）的联通方式
if(OneCornerLink(x1,y1,x2,y2))
{
    LType = LinkType.OneCornerType ;
    return true;
}
//两个转弯（折点）的联通方式
else if(TwoCornerLink(x1,y1,x2,y2))
{
    LType = LinkType.TwoCornerType;
    return true;
}
return false;
}

```

直连方式分为 x 或 y 连通情况，分别使用 `X_Link ()` 方法实现判断 x 直接连通即垂直方向连通和 `Y_Link ()` 方法实现判断 y 直接连通即水平方向连通。

```

//
//x 直接连通即垂直方向连通
//
boolean X_Link(int x, int y1,int y2)
{
    //保证 y1 的值小于 y2
    if(y1>y2)
    {
        //数据交换
        int n=y1;
        y1=y2;
        y2=n;
    }

    //直通
    for(int i=y1+1;i<=y2;i++)
    {
        if(i==y2)
            return true;
        if(m_map[i*m_nCol+x]!=BLANK_STATE)
            break;
    }
    return false;
}
//
//y 直接连通即水平方向连通
//
boolean Y_Link(int x1,int x2,int y)
{
    if(x1>x2)
    {
        int x=x1;
        x1=x2;
        x2=x;
    }
}

```

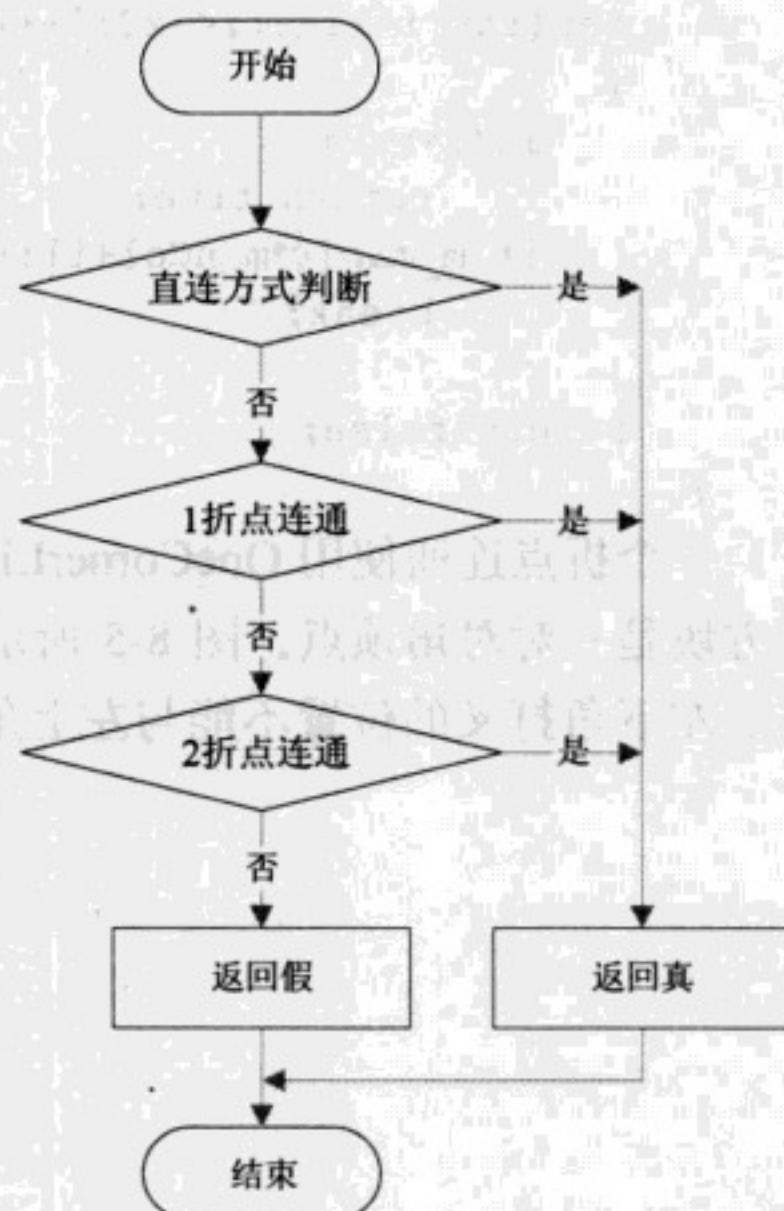


图 8-4 流程图


```

//直通
for(int i=x1+1;i<=x2;i++)
{
    if(i==x2)
        return true;
    if(m_map[y*m_nCol+i]!=BLANK_STATE)
        break;
}
return false;
}

```

一个折点连通使用 OneCornerLink()方法实现判断。其实相当于两个方块划出一个矩形，这两个方块是一对对角顶点，图 8-5 所示为两个黑色目标方块的连通情况，右上角打叉的位置就是折点。左下角打叉的位置不能与左上角黑色目标方块连通，因此不能作为折点。

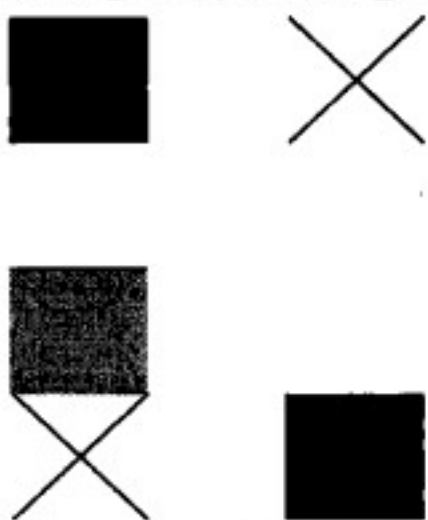


图 8-5 一个折点连通示意图

```

//
//一个折点连通
//
boolean OneCornerLink(int x1, int y1, int x2, int y2)
{
    if (x1 > x2) //目标点(x1,y1), (x2,y2)两点交换
    {
        int n=x1;
        x1=x2;
        x2=n;
        n=y1;
        y1=y2;
        y2=n;
    }
    if (y2 < y1) //(x1,y1)为矩形左下顶点, (x2,y2)为矩形右上顶点
    {
        //判断矩形右下角折点(x2,y1)是否为空
        if (m_map[y1 * m_nCol + x2] == BLANK_STATE)
        {
            if (Y_Link(x1, x2, y1) && X_Link(x2, y1, y2))
                //判断折点(x2,y1)与两个目标点是否直通
                {
                    z1.x= x2; z1.y = y1; //保存折点坐标到 z1
                    return true;
                }
        }
        //判断矩形左上角折点(x1,y2)是否为空
        if (m_map[y2 * m_nCol + x1] == BLANK_STATE)
        {
            if (Y_Link(x2, x1, y2) && X_Link(x1, y2, y1))

```



```

        //判断折点 (x1,y2) 与两个目标点是否直通
    {
        z1.x = x1; z1.y = y2; //保存折点坐标到 z1
        return true;
    }
    }
    return false;
}
else // (x1,y1) 为矩形左上顶点, (x2,y2) 为矩形右下顶点
{
    //判断矩形左下角折点 (x1,y2) 是否为空
    if (m_map[y2 * m_nCol + x1] == BLANK_STATE)
    {
        if (Y_Link(x1, x2, y2) && X_Link(x1, y1, y2))
            //判断折点 (x1,y2) 与两个目标点是否直通
        {
            z1.x = x1; z1.y = y2; //保存折点坐标到 z1
            return true;
        }
    }
    //判断矩形右上角折点 (x2,y1) 是否为空
    if (m_map[y1 * m_nCol + x2] == BLANK_STATE)
    {
        if (Y_Link(x1, x2, y1) && X_Link(x2, y1, y2))
            //判断折点 (x2,y1) 与两个目标点是否直通
        {
            z1.x = x2; z1.y = y1; //保存折点坐标到 z1
            return true;
        }
    }
    return false;
}
}

```

两个折点是否连通使用 TwoCornerLink () 方法实现判断。按 p1 (x1,y1) 点向 4 个方向探测新 z2 点与 p2 (x2,y2) 点可否形成一个折点连通性。

```

///
///两个折点连通
///
boolean TwoCornerLink(int x1, int y1, int x2, int y2)
{
    if (x1 > x2)
    {
        int n = x1;      x1 = x2;      x2 = n;
        n = y1;          y1 = y2;      y2 = n;
    }
    //右侧
    int x, y;
    for (x = x1 + 1; x <= m_nCol; x++)
    {
        if (x == m_nCol)
            //两个折点在选中方块的右侧, 且两个折点在图案区域之外
            if (XThrough(x2 + 1, y2, true))
            {

```



```

        z2.x = m_nCol; z2.y = y1;
        z1.x = m_nCol; z1.y = y2;
        return true;
    }
    else
        break ;
    if (m_map[y1 * m_nCol + x] != BLANK_STATE)
        break;
    if (OneCornerLink(x, y1, x2, y2))
    {
        z2.x = x; z2.y = y1;
        return true;
    }
}
//左侧
for (x = x1 - 1; x >=-1; x--)
{
    if (x == -1)
        //两个折点在选中方块的左侧，且两个折点在图案区域之外
        if (XThrough(x2 - 1, y2, false))
        {
            z2.x = -1; z2.y = y1;
            z1.x = -1; z1.y = y2;
            return true;
        }
    else
        break;

    if (m_map[y1 * m_nCol + x] != BLANK_STATE)
        break;
    if (OneCornerLink(x, y1, x2, y2))
    {
        z2.x = x; z2.y = y1;
        return true;
    }
}
//上侧
for (y = y1 - 1; y >=-1; y--)
{
    if (y == -1)
        //两个折点在选中方块的上侧，且两个折点在图案区域之外
        if (YThrough(x2, y2 - 1, false))
        {
            z2.x = x1; z2.y = -1;
            z1.x = x2; z1.y = -1;
            return true;
        }
    else
        break;
    if (m_map[y * m_nCol + x1] != BLANK_STATE)
        break;
    if (OneCornerLink(x1, y, x2, y2))
    {
        z2.x = x1; z2.y = y;
        return true;
    }
}

```



```

    }
}
//下侧
for (y = y1 + 1; y <= m_nRow; y++)
{
    if (y == m_nRow)
        //两个折点在选中方块的下侧, 且两个折点在图案区域之外
        if (YThrough(x2, y2 + 1, true))
        {
            z2.x = x1; z2.y = m_nRow;
            z1.x = x2; z1.y = m_nRow;
            return true;
        }
    else
        break;
    if (m_map[y * m_nCol + x1] != BLANK_STATE)
        break;
    if (OneCornerLink(x1, y, x2, y2))
    {
        z2.x = x1; z2.y = y;
        return true;
    }
}
return false;
}

```

Xthrough()方法用于水平方向判断到边界的连通性, 如果 bAdd 为 True, 则从 (x,y) 点水平向右直到边界, 判断是否全部为空块; 如果 bAdd 为 False, 则从 (x,y) 点水平向左直到边界, 判断是否全部为空块。

boolean XThrough(int x, int y, boolean bAdd) //水平方向判断到边界的连通性

```

{
    if (bAdd) //True, 水平向右判断是否连通 (是否为空块)
    {
        for (int i = x; i < m_nCol; i++)
            if (m_map[y * m_nCol + i] != BLANK_STATE)
                return false;
    }
    else //False, 水平向左判断是否连通 (是否为空块)
    {
        for (int i = 0; i <= x; i++)
            if (m_map[y * m_nCol + i] != BLANK_STATE)
                return false;
    }
    return true;
}

```

Ythrough()方法用于垂直方向判断到边界的连通性, 如果 bAdd 为 True, 则从 (x, y) 点垂直向下直到边界, 判断是否全部为空块; 如果 bAdd 为 False, 则从 (x,y) 点垂直向上直到边界, 判断是否全部为空块。

boolean YThrough(int x, int y, boolean bAdd) //垂直方向判断到边界的连通性

```

{
    if (bAdd) //True, 垂直方向向下判断是否连通 (是否为空块)
    {
        for (int i = y; i < m_nRow; i++)

```



```

        if(m_map[i*m_nCol+x]!=BLANK_STATE)
            return false;
    }
    else    //False, 垂直方向向上判断是否连通 (是否为空块)
    {
        for(int i=0;i<=y;i++)
            if(m_map[i*m_nCol+x]!=BLANK_STATE)
                return false;
    }
    return true;
}

```

8.2.4 智能查找功能的实现

在地图上自动查找出一组图案相同可以抵消的方块，可以采用遍历算法。下面通过图 8-6 协助我们分析此算法。

在图中找相同物种的方块时，将按方块地图数组 `m_map` 的下标位置对每个方块进行查找，一旦找到一组图案相同可以抵消的方块则马上返回。查找相同方块组的时候，必须先确定第一个选定方块（例如 0 号方块），然后在这个基础上做遍历查找第二个选定方块，即从 1 开始按照 1, 2, 3, 4, 5, 6, 7……顺序查找第二个选定方块，并判断选定的两个方块是否连通抵消。假如 0 号方块与 5 号方块连通，我们则经历(0,1)、(0,2)、(0,3)、(0,4)、(0,5)5 组数据的判断对比，成功后则立即返回。

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

图 8-6 方块匹配示意图 (1)

如果找不到匹配的第二个选定方块，则如图 8-7 (a) 所示，编号加 1 重新选定第一个选定方块（即 1 号方块）后进入下一轮，然后在这个基础上做遍历查找第二个选定方块，即如图 8-7 (b)

所示，从 2 号开始按照 2, 3, 4, 5, 6, 7……顺序查找第二个选定方块，直到搜索到最后一块方块（即 15 号方块）。那么为什么从 2 开始查找第二个选定方块，而不是从 0 号开始呢？因为将 1 号方块选定为第一个选定方块前，0 号已经作为第一个选定方块对后面的方块进行可连通的判断了，它必然不会与后面的方块连通。

如果找不到与 1 号方块连通且相同的方块，则编号加 1 重新选定第一个选定方块（即 2 号方块）进入下一轮，从 3 号开始按照 3, 4, 5, 6, 7……顺序查找第二个选定方块。

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

(a) 0 号方块找不到匹配方块，则选定 1 号方块

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

(b) 从 2 号开始找匹配方块

图 8-7 方块匹配示意图 (2)

按照上面设计的算法，智能查找匹配方块的流程图如图 8-8 所示。

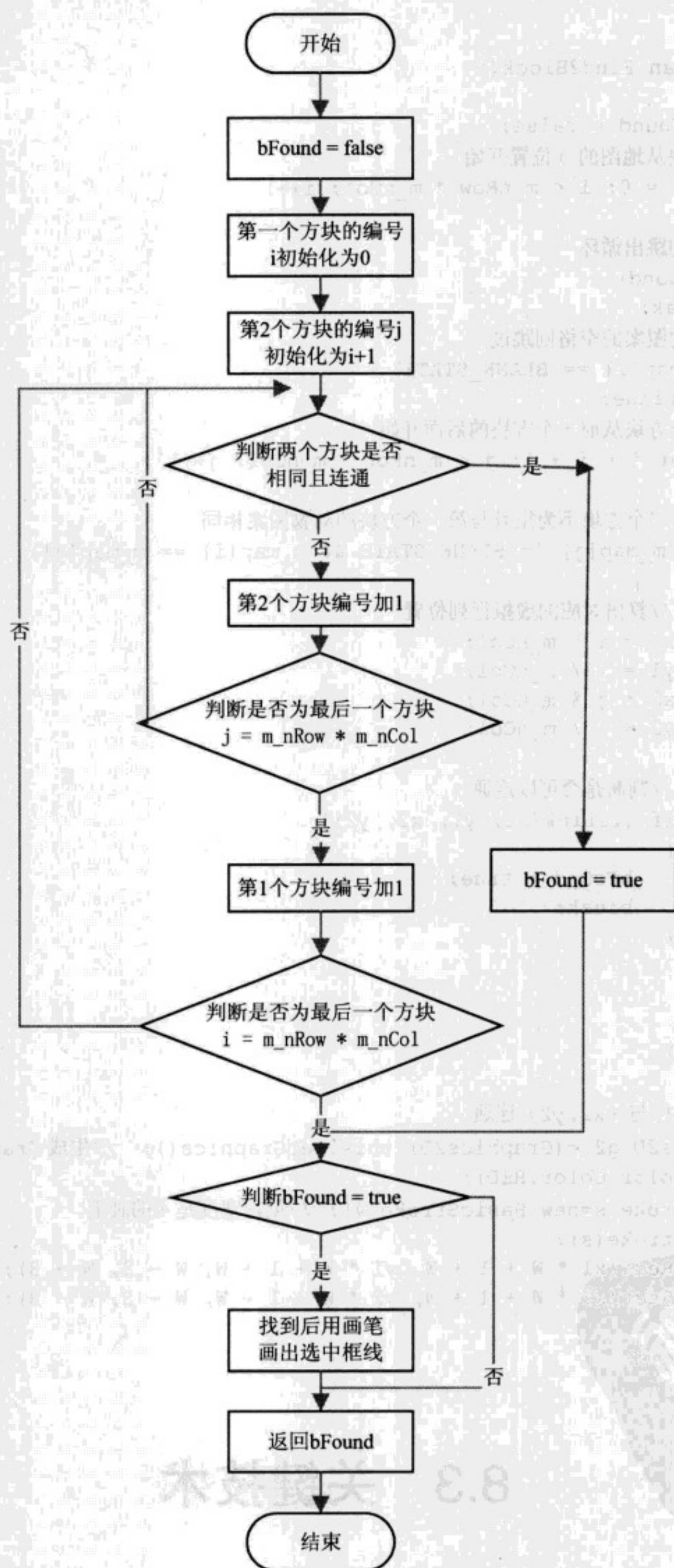


图 8-8 智能查找匹配方块的流程图

根据流程图，将自动查找出一组图案相同可以抵消的方块功能封装在 Find2Block()方法中，

其代码如下：

```
private boolean Find2Block()
{
    boolean bFound = false;
    //第一个方块从地图的 0 位置开始
    for (int i = 0; i < m_nRow * m_nCol; i++)
    {
        //找到则跳出循环
        if (bFound)
            break;
        //无动物图案的空格则跳过
        if (m_map[i] == BLANK_STATE)
            continue;
        //第二个方块从前一个方块的后面开始
        for (int j = i + 1; j < m_nRow * m_nCol; j++)
        {
            //第二个方块不为空且与第一个方块的动物图案相同
            if (m_map[j] != BLANK_STATE && m_map[i] == m_map[j])
            {
                //算出对应的虚拟行列位置
                x1 = i % m_nCol;
                y1 = i / m_nCol;
                x2 = j % m_nCol;
                y2 = j / m_nCol;

                //判断是否可以连通
                if (IsLink(x1, y1, x2, y2))
                {
                    bFound = true;
                    break;
                }
            }
        }
    }
    if (bFound)
    {
        //(x1,y1)与(x2,y2)连通
        Graphics2D g2 =(Graphics2D) this.getGraphics(); //生成 Graphics 对象
        g2.setColor(Color.RED);
        BasicStroke s=new BasicStroke(4); //创建宽度是 4 的画笔
        g2.setStroke(s);
        g2.drawRect(x1 * W + 1 + W, y1 * W + 1 + W, W - 3, W - 3);
        g2.drawRect(x2 * W + 1 + W, y2 * W + 1 + W, W - 3, W - 3);
    }
    return bFound;
}
```

8.3 关键技术

8.3.1 动物方块图案的显示

程序内部不需要认识动物方块的图像，只需要用一个 ID 来表示，运行界面上画出来的动物

图形是根据地图中 ID 的值取资源里的图片画的。Graphics 类的 drawImage()方法用于在指定位置显示原始图像或者缩放后的图像。该方法的常用形式为：

```
public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)
```

其中，

img: 要绘制的指定图像。如果 img 为 null, 则此方法不执行任何动作。

x: x 坐标。

y: y 坐标。(x, y) 指定所绘制图像的位置。

width: 矩形的宽度。

height: 矩形的高度。矩形指定所绘制图像的大小。将图像进行缩放以适合该矩形。

observer: 当转换了更多图像时要通知的对象。

例如, 将汽车图片显示在 (100, 200) 位置以 80×80 像素大小显示。

```
int W=80;
Image myImage = Toolkit.getDefaultToolkit().getImage("car.jpg");
g.drawImage(myImage, 100, 200, W, W,this);
```

程序中需要从一幅图(所有动物图案)中截取一部分图案(某个动物图案), 应用 BufferedImage 类的 getSubimage 方法可以实现。

```
BufferedImage getSubimage(int x, int y, int w, int h)
```

返回由指定矩形区域定义的子图像。

以下代码实现在原图 animal2.bmp 文件中, 从坐标 (0, 39×3) 处截取长宽为 39 的子图像, 并将截取到的子图像生成新的图形 (Image) 对象 (newpic)。

```
int x=0;
int y=3*39 ;
int w=39;
int h=39;
BufferedImage src=null;
BufferedImage newpic=null;
try{
    src = ImageIO.read(new File("pic\\animal2.bmp"));
}
catch(Exception e)
{
    System.out.println(e);
}
//生成新的图形 (Image) 对象 (newpic)
Image newpic=src.getSubimage(x, y, w, h); //截取原图中矩形区域的图形
```

游戏开发中需要大量使用 Graphics 类的 drawImage()方法, 该方法需要读者重点掌握。

8.3.2 鼠标相关的事件

用户在窗体游戏面板上单击时, 由屏幕像素坐标(e.X,e.Y)计算被单击方块的地图位置坐标 (x,y)。同时还需要判断用户是用左键还是右键单击, 如果右键则调用智能查找功能。和鼠标相关的事件的典型问题, 包括读取鼠标的当前位置; 判断究竟是哪个鼠标按键被按动。下面介绍鼠标相关的事件。

Java 语言中主要提供了 3 种不同类型的鼠标事件: 鼠标键事件、鼠标移动事件和鼠标轮滚动事件。鼠标键事件多用于鼠标的单击处理; 鼠标移动事件用于鼠标移动的处理; 鼠标轮滚动事件是从 JDK 1.4 后引入的鼠标事件, 用于鼠标轮的动作处理。这 3 种类型的鼠标事件一般是以容器

组件作为事件源，它们各有自己的监听器。

(1) 鼠标键事件处理

鼠标键事件处理是一种最常见的鼠标事件处理方式。鼠标键事件处理涉及监听器接口 `MouseListener` 和鼠标事件 `MouseEvent`。对于这种事件处理的具体步骤如下。

- 组件通过方法 `addMouseListener()` 注册到 `MouseListener` 中。允许监听器对象在程序运行过程中监听组件是否有鼠标键事件 `MouseEvent` 对象发生。
- 实现 `MouseListener` 接口的所有方法，提供事件发生的具体处理办法。

鼠标键事件对应的是事件类 `MouseEvent`，该类的主要方法如表 8-1 所示。鼠标键事件对应的监听器接口是 `MouseListener`，这种监听器接口的主要方法如表 8-2 所示。

表 8-1

MouseEvent 类的主要方法

方 法	功 能
int getButton()	获取鼠标按键变更的状态
int getClickCount()	获取鼠标单击的次数
Point getPoint()	获取鼠标单击的位置
int getX()	获取鼠标的 x 坐标位置
int getY()	获取鼠标的 y 坐标位置
String getMouseModifiersText(int)	获取控制键与鼠标的组合键的字符串

表 8-2

MouseListener 接口的方法

方 法	功 能
void mousePressed(MouseEvent)	鼠标按下调用
void mouseReleased(MouseEvent)	鼠标释放调用
void mouseEntered(MouseEvent)	鼠标进入调用
void mouseExited(MouseEvent)	鼠标离开调用
void mouseClicked(MouseEvent)	鼠标单击调用

例如设计一个程序，可以获取并显示鼠标位置和鼠标状态。当鼠标按下时，能在当前位置绘制一个正方形。程序运行效果如图 8-9 所示。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MouseStatus extends JFrame implements MouseListener{
    Container container;
    JTextArea displayArea;           //定义显示详细信息的文本区
    JPanel panel;                    //定义绘制区面板
    JLabel statusLabel;              //定义状态标签

    public MouseStatus(){
        super("鼠标事件示例");
        container=getContentPane();  container.setLayout(new BorderLayout());
        displayArea=new JTextArea(10,20);
        panel=new JPanel();
        panel.setSize(100,100);
        panel.setBackground(Color.yellow);
    }
}
```



```

        panel.addMouseListener(this);           //注册鼠标监听器
        statusLabel=new JLabel("鼠标初始化状态");
        container.add(new JScrollPane(displayArea),BorderLayout.WEST);
        container.add(panel,BorderLayout.CENTER);
        container.add(statusLabel,BorderLayout.SOUTH);
        setSize(400,300);
        setVisible(true);
    }
    public void mouseEntered(MouseEvent e){    //鼠标进入
        statusLabel.setText("鼠标进入面板");
        displayArea.append("鼠标进入,X位置: "+e.getX()+"Y位置: "+e.getY()+"\n");
    }
    public void mouseExited(MouseEvent e){    //鼠标退出
        statusLabel.setText("鼠标离开面板");
        displayArea.append("鼠标离开"+"\\n");
    }
    public void mouseClicked(MouseEvent e){    //鼠标单击
        statusLabel.setText("鼠标在面板点击");
        displayArea.append("鼠标单击,X位置: "+e.getX()+"Y位置: "+e.getY()+"\\n");
    }
    public void mousePressed(MouseEvent e){    //鼠标按下
        statusLabel.setText("鼠标在面板按下");
        Graphics g=panel.getGraphics();        //获取图形上下文
        g.setColor(Color.blue);                //设置颜色
        g.drawRect(e.getX(),e.getY(),40,40);    //在当前位置绘制正方形
        panel.paintComponents(g);
        displayArea.append("鼠标按下,X位置: "+e.getX()+"Y位置: "+e.getY()+"\\n");
    }
    public void mouseReleased(MouseEvent e){    //鼠标释放
        statusLabel.setText("鼠标在面板释放");
        displayArea.append("鼠标释放,X位置: "+e.getX()+"Y位置: "+e.getY()+"\\n");
    }
    public static void main(String args[]){
        MouseStatus ms=new MouseStatus();
        ms.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

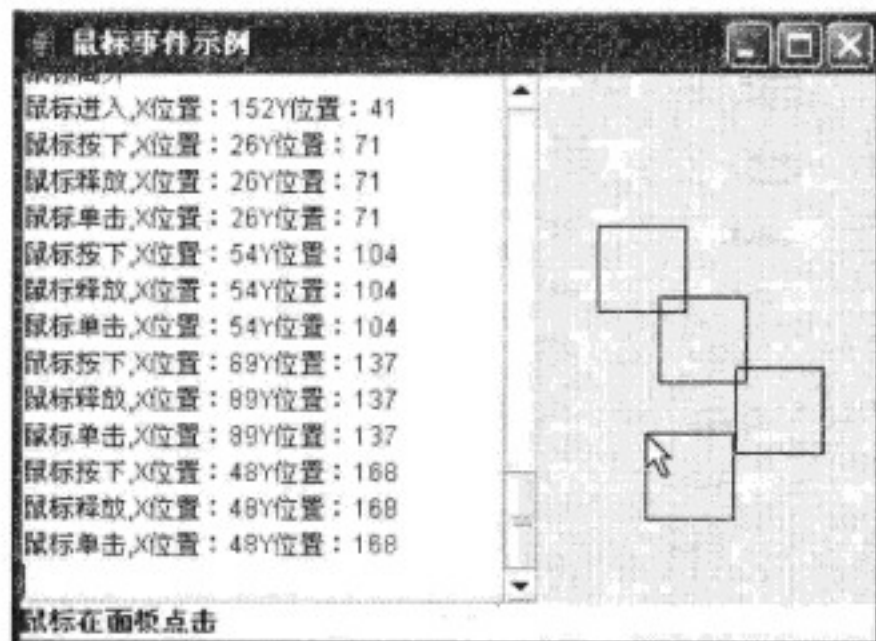


图 8-9 显示鼠标位置和鼠标状态的运行结果

(2) 鼠标移动事件处理

事件类 `MouseEvent` 还对应另一个监听器接口 `MouseMotionListener`。该接口可以实现鼠标两种运动的处理：即鼠标移动的处理和鼠标拖动的处理。实现鼠标移动事件处理的一般步骤如下。

- 组件通过方法 `addMouseMotionListener()`注册到 `MouseMotionListener` 中。允许 `MouseMotionListener` 监听器对象在程序运行过程中监听组件是否有鼠标键事件 `MouseEvent` 对象发生。
- 实现 `MouseMotionListener` 接口的所有方法，提供事件发生的具体处理办法。

`MouseMotionListener` 接口的方法如表 8-3 所示。

表 8-3 `MouseMotionListener` 接口的方法

方 法	功 能
<code>void mouseDragged(MouseEvent)</code>	鼠标拖动调用
<code>void mouseMoved(MouseEvent)</code>	鼠标移动调用

例如设计一个程序，可以实现文字“Hello, Java 世界”随鼠标的移动而移动，如果在鼠标拖动的时候，会将文字放大显示。程序运行效果如图 8-10 所示。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MouseMotionText extends JPanel{
    int x=20,y=20;           //设置初始坐标
    int mode=1;              //表示默认绘制模式，1:拖动，2:移动
    public MouseMotionText(){
        addMouseMotionListener(new MouseMotionListener(){//创建匿名内部类
            public void mouseDragged(MouseEvent e){        //鼠标拖动
                x=e.getX();                                //x 轴的坐标
                y=e.getY();                                //y 轴的坐标
                repaint();
            }
            public void mouseMoved(MouseEvent e){            //鼠标移动
                mode=2;                                     //设置为移动模式
                x=e.getX();
                y=e.getY();
                repaint();
            }
        });
    }
    public void paintComponent(Graphics g){
        g.clearRect(0,0,400,200);                          //清屏
        if(mode==1) g.setFont(new Font("宋体",Font.BOLD,g.getFont().getSize()+10));
        draw(g,x,y);
    }
    public void draw(Graphics g,int x,int y){
        g.drawString("Hello, Java 世界",x,y);
    }
    public Dimension getPreferredSize(){                    //获取最佳尺寸
        return new Dimension(400,200);
    }
    public static void main(String args[]){
```



```
JFrame frame=new JFrame();
frame.add(new MouseMotionText());
frame.setTitle("鼠标移动事件示例");
frame.setSize(400,200);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

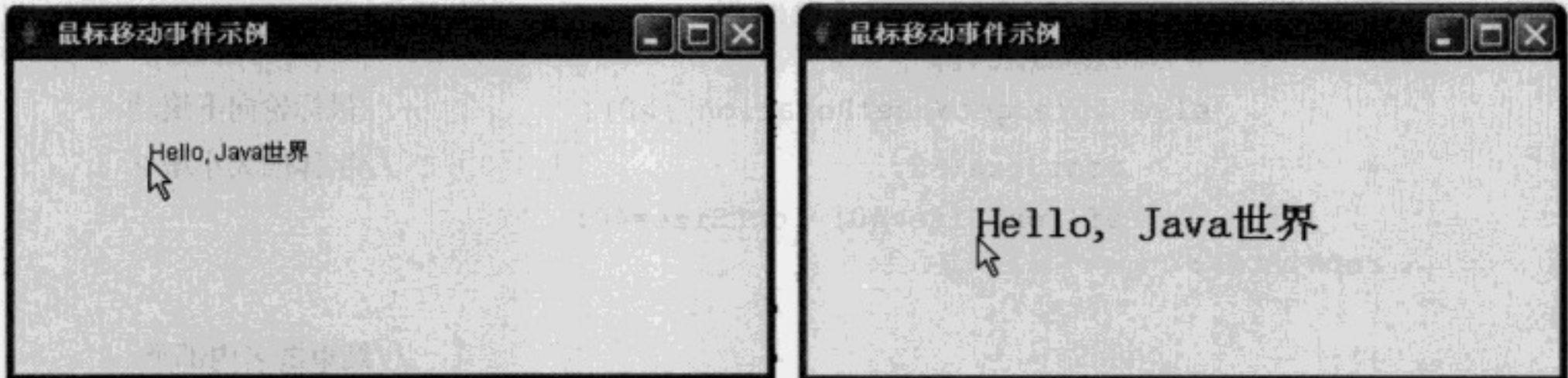


图 8-10 鼠标移动和拖动的运行效果

(3) 鼠标轮滚动事件处理

鼠标轮滚动事件可以处理鼠标中间的鼠标轮的动作。这种事件的实现依赖于事件类 `MouseEvent` 和接口 `MouseListener`。实现鼠标轮滚动事件的具体步骤如下。

- 组件通过方法 `addMouseListener()`注册到 `MouseListener` 中。允许监听器在程序运行过程中监听组件是否有鼠标轮事件 `MouseEvent` 对象的发生。
- 实现 `MouseListener` 接口的所有方法，提供事件发生的具体处理办法。

对于类 `MouseEvent` 是 `MouseEvent` 类的直接子类，具有 `MouseEvent` 类的特点。同时，它也具有自身的特征。`MouseEvent` 的常见方法如表 8-4 所示。

表 8-4 MouseEvent 类的常见方法	
方 法	功 能
int getScrollAmount()	获取滚动的单位数
int getScrollType()	获取滚动类型
int getWheelRotation()	获取鼠标轮旋转的运动量
int getUnitsToScroll()	实现 MouseWheelListener 类的便捷方法

与事件类 `MouseEvent` 对应的监听器接口是 `MouseListener`,该接口的方法如表 8-5 所示。

表 8-5 MouseWheelListener 接口的方法	
方 法	功 能
void mouseWheelMoved(MouseEvent)	鼠标轮移动调用

例如，设计一个程序，可以实现文字“欢迎来到 Java 世界”的显示，当向下滚动鼠标轮时，文字字体变大；当向上滚动鼠标轮时，文字字体变小。

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```



```

public class MouseWheelText extends JPanel{
    int fontSize=20;
    int x=50,y=120;
    public MouseWheelText(){
        addMouseListener(new MouseWheelListener(){
            public void mouseWheelMoved(MouseWheelEvent e){//创建匿名内部类
                if(e.getWheelRotation()<0){                //鼠标轮向上滚动
                    fontSize-=2;                            //字体的大小减 2
                    if(fontSize<5) fontSize=6;
                    repaint();
                }
                else if(e.getWheelRotation()>0){            //鼠标轮向下滚动
                    fontSize+=2;                            //字体的大小增 2
                    if(fontSize>40) fontSize=40;
                    repaint();
                }
            }
        });
    }
    public void paintComponent(Graphics g){
        g.clearRect(0,0,400,300);
        g.setFont(new Font("楷体",Font.BOLD,fontSize));
        g.drawString("欢迎来到 Java 世界",x,y);
    }
    public Dimension getPreferredSize(){
        return new Dimension(400,300);
    }
    public static void main(String args[]){
        JFrame frame=new JFrame("鼠标轮滚动事件示例");
        frame.add(new MouseWheelText());
        frame.setSize(400,300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

本游戏在鼠标的 MouseClicked 事件中判断用户是用左键单击还是右键单击，如果用右键单击则调用智能查找功能 Find2Block()。

8.3.3 延时功能

在 Java 中有时候需要使程序暂停一段时间，称为延时。普通延时用 Thread.sleep(int)方法实现。它将当前线程挂起指定的毫秒数。例如：

```

try
{
    Thread.currentThread().sleep(1000);//1000ms
}
catch(Exception e){}

```

在这里需要解释一下线程沉睡的时间。sleep()方法并不能够让程序“严格”地沉睡指定的时间。例如当使用 5000 作为 sleep()方法的参数时，线程可能在实际被挂起 5000.001ms 后才会继续运行。当然，对于一般的应用程序来说，sleep()方法对时间控制的精度足够了。

但是如果需要使用精确延时，最好使用 Timer 类：


```

Timer timer=new Timer();//实例化 Timer 类
timer.schedule(new TimerTask(){
public void run(){
    System.out.println("退出");
    this.cancel();}},500);//500ms

```

这种延时比 `sleep()` 方法精确。上述延时方法只运行一次，如果需要运行多次，应使用 `timer.schedule(new MyTask(), 1000, 2000);` 语句，则每间隔 2s 执行一次 `MyTask()` 方法。

8.4 程序设计的步骤

8.4.1 设计游戏界面窗体（LLKFrame.java）

在项目中创建一个继承 `JFrame` 的 `LLKFrame` 类，用于显示游戏面板 `LLKPanel` 和实现游戏逻辑。由于其他类需要访问并显示已消去方块数量的文本框 `textareal`，因此 `textareal` 定义为 `static`。

`LLKFrame` 窗体有两个面板组成，一个面板 `panell` 用来添加“重来一局”、“退出”按钮及显示已消去方块数量的文本框 `textareal`；另一个面板 `centerPanel` 用于显示游戏界面。

```

public class LLKFrame3 extends JFrame {
    //显示已消去方块数量，由于其他类需要访问因此定义为 static
    static JTextField textareal = new JTextField(10);
    JPanel panell = new JPanel();
    LLKPanel centerPanel;
    public LLKFrame3() {
        JLabel labell = new JLabel("已消去方块数量:");
        JButton exitButton, newlyButton;
        newlyButton = new JButton("重来一局");
        exitButton = new JButton("退出");
        this.setLayout(new BorderLayout());
        panell.setLayout(new FlowLayout());
        panell.add(labell);
        panell.add(textareal);

        panell.add(newlyButton);
        panell.add(exitButton);
        textareal.setEditable(false);
        textareal.setText(Integer.toString(0)); //显示已消去方块的数量
        Container contentPane = getContentPane();
        contentPane.add(panell, BorderLayout.NORTH);
        centerPanel = new LLKPanel();
        contentPane.add(centerPanel, BorderLayout.CENTER);
        this.setBounds(280, 100, 640, 660); // 500 450
        this.setVisible(true);
        this.setFocusable(true);
        //鼠标事件监听
        exitButton.addMouseListener(new MouseAdapter() { //退出
            public void mouseClicked(MouseEvent e) {
                System.exit(0);
            }
        });
    }
}

```



```

        //鼠标事件监听
        newlyButton.addMouseListener(new MouseAdapter() { //重来一局
            public void mouseClicked(MouseEvent e) {
                textareal.setText(Integer.toString(0)); //清除已消去方块的数量
                centerPanel.StartNewGame();
                centerPanel.Init_Graphic();
            }
        });
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        LLKFrame3 llk = new LLKFrame3();
    }
}

```

8.4.2 设计游戏面板类 (LLKPanel.java)

游戏面板类 LLKPanel 实现方块的显示, 显示选中方块之间的连接示意线以及智能查找等游戏功能。

LLKFrame 面板类定义的成员变量如下:

```

private int W = 50; //动物方块图案的宽度
private int GameSize=10; //布局大小即行列数
private boolean Select_first = false; //是否已经选中第一块
private int x1, y1; //被选中第一块的地图坐标
private int x2, y2; //被选中第二块的地图坐标
private Point z1=new Point(0,0);
private Point z2=new Point(0,0); //折点棋盘坐标
private int m_nCol = 10;
private int m_nRow = 10;
private int[] m_map = new int[10*10];
private int BLANK_STATE = -1;

```

关于连通方式的枚举类型 LinkType 的定义:

```

public enum LinkType {LineType,OneCornerType,TwoCornerType};
private LinkType LType; //连通方式

```

枚举值 LineType、OneCornerType 和 TwoCornerType 分别代表直连方式、一个折点连通和两个折点连通方式。

游戏开始时调用 StartNewGame()方法实现将动物图案随机放到地图中, 地图中记录的是图案的 ID。然后调用 Init_Graphic()方法按地图中记录的图案信息将图 8-11 中的动物图案显示在游戏面板中, 生成游戏开始的界面。



图 8-11 图片 animal.bmp

```

public LLKPanel() {
    setPreferredSize(new Dimension(500, 450));
    this.addMouseListener(this); //否则鼠标单击无反应
    StartNewGame();
}

```



```

private void StartNewGame() {
    //见前文程序设计的思路
}
private void Init_Graphic()//生成游戏开始的界面
{
    Graphics g = this.getGraphics();           //生成 Graphics 对象
    for (int i = 0; i < 10 * 10; i++)
    {
        g.drawImage(create_image(m_map[i]), W * (i % GameSize)+W,
            W * (i / GameSize)+W, W, W,this);
    }
}

```

create_image()方法实现按顺序标号 n 从所有动物图案的图片 animal.bmp 中截取相应的动物图案。

```

//create_image()方法用于实现按标号 n 从所有动物图案的图片中截图
private Image create_image(int n) //按标号 n 截图
{
    int x=0;
    int y=n*39 ;
    int w=39;
    int h=39;
    BufferedImage src=null;
    BufferedImage newpic=null;
    try{
        src = ImageIO.read(new File("pic\\animal2.bmp"));
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
    newpic=src.getSubimage(x, y, w, h);//截取原图中矩形区域的图形
    return newpic;
}

```

用户在窗体游戏面板上单击时,由屏幕像素坐标(e.X,e.Y)计算被单击方块的地图位置坐标(x,y)。判断是否为第一次选中方块,若是则仅对选定方块加上黑色示意框线。如果是第二次选中方块,则需要判断是否图案相同且连通。假如连通则画选中方块之间的连接线,延时 0.5s 后,清除第一个选定方块和第二个选定方块的图案,同时清除选中方块之间的连接线。假如不连通则重新选定第一个方块。

MouseDown 事件中还需要判断用户是左键单击还是右键单击,如果是右键单击则调用智能查找功能 Find2Block()。最后调用 IsWin()方法查看是否已经赢得了游戏。

```

public void mouseClicked(MouseEvent e) {
    Graphics g = this.getGraphics();           //生成 Graphics 对象
    int x, y;
    if (e.getButton() == MouseEvent.BUTTON1)//左键单击
    {
        //计算单击的方块的位置坐标
        x = (e.getX() - W) / W;
        y = (e.getY() - W) / W;
        System.out.print(x);
        System.out.println(x);
    }
}

```



```

//如果该区域无方块
if (m_map[y * m_nCol + x] == BLANK_STATE) return;
if (Select_first == false)
{
    x1 = x; y1 = y;
    //画选定 (x1,y1) 处的框线
    DrawSelectedBlock(x1, y1, g);
    Select_first = true;
}
else
{
    x2 = x; y2 = y;
    //判断第二次单击的方块是否已被第一次单击所选取, 如果是则返回
    if ((x1 == x2) && (y1 == y2)) return;
    //画选定 (x2,y2) 处的框线
    DrawSelectedBlock(x2, y2, g);
    //判断是否连通
    if (IsSame(x1, y1, x2, y2) && IsLink(x1, y1, x2, y2))
    {
        DrawLinkLine(x1, y1, x2, y2, LType); //画选中方块之间的连接线
        System.out.println(x1+"连通"+y1);
        try
        {
            Thread.currentThread().sleep(500); //延时 0.5s
        }
        catch (Exception e1){}
        //清空记录方块的值
        m_map[y1 * m_nCol + x1] = BLANK_STATE;
        m_map[y2 * m_nCol + x2] = BLANK_STATE;
        Select_first = false;
        repaint();
    }
    else //重新选定第一个方块
    {
        //重画 (x1,y1) 处的动物图案以取消原选定 (x1,y1) 处的框线
        int i = y1 * m_nCol + x1;
        g.drawImage(create_image(m_map[i]), W * (i % GameSize)+W,
            W * (i / GameSize)+W, W, W, this);
        //设置重新选定第一个方块的坐标
        x1 = x; y1 = y;
        Select_first = true;
    }
}
}
if (e.getButton() == MouseEvent.BUTTON3) //智能查找功能
{
    if (!Find2Block())
        JOptionPane.showMessageDialog(this, "没有连通的方块了!!");
}
//查看是否已经胜利
if (IsWin())
{
    JOptionPane.showMessageDialog(this, "恭喜您胜利闯关,即将开始新局");
}
}
}

```


窗体 Paint 的重画事件重画所有非空的动物图案方块。由于在消去连接方块时，需要重画界面上所有的方块，没有动物图案（BLANK_STATE）的空白块用 g.clearRect()方法清除此处区域。由于方块周围可能存在示意连接线，因此需要用 g.clearRect()方法清除四周连线。

此处没有使用 g.clearRect(0, 0, this.getWidth(), this.getHeight())方法清屏后再重画所有动物方块，原因是使用 g.clearRect()方法清屏会产生闪烁情况。

```
// 画游戏界面
public void paint(Graphics g) {
    //g.clearRect(0, 0, this.getWidth(), this.getHeight());
    for (int i = 0; i < 10 * 10; i++)
    {
        if(m_map[i]==BLANK_STATE)//此处是空白块
            g.clearRect(W * (i % GameSize)+W, W * (i / GameSize)+W, W, W);
        else
            g.drawImage(create_image(m_map[i]), W * (i % GameSize)+W,
                W * (i / GameSize)+W, W, W,this);
    }
    //清除四周连线
    g.clearRect(0, 0, W, 12*W);
    g.clearRect(11*W, 0, W, 12*W);
    g.clearRect(0, 0, 12*W, W);
    g.clearRect(0,11*W , 12*W, W);
}
```

IsWin()方法检测是否尚有未被消除的方块，即地图 m_map 中的元素值为 BLANK_STATE，如果没有则已经赢得了游戏。

```
///
///检测是否已经赢得了游戏
///
boolean IsWin()
{
    //检测是否尚有未被消除的方块
    // (非 BLANK_STATE 状态)
    for(int i=0;i<m_nRow*m_nCol;i++)
    {
        if(m_map[i] != BLANK_STATE)
        {
            return false;
        }
    }
    return true;
}
```

IsSame()方法用来判断(x1,y1)与(x2,y2)处的方块图案是否相同。

```
private boolean IsSame(int x1, int y1,int x2, int y2)
{
    if (m_map[y1 * m_nCol + x1] == m_map[y2 * m_nCol + x2])
        return true;
    else
        return false;
}
```

以下是画方块之间的连接线、示意框线的方法。DrawLinkLine()方法画选中方块(x1,y1)与(x2,y2)之间的连接线。LinkType Ltype 参数的含义是连通方式（包括直连方式、一个折点连通

和两个折点连通方式)。

```

/// <summary>
/// 画选中方块之间的连接线
/// </summary>
private void DrawLinkLine(int x1, int y1, int x2, int y2, LinkType LType)
{
    Graphics g = this.getGraphics();    //生成 Graphics 对象
    Point p1 = new Point(x1 * W + W / 2+W, y1 * W + W / 2+W);
    Point p2 = new Point(x2 * W + W / 2+W, y2 * W + W / 2+W);
    if (LType == LinkType.LineType)
        g.drawLine(p1.x,p1.y,p2.x,p2.y);
    if (LType == LinkType.OneCornerType)
    {
        Point pixel_z1 = new Point(z1.x * W + W / 2+W, z1.y * W + W / 2+W);
        g.drawLine(p1.x,p1.y,pixel_z1.x,pixel_z1.y);
        g.drawLine(pixel_z1.x,pixel_z1.y, p2.x,p2.y);
    }
    if (LType == LinkType.TwoCornerType)
    {
        Point pixel_z1 = new Point(z1.x * W + W / 2+W, z1.y * W + W / 2+W);
        Point pixel_z2 = new Point(z2.x * W + W / 2+W, z2.y * W + W / 2+W);
        if (!(p1.x == pixel_z2.x || p1.y == pixel_z2.y))
        {
            //p1 与 pixel_z2 不在一条直线上, 则 pixel_z1, pixel_z2 交换
            Point c;
            c = pixel_z1;
            pixel_z1 = pixel_z2;
            pixel_z2 = c;
        }
        g.drawLine( p1.x,p1.y, pixel_z2.x,pixel_z2.y);
        g.drawLine( pixel_z2.x,pixel_z2.y, pixel_z1.x,pixel_z1.y);
        g.drawLine( pixel_z1.x,pixel_z1.y, p2.x,p2.y);
    }
}

```

DrawSelectedBlock() 方法用来画选中方块的示意边框线。

```

private void DrawSelectedBlock(int x, int y, Graphics g)
{
    //画选中方块的示意边框线
    Graphics2D g2 =(Graphics2D)g;    //生成 Graphics 对象
    BasicStroke s=new BasicStroke(4); //创建宽度是 4 的画笔
    g2.setStroke(s);
    g.drawRect(x * W + 1+W, y * W + 1+W, W - 3, W - 3);
}

```

init()方法用于完成窗体初始化。设置窗口大小且加入鼠标侦听器。

```

public void init() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setBounds(280, 100, 640, 660); // 500 450
    this.setTitle("连连看游戏");
    this.setVisible(true);
    this.addMouseListener(this); //否则鼠标单击无反应
    this.setFocusable(true);
}

```

至此我们已经完成了连连看游戏的开发, 运行一下程序看看效果吧!

第 9 章

人物拼图游戏

9.1 人物拼图游戏介绍

拼图游戏是将一幅图片分割成若干拼块并把它们随机打乱顺序，当将所有拼块都放回原位置时，就完成了拼图（游戏结束）。

在拼图游戏中，拼块以随机顺序排列，网格上有一个位置是空的。完成拼图的方法是利用这个空位置移动拼块，玩家用鼠标单击空位置周围的拼块来交换拼块的位置，直到所有拼块都回到原位置。拼图游戏运行界面如图 9-1。



图 9-1 拼图游戏运行界面

9.2 程序设计的思路

游戏中动态生成一个大小为 3×3 的图片按钮数组 cells。将图片 woman.jpg 分割成行列数为 3×3 的小图片，并按顺序编号；每个图片按钮显示一张小图片，其位置成员 place 存放 0~8 的整数，代表正确位置编号。注意，最后一张图片按钮显示的是空白信息图片“9.jpg”而位置成员 place 存放的是 8。

游戏开始时, 随机打乱这个图片按钮数组 cells, 根据玩家用鼠标单击的情况来交换图片按钮数组 cells 对应的按钮与空白图片按钮的位置, 根据按钮数组 cells 所有元素的位置成员 place 是否有序来判断是否已经完成游戏。

9.3 关键技术

9.3.1 按钮显示图片的实现

Swing 中的按钮可以显示图像 (图片)。

JButton 中显示图像的构造方法如下:

- JButton(Icon icon): 在按钮上显示图标。

JButton 类的方法设置不同状态下按钮显示的图片:

- setIcon(Icon defaultIcon): 设置按钮在默认状态下显示的图片。
- setRolloverIcon(Icon rolloverIcon): 设置当光标移动到按钮上方时显示的图片。
- setPressedIcon(Icon pressedIcon): 设置当按钮被按下时显示的图片。

下面是一个控制光标移动到按钮上方, 当按钮被按下时显示不同图片的示例。

```
import javax.swing.*;
import java.awt.*;

public class MyFrame extends JFrame{
    JButton button=new JButton(); //创建一个不带文本的按钮
    //JButton b = new JButton("开始");//创建一个带初始文本的按钮
    public MyFrame() {
        super();
        setTitle("利用 JFrame 创建窗体");
        setBounds(100,100,500,375);
        getContentPane().setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        button.setMargin(new Insets(0,0,0,0));
        button.setContentAreaFilled(false);
        button.setBorderPainted(false);
        button.setBounds(10,10,300,300);
        button.setIcon( new ImageIcon("001.gif"));
        button.setRolloverIcon(new ImageIcon("002.gif"));
        button.setPressedIcon(new ImageIcon("003.gif"));
        getContentPane().add(button);
    }
    public static void main(String args[]){
        MyFrame f=new MyFrame(); //创建窗口对象
        f.setVisible(true); //显示窗口
    }
}
```

程序运行结果如图 9-2、图 9-3 所示。

9.3.2 图片按钮移动的实现

当图片按钮移动后, 按钮的坐标发生改变, 此操作通过 setLocation()方法实现。setLocation()

方法是从 Component 类继承的，其定义如下：

```
public void setLocation (int x, int y)
```

参数 x 是当前组件左上角在父级坐标空间中新位置的 x 坐标，参数 y 是当前组件左上角在父级坐标空间中新位置的 y 坐标。



图 9-2 按钮按下去时显示的图片

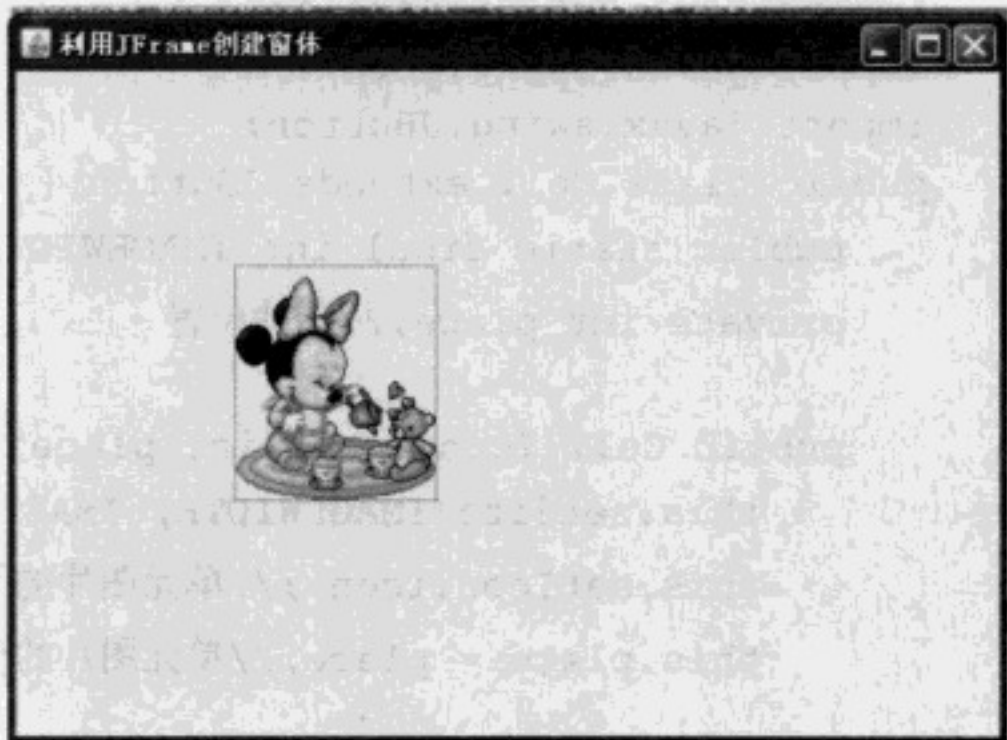


图 9-3 鼠标滚动时显示的图片

9.3.3 从 BufferedImage 转换成 ImageIcon

BufferedImage 类是 java.awt.Image 的子类，在 image 基础上增加了缓存功能。
ImageIcon 类是一个 Icon 接口的实现，它根据 Image 绘制 Icon。可以使用 URL、文件名或字节数组创建的图像。
从 BufferedImage 转换成 ImageIcon 只需要将 “ImageIcon im=new ImageIcon”（BufferedImage 实例）即可。
反之则用 ImageIcon 的 Image getImage() 方法返回此图标的 Image。

9.4 程序设计的步骤

项目组成如图 9-4 所示。

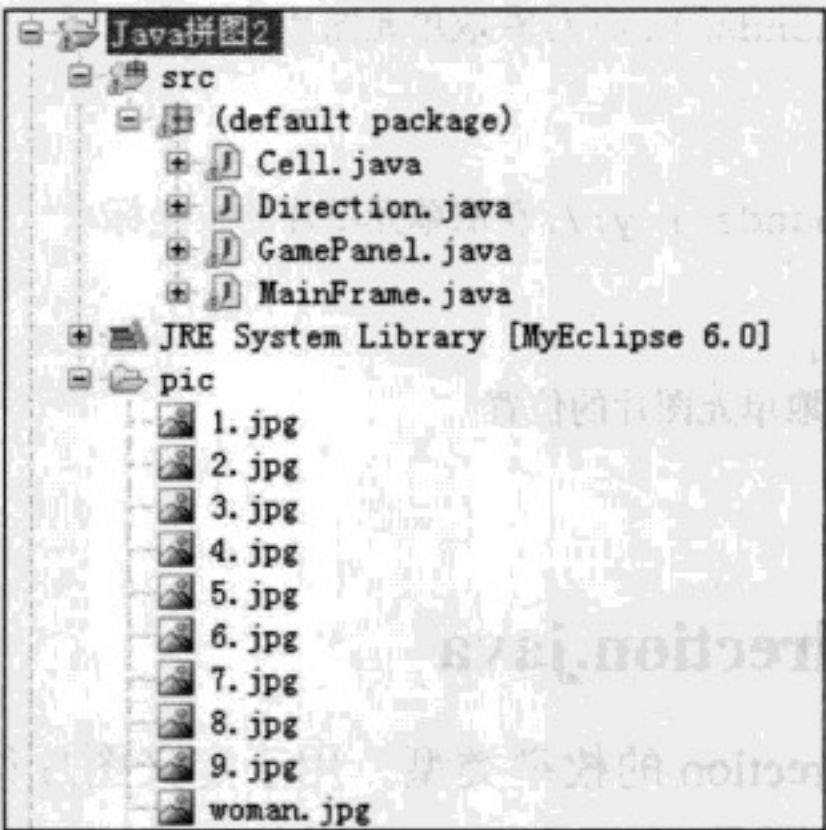


图 9-4 项目组成

9.4.1 设计单元图片类 (Cell.java)

创建名称为 Cell 的类，用于封装一个单元图片对象，此类继承 JButton 对象，并对 JButton 按钮组件进行重写，其代码如下：

```
import java.awt.Rectangle;
import javax.swing.Icon;
import javax.swing.JButton;
public class Cell extends JButton {
    public static final int IMAGEWIDTH = 100; // 图片宽度
    private int place; // 图片位置

    public Cell(Icon icon, int place) {
        this.setSize(IMAGEWIDTH, IMAGEWIDTH); // 单元图片的大小
        this.setIcon(icon); // 单元图片的图标
        this.place = place; // 单元图片的位置
    }

    public void move(Direction dir) { // 移动单元图片的方法
        Rectangle rec = this.getBounds(); // 获取图片的 Rectangle 对象
        switch (dir) { // 判断方向
            case UP: // 向上移动
                this.setLocation(rec.x, rec.y - IMAGEWIDTH);
                break;
            case DOWN: // 向下移动
                this.setLocation(rec.x, rec.y + IMAGEWIDTH);
                break;
            case LEFT: // 向左移动
                this.setLocation(rec.x - IMAGEWIDTH, rec.y);
                break;
            case RIGHT: // 向右移动
                this.setLocation(rec.x + IMAGEWIDTH, rec.y);
                break;
        }
    }

    public int getX() {
        return this.getBounds().x; // 获取单元图片的 x 坐标
    }

    public int getY() {
        return this.getBounds().y; // 获取单元图片的 y 坐标
    }

    public int getPlace() {
        return place; // 获取单元图片的位置
    }
}
```

9.4.2 枚举类型 Direction.java

项目中创建一个名称为 Direction 的枚举类型，用于定义图片移动的 4 个方向。

//枚举类型的 4 个方向


```
public enum Direction {  
    UP, // 上  
    DOWN, // 下  
    LEFT, // 左  
    RIGHT // 右  
}
```

9.4.3 游戏面板类 (GamePanel.java)

在项目中创建一个名称为 GamePanel 的类, 此类继承 JPanel 类, 实现 MouseListener 接口, 用于创建游戏面板对象。GamePanel 类中定义长度为 9 单元的图片数组对象 cells, 并通过 init() 方法对所有单元图片对象进行实例化。

```
import java.awt.Image;  
import java.awt.Toolkit;  
import java.awt.event.MouseEvent;  
import java.awt.event.MouseListener;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import java.io.IOException;  
import java.util.Random;  
import javax.imageio.ImageIO;  
import javax.swing.*;  
public class GamePanel extends JPanel implements MouseListener {  
    private Cell[] cells = new Cell[9]; // 创建单元图片数组  
    private Cell cellBlank = null; // 空白
```

构造方法 GamePanel() 通过调用 init() 初始化方法对所有单元图片对象进行实例化。对单元图片对象进行实例化可以直接用已经分割好的图片 1.jpg、2.jpg……9.jpg (如图 9-5 所示) 实现, 其中 9.jpg 为空白图片。

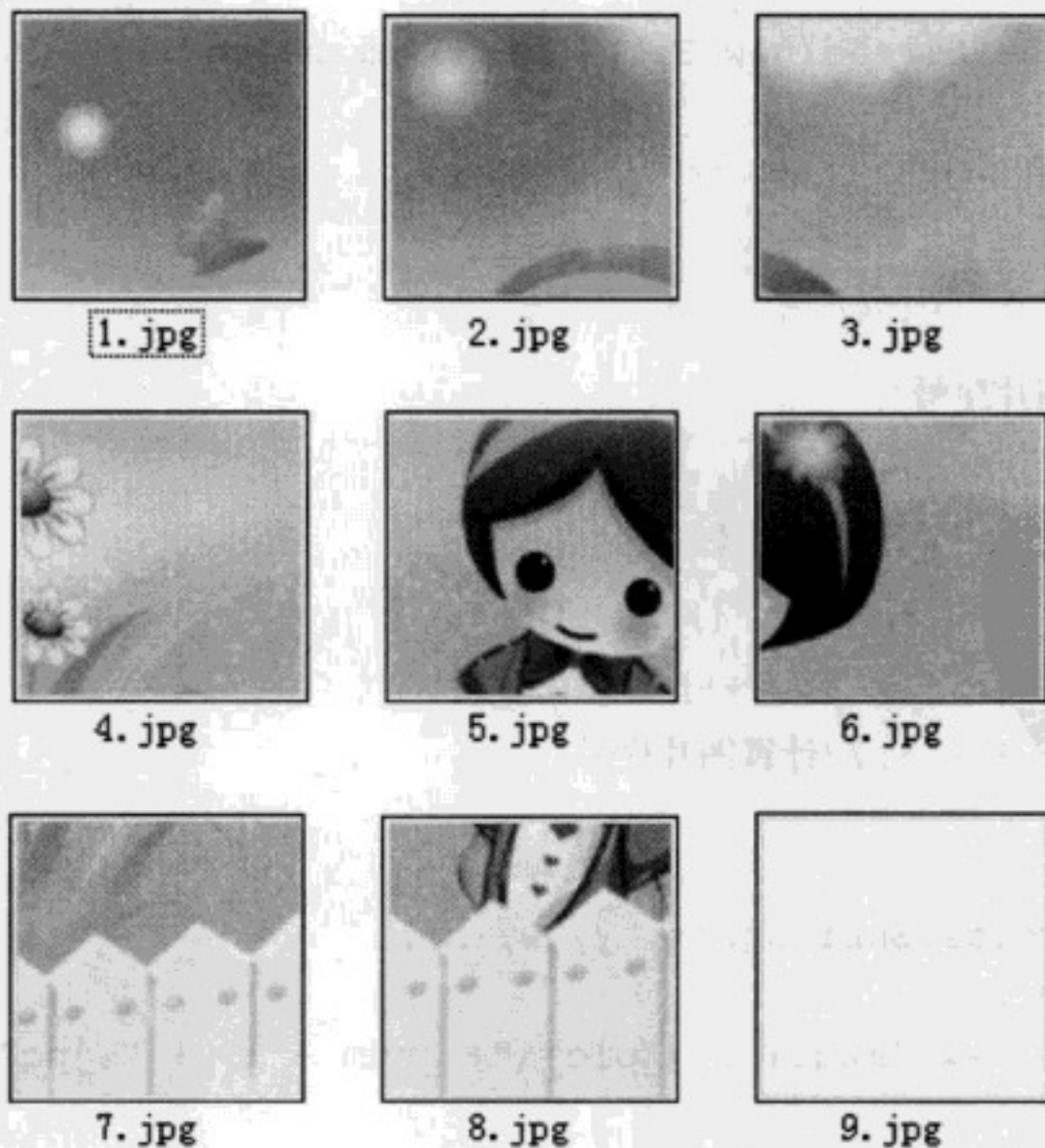


图 9-5 拼图游戏所用图片

单元图片对象直接用已经分割好的图片进行实例化，代码如下：

```
for (int i = 0; i < 3; i++) { //循环行
    for (int j = 0; j < 3; j++) { //循环列
        num = i * 3 + j; //计算图片序号
        icon = new ImageIcon("pic\\" + (num + 1) + ".jpg"); //获取图片
        cell = new Cell(icon, num); //实例化单元图片对象
    }
}
```

也可以不用已经分割好的图片，使用 `BufferedImage` 类的 `getSubimage()` 方法可以对一张大的图片 `woman.jpg` 分割成任意子图像。

//返回由指定矩形区域定义的子图像。

```
BufferedImage getSubimage(int x, int y, int w, int h)
```

使用时先得到原图片的长和宽，根据要求需要分成多少块，就能算出每块的 x, y 坐标，这样就可以分割了。注意分解出来的是 `BufferedImage` 对象，而按钮的图片需要是 `ImageIcon` 类型，从 `BufferedImage` 转换成 `ImageIcon` 只需要将 “`ImageIcon im=new ImageIcon`”（`BufferedImage` 对象）即可。

```
public GamePanel() { //构造方法
    super();
    setLayout(null); //设置空布局
    init(); //初始化
}

public void init() { //初始化游戏
    int x=0;
    int y=0;
    int w=110;
    int h=110;
    BufferedImage src=null;
    BufferedImage newpic=null;
    try{
        src = ImageIO.read(new File("pic\\woman.jpg"));
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
    int num = 0; //图片序号
    Icon icon = null; //图标对象
    Cell cell = null; //单元图片对象
    for (int i = 0; i < 3; i++) { //循环行
        for (int j = 0; j < 3; j++) { //循环列
            num = i * 3 + j; //计算图片序号
            x=j*100;
            y=i*100;
            newpic=src.getSubimage(x, y, w, h);
            if(num+1==9)
                icon = new ImageIcon("pic\\" + (num + 1) + ".jpg"); //获取空白图片
            else
                icon=new ImageIcon(newpic);
            cell = new Cell(icon, num); //实例化单元图片对象
        }
    }
}
```



```

        //设置单元图片的坐标
        cell.setLocation(j * Cell.IMAGEWIDTH, i * Cell.IMAGEWIDTH);
        cells[num] = cell;//将单元图片存储到单元图片数组中
    }
}
for (int i = 0; i < cells.length; i++) {
    this.add(cells[i]); //向面板中添加所有单元图片
}
}

```

random()方法对图片进行随机排序,产生两个(0~8)随机数m,n作为被交换图片按钮数组元素的下标,对调此两个被交换图片按钮的位置,代码如下:

```

public void random() {
    Random rand = new Random();//实例化 Random
    int m, n, x, y;
    if (cellBlank == null) { //判断空白的图片位置是否为空
        cellBlank = cells[cells.length - 1]; //最后一个作为空白的图片按钮
        for (int i = 0; i < cells.length; i++) { //遍历所有单元图片
            if (i != cells.length - 1) {
                cells[i].addMouseListener(this); //对非空白图片注册鼠标监听
            }
        }
    }
    for (int i = 0; i < cells.length; i++) { //遍历所有单元图片
        m = rand.nextInt(cells.length); //产生随机数
        n = rand.nextInt(cells.length); //产生随机数
        x = cells[m].getX(); //获取 x 坐标
        y = cells[m].getY(); //获取 y 坐标
        //对单元图片进行调换
        cells[m].setLocation(cells[n].getX(), cells[n].getY());
        cells[n].setLocation(x, y);
    }
}

```

图片块上的单击事件中,通过e.getSource()方法获取触发事件的对象cell,与空白图片块cellBlank进行位置比较,从而决定被单击对象cell和空白图片块cellBlank的移动方向。

```

public void mousePressed(MouseEvent e) {
}
public void mouseReleased(MouseEvent e) {
}
public void mouseClicked(MouseEvent e) {
    Cell cell = (Cell) e.getSource(); //获取触发事件的对象
    int x = cellBlank.getX(); //获取空白图片块的 x 坐标
    int y = cellBlank.getY(); //获取空白图片块的 y 坐标
    if ((x - cell.getX()) == Cell.IMAGEWIDTH && cell.getY() == y) {
        cell.move(Direction.RIGHT); //向右移动
        cellBlank.move(Direction.LEFT);
    } else if ((x - cell.getX()) == -Cell.IMAGEWIDTH && cell.getY() == y) {
        cell.move(Direction.LEFT); //向左移动
        cellBlank.move(Direction.RIGHT);
    } else if (cell.getX() == x && (cell.getY() - y) == Cell.IMAGEWIDTH) {

```



```

        cell.move(Direction.UP); //向上移动
        cellBlank.move(Direction.DOWN);
    } else if (cell.getX() == x && (cell.getY() - y) == -Cell.IMAGEWIDTH) {
        cell.move(Direction.DOWN); //向下移动
        cellBlank.move(Direction.UP);
    }
    if (isSuccess()) { //判断是否拼图成功
        int i = JOptionPane.showConfirmDialog(this, "成功, 再来一局?", "拼图成功",
        JOptionPane.YES_NO_OPTION); //提示成功
        if (i == JOptionPane.YES_OPTION) {
            random(); //开始新一局
        }
    }
}
public void mouseEntered(MouseEvent e) {
}
public void mouseExited(MouseEvent e) {
}

```

isSuccess()方法判断游戏是否成功, 只需判断图片块原始位置 cells[i].getPlace()是否符合现在的位置, 只要有一个单元图片的位置不正确, 就返回 False; 当所有单元图片的位置都正确则返回 True。

```

public boolean isSuccess() { //判断是否拼图成功
    for (int i = 0; i < cells.length; i++) { // 遍历所有单元图片
        int x = cells[i].getX(); //获取 x 坐标
        int y = cells[i].getY(); //获取 y 坐标
        if (i != 0) {
            if (y / Cell.IMAGEWIDTH * 3 + x / Cell.IMAGEWIDTH != cells[i]
                .getPlace()) { //判断单元图片位置是否正确
                return false; //只要有一个单元图片的位置不正确, 就返回 False
            }
        }
    }
    return true; //所有单元图片的位置都正确则返回 True
}
}

```

9.4.4 主窗口类 (MainFrame.java)

在项目中创建一个继承 JFrame 的 MainFrame 类, 用于显示自定义游戏面板 GamePanel 界面。

代码如下:

```

import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class MainFrame extends JFrame {
    public static void main(String args[]) {
        EventQueue.invokeLater(new Runnable() {

```



```

        public void run() {
            try {
                MainFrame frame = new MainFrame();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

public MainFrame() {
    super();
    getContentPane().setLayout(new BorderLayout());
    setTitle("拼图游戏");
    setBounds(300, 300, 358, 414);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    final JPanel panel = new JPanel();           //实例化 JPanel
    getContentPane().add(panel, BorderLayout.NORTH); //添加到上方
    final GamePanel gamePanel = new GamePanel(); //实例化游戏面板
    //添加到中央位置
    getContentPane().add(gamePanel, BorderLayout.CENTER);
    final JButton button = new JButton();        //实例化按钮
    //注册事件
    button.addActionListener(new ActionListener() {
        public void actionPerformed(final ActionEvent e) {
            //开始游戏
            gamePanel.random();
        }
    });
    button.setText("开始");
    panel.add(button);
}
}

```

拼图游戏总体设计情况如前所述,并没有很高深的内容,实现的核心在于对按钮数组的操作。拼图游戏成功的界面如图 9-6 所示。



图 9-6 拼图游戏成功的界面

第 10 章

对对碰游戏（按钮版）

10.1 对对碰游戏介绍

对对碰游戏在 8×8 格子的游戏池中进行，每个格子中有一个图像。玩家用鼠标连续选中两个相邻的图像，它们的位置会互换，互换后如果横排或竖排有 3 个以上相同的图像，则可以消去该图像并得分。

游戏基本规则如下所述。

(1) 交换

玩家选中相邻（横、竖）的两个图像，则这两个图像的位置发生互换，如果互换成功则消去图像，否则取消位置交换。

(2) 消去

玩家选择两个图像进行位置互换，互换后如果横排或竖排有 3 个以上相同的图像，则消去这几个相同的图像；如果互换后没有可以消去的图像，则选中的两个图像换回原来的位置。消去后的空格由上面的图像掉下来补齐。每次消去图像玩家都能得到一定的分数。

(3) 连锁

玩家消去图像后，上面的图像掉下来补充空格。如果这时游戏池中有连续摆放（横、竖）的 3 个或 3 个以上相同的图像，则可以消去这些图像，这就是一次连锁。空格被新的图像填充，又可以进行一次连锁。每次连锁会有加分。

本章开发的对对碰游戏开始界面如图 10-1 所示，用户按“开始”按钮开始游戏，直到窗口上方的时间进度条为 100% 时游戏结束。游戏过程中将不断刷新并显示玩家的得分。游戏开始界面如图 10-2 所示。

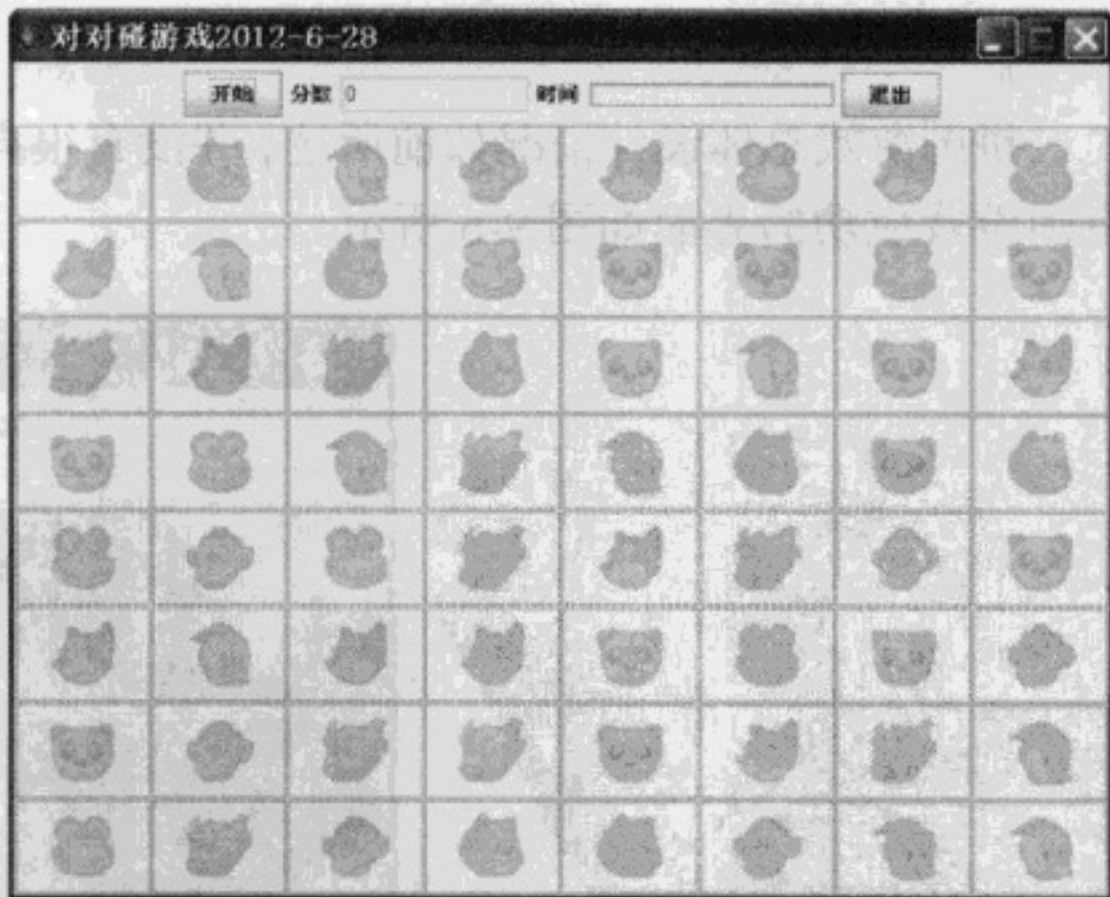


图 10-1 对对碰游戏界面



图 10-2 游戏开始界面

10.2 程序设计的思路

10.2.1 游戏素材

对对碰游戏中用到的游戏池、动物等图案，分别使用图 10-3 所示的图片表示。

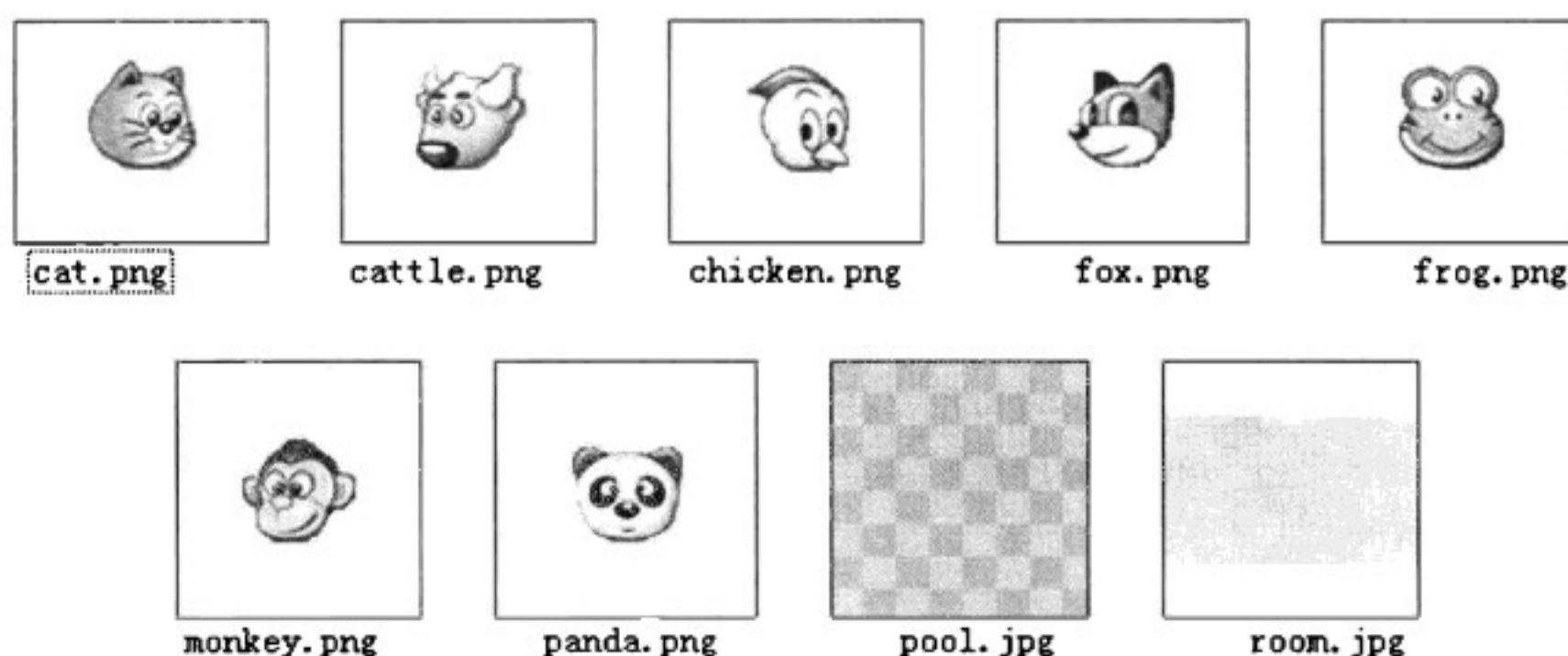


图 10-3 相关图片素材

10.2.2 设计思路

游戏屏幕由 8 行 8 列的方块组成，方块的动物图案各不相同，为显示方块的动物图案，采用图形按钮实现。因为屏幕由 8 行 8 列的方块组成，所以采用二维 JButton 数组 `button[8][8]` 来表示；为了方便判断横排或竖排有 3 个以上相同的图像按钮，这里使用二维整型数组 `animal[8][8]`，来存储对应按钮的动物图案 ID（0~6 的数字）。

在定时器 `timer` 的控制下，不停地统计用户的得分，并控制时间进度条，如果生命进度条为 100%，则游戏结束，出现游戏结束提示框。

当用户用鼠标连续选中两个相邻的方块（图形按钮）`button[y2][x2]` 和 `button[y1][x1]` 时，将交

换二维数组 `animal` 中两个按钮的动物图案 ID，而不是交换位置。交换以后调用 `isThreeLinked(y2, x2)` 和 `isThreeLinked(y1, x1)` 方法检测屏幕上是否有符合消去规则的方块，如果有被消去的方块，则 `removeLinked(y2, x2)` 方法修改并记录要绘制方块的动物图案 ID 的 `animal[8][8]` 数组对应元素的值，不需要绘制记为 `EMPTY`（即 7，因为动物图案 ID 是 0~6 的数字），并调用 `updateAnimal()` 方法从游戏屏幕该列上方重新随机产生新的动物图案 ID，更新动物图案 ID 数组 `animal[8][8]`。最后 `print()` 方法更新所有方块按钮的图形 `Icon`，从而得到动态游戏效果。

10.3 关键技术

10.3.1 动态生成 8×8 的按钮

本章 8 行 8 列的方块是由按钮实现的，Java 能实现这种图形化按钮，仅使用 `Jbutton` 相关方法就可以实现，主要代码如下：

```
Button button= new JButton( );
ImageIcon exitedImageIcon =new ImageIcon("res / exited.png");
ImageIcon enteredImageIcon =new ImageIcon("res / roll.png");
ImageIcon pressedImageIcon =new ImageIcon("res / down.png");
button.setIcon(exitedImageIcon);           //设置鼠标不在按钮上时的图标
button.setRolloverIcon(enteredImageIcon);  //设置鼠标移到按钮上时的图标
button.setPressedIcon(pressedImageIcon);   //设置鼠标单击时的图标
button.setContentAreaFilled(false); //是否显示外围矩形区域，设置为否
button.setFocusable(false); //去掉按钮的聚焦框
button.setBorderPainted(false); //去掉边框
```

下面是一个控制光标移动到按钮上方，当按钮被按下时显示不同图片的示例。

```
import javax.swing.*;
import java.awt.*;
public class MyFrame extends JFrame{
    JButton button=new JButton(); //创建一个不带文本的按钮
    //JButton b = new JButton("开始");//创建一个带初始文本的按钮
    public MyFrame() {
        super();
        setTitle("利用 JFrame 创建窗体");
        setBounds(100,100,500,375);
        getContentPane().setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        button.setMargin(new Insets(0,0,0,0));
        button.setContentAreaFilled(false);
        button.setBorderPainted(false);
        button.setBounds(10,10,300,300);
        button.setIcon( new ImageIcon("001.gif"));
        button.setRolloverIcon(new ImageIcon("002.gif"));
        button.setPressedIcon(new ImageIcon("003.gif"));
        getContentPane().add(button);
    }
    public static void main(String args[]){
        MyFrame f=new MyFrame(); //创建窗口对象
        f.setVisible(true); //显示窗口
    }
}
```



```
    }
}
```

10.3.2 进度条组件（JProgressBar）

使用 JProgressBar 类创建进度条组件。该组件能用一种颜色动态地填充自身，以便显示某任务完成的百分比。

构造方法如下：

- JProgressBar(): 创建一个显示边框但不带进度字符串的水平进度条。
- JProgressBar(BoundedRangeModel newModel): 创建使用指定的保存进度条数据模型的水平进度条。
- JProgressBar(int orient): 创建具有指定方向（JProgressBar.VERTICAL 或 JProgressBar.HORIZONTAL）的进度条。
- JProgressBar(int min, int max): 创建具有指定最小值和最大值的水平进度条。
- JProgressBar(int orient, int min, int max): 创建使用指定方向、最小值和最大值的进度条。

常用方法如下：

- public void setValue(int n): 将进度条的当前值设置为 n。
- public int getValue(): 返回进度条的当前值。
- setMinimum(int min): 改变最小值。
- setMaximum(int max): 改变最大值。

10.3.3 定时器功能

Timer 组件可以定时执行任务，这在游戏动画编程上非常有用。Timer 组件可以用 javax.swing.Timer 包中的 Timer 类来实现，该类的构造方法为：

```
Timer(int delay, ActionListener listener);
```

该构造方法用于建立一个 Timer 组件对象，参数 listener 用于指定一个接收该计时器操作事件的侦听器，指定所要触发的事件；参数 delay 用于指定每一次触发事件之间的时间间隔。也就是说，Timer 组件会根据用户所指定的 delay 时间，周期性地触发(ActionEvent)事件。如果要处理这个事件，就必须实现 ActionListener 接口类，以及接口类中的 actionPerformed()方法。

例如，开始按钮事件代码中创建 Timer 组件对象的代码如下：

```
if (e.getSource() == buttona) { //开始 buttona
    timer = new Timer(800, new TimeListener());
    timer.start();
}
```

本程序内部定时器类 TimeListener 修改进度条的状态，并判断是否达到最大值 100，如果达到则定时器结束，8×8 的图形按钮无效，而“开始”按钮有效，从而可以开始新游戏。

10.4 程序设计的步骤

10.4.1 设计游戏窗口类（MyJframes.java）

游戏窗口类 MyJframes 实现游戏的全部功能，继承 JFrame 组件实现，由上方 Panel1 和中间

Panel2 组成。

导入包及相关类：

```
import java.awt.*;
import java.util.Random;
import javax.swing.*;
```

游戏窗口类 MyJframes 定义的成员变量如下：

```
public class MyJframes extends JFrame {
    private JPanel panell = new JPanel();
    private JButton buttona = new JButton("开始");
    private JLabel labell = new JLabel("分数");
    private JTextField textareal = new JTextField(10);
    private JLabel buttonc = new JLabel("时间");
    private JProgressBar jindu = new JProgressBar();
    private Timer timer;           //定时器，控制时间进度条
    private JButton buttonb = new JButton("退出");
    private JPanel panel2 = new JPanel();
    private JButton button[][] = new JButton[8][8];
    private int animal[][] = new int[8][8];
    private ImageIcon Iocn[] = new ImageIcon[7];
    private final int EMPTY = 0;    //无为 0，有为 1
    private Random rand = new Random(); //随机数
    private boolean isThreeLinked;  //标记是否有 3 个以上的连接
    private boolean isDoubleClicked; //标记单击次数
    private int x1;                 //记录第一次被单击按钮的 x 坐标
    private int y1;                 //记录第一次被单击按钮的 y 坐标
    private int grade = 0;          //得分
}
```

游戏窗口类 MyJframes 的构造方法加载所有动物图片形成 Iocn[7]数组，同时上方 Panel1 加入“开始”按钮和“退出”按钮及进度条。中间 Panel2 动态添加 8×8 的按钮，并随机产生按钮图标，图标的动物图案 ID 记录在 animal 数组中，同时将这些 8×8 的按钮设置为无效。代码如下：

```
MyJframes() {
    //加载图片
    Iocn[0] = new ImageIcon("image//cat.png");
    Iocn[1] = new ImageIcon("image//cattle.png");
    Iocn[2] = new ImageIcon("image//chicken.png");
    Iocn[3] = new ImageIcon("image//fox.png");
    Iocn[4] = new ImageIcon("image//monkey.png");
    Iocn[5] = new ImageIcon("image//panda.png");
    Iocn[6] = new ImageIcon("image//frog.png");
    panell.setLayout(new FlowLayout());
    panell.add(buttona);
    panell.add(labell);
    panell.add(textareal);
    textareal.setEditable(false);
    textareal.setText(Integer.toString(grade)); //
    panell.add(buttonc);
    jindu.setMaximum(100);
    panell.add(jindu);
    panell.add(buttonb);
    this.setLayout(new BorderLayout());
}
```



```

this.add(panel1, BorderLayout.NORTH);
panel2.setLayout(new GridLayout(8, 8, 1, 1));
MouseListener mylisten = new MyListener();//自定义监听器类
int m;
//初始化动物数组
for (int i = 0; i < 8; i++)
    for (int j = 0; j < 8; j++) {
        m = (int) (Math.random() * 7);
        button[i][j] = new JButton(Iocn[m]);
        animal[i][j] = m;
        button[i][j].setSize(50, 50);
        //为按钮添加侦听
        button[i][j].addActionListener(mylisten);
        button[i][j].setEnabled(false); //设置图形按钮无效
        panel2.add(button[i][j]);
    }
this.add(panel2, BorderLayout.CENTER);
buttona.addActionListener(mylisten);
buttonb.addActionListener(mylisten);
}

```

isThreeLinked(int x, int y)方法用来判断(x, y)处附近是否有 3 个以上连续相同的方块按钮。代码如下:

```

private boolean isThreeLinked(int x, int y) { // 判断是否有 3 个以上连续相同的方块按钮
    int tmp;
    int linked = 1;
    if (x + 1 < 8) {
        tmp = x + 1;
        while (tmp < 8 && animal[x][y] == animal[tmp][y]) {
            linked++;
            tmp++;
        }
    }
    if (x - 1 >= 0) {
        tmp = x - 1;
        while (tmp >= 0 && animal[x][y] == animal[tmp][y]) {
            linked++;
            tmp--;
        }
    }
    if (linked >= 3) {
        return true;
    }
    linked = 1;
    if (y + 1 < 8) {
        tmp = y + 1;
        while (tmp < 8 && animal[x][y] == animal[x][tmp]) {
            linked++;
            tmp++;
        }
    }
    if (y - 1 >= 0) {
        tmp = y - 1;
        while (tmp >= 0 && animal[x][y] == animal[x][tmp]) {
            linked++;

```



```

        tmp--;
    }
}
if (linked >= 3) {
    return true;
}
return false;
}

```

removeLinked(int x, int y)方法将(x, y)附近有 3 个以上连续相同的方块按钮的图案置为空 (EMPTY), 并根据数量计算用户的得分, 每消去一块加 10 分。代码如下:

```

private void removeLinked(int x, int y) {
    if (animal[x][y] == EMPTY) return;
    int n=0;
    int tmp;
    int linked = 1;
    if (x + 1 < 8) {
        tmp = x + 1;
        while (tmp < 8 && animal[x][y] == animal[tmp][y]) {
            linked++;
            tmp++;
        }
    }
    if (x - 1 >= 0) {
        tmp = x - 1;
        while (tmp >= 0 && animal[x][y] == animal[tmp][y]) {
            linked++;
            tmp--;
        }
    }
    if (linked >= 3) {
        n=n+linked;
        tmp = x + 1;
        while (tmp < 8 && animal[tmp][y] == animal[x][y]) {

            animal[tmp][y] = EMPTY;
            tmp++;
        }
        tmp = x - 1;
        while (tmp >= 0 && animal[tmp][y] == animal[x][y]) {

            animal[tmp][y] = EMPTY;
            tmp--;
        }
        //当前交换过来的点
        animal[x][y] = EMPTY;
    }
    tmp = 0;
    linked = 1;
    if (y + 1 < 8) {
        tmp = y + 1;
        while (tmp < 8 && animal[x][y] == animal[x][tmp]) {
            linked++;
            tmp++;
        }
    }
}

```



```

    if (y - 1 >= 0) {
        tmp = y - 1;
        while (tmp >= 0 && animal[x][y] == animal[x][tmp]) {
            linked++;
            tmp--;
        }
    }
    if (linked >= 3) {
        n=n+linked;
        tmp = y + 1;
        while (tmp < 8 && animal[x][y] == animal[x][tmp]) {
            animal[x][tmp] = EMPTY;
            tmp++;
        }
        tmp = y - 1;
        while (tmp >= 0 && animal[x][y] == animal[x][tmp]) {
            animal[x][tmp] = EMPTY;
            tmp--;
        }
        //当前交换过来的点
        animal[x][y] = EMPTY;
    }
    grade+=n*10;
    textareal.setText(Integer.toString(grade));
}

```

globalSearch(int flag)方法全盘扫描是否有 3 个以上连续相同的方块按钮,当参数 flag=1 时仅扫描;当参数 flag=2 时则将 3 个相连的相同的方块按钮图案置为空。代码如下:

```

private boolean globalSearch(int flag) {
    if (flag ==1) {
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                if (isThreeLinked(i, j)) {
                    return true;
                }
            }
        }
    } else if (2 == flag) {
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                (i, j)//将三个以上连续相同的方块按钮的图案置为空
                removeLinked(i, j);
            }
        }
    }
    return false;
}

```

downAnimal()方法从游戏屏幕该列底部依次下移上方的方块来填充被消去的方块(EMPTY)。代码如下:

```

//动物下降
private void downAnimal() {
    int tmp;
    for (int j = 8 - 1; j >= 0; j--) {

```



```

        for (int i = 0; i < 8; i++) {
            if (animal[j][i] == EMPTY) {
                for (int k = j - 1; k >= 0; k--) {
                    if (animal[k][i] != EMPTY) {
                        tmp = animal[k][i];
                        animal[k][i] = animal[j][i];
                        animal[j][i] = tmp;
                        break;
                    }
                }
            }
        }
    }
}

```

print()方法重新显示设置按钮的图形 Icon，从而得到动态游戏效果。代码如下：

```

private void print() {
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            button[i][j].setIcon(Iocn[animal[i][j]]);
        }
    }
}

```

对对碰游戏中 swapAnimal(int x, int y)方法交换选中的两个相邻方块，参数(x1,y1)为第一个选定方块按钮数组的坐标，参数(x2,y2)为第二个选定方块按钮数组的坐标。因此 animal[x1][y1]和 animal[x2][y2]的交换即可以实现图案交换。

对对碰游戏中需要检测在交换两个被选中的相邻方块后，横排或竖排是否有 3 个以上方块对象有相同的图像。isThreeLinked(y2, x2)和 isThreeLinked(y1, x1)方法完成检测是否有可以消除的方块对象。如果可以消去方块，则 removeLinked()方法修改并记录要绘制方块的动物图案 ID 的 animal[8][8]数组对应元素的值，不需要绘制记为 EMPTY(即 7, 因为动物图案 ID 是 0~6 的数字)，并调用 updateAnimal()方法从游戏屏幕该列上方重新随机产生新的动物图案 ID，更新动物图案 ID 数组 animal[8][8]。最后 print()方法重新显示需要绘制的所有方块图形 Icon，从而得到动态游戏效果。代码如下：

```

private void swapAnimal(int x, int y) {
    if ((x >= 0 && x <= 8) && (y >= 0 && y <= 8)) {
        //被单击的动物图案的坐标
        int x2;
        int y2;
        if (!isDoubleClicked) { //第一次单击
            isDoubleClicked = true;
            x1 = x;
            y1 = y;
            System.out.println("被单击的点的 X 坐标 = " + x1);
            System.out.println("被单击的点的 Y 坐标 = " + y1);
        } else { //第二次单击
            x2 = x;
            y2 = y;
            isDoubleClicked = false;
            //两点的坐标绝对值等于 1 时视为相邻的两点
            if (1 == Math.abs(x2 - x1) && y2 == y1

```



```

do {
    System.out.println("重新初始化");
    initAnimalMatrix();
} while (globalSearch(1));
print();
pack();
setResizable(false);
setVisible(true);
}

```

由于随机产生按钮图标可能会出现横排或竖排有 3 个以上连续相同的图像，而 initAnimalMatrix()方法可以再次重新产生按钮图标的动物图案。代码如下：

```

//初始化动物图案数组
private void initAnimalMatrix() {
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            // 随机选取动物图案
            animal[i][j] = rand.nextInt(7);
        }
    }
}

```

监听器类用来判断产生动作的来源是哪个按钮，如果是“开始”按钮 buttona 则定时器 new Timer(800, new TimeListener())启动；如果是“退出”按钮 buttonb 则结束程序。其余的可能就是 8 × 8 的动物按钮数组产生的问题，调用 swapAnimal(j, i)方法实现按钮图形交换并消去横排或竖排有 3 个以上连续相同的图像。代码如下：

```

class MyListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == buttona) { // “开始”按钮 buttona
            buttona.setEnabled(false);
            jindu.setStringPainted(true);
            jindu.setMaximum(100);
            jindu.setMinimum(0);
            timer = new Timer(800, new TimeListener());
            timer.start();
            grade = 0;
            textareal.setText(Integer.toString(grade));
            for (int i = 0; i < 8; i++)
                for (int j = 0; j < 8; j++) {
                    button[i][j].setEnabled(true); //图形按钮有效
                }
        }
        if (e.getSource() == buttonb) { // “退出”按钮 buttonb
            System.out.println("end");
            System.exit(1);
        }
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                if (e.getSource() == button[i][j]) { // button[i][j]为动物按钮数组
                    System.out.println("第" + i + " " + j + "键");
                    swapAnimal(j, i);
                }
            }
        }
    }
}

```



```
    }  
}
```

10.4.2 设计内部定时器类

内部定时器类 TimeListener 修改进度条的状态，并判断是否达到最大值 100，如果达到则定时器结束，8×8 的图形按钮无效，而“开始”按钮有效，从而可以开始新游戏。代码如下：

```
class TimeListener implements ActionListener {  
    int times = 0;  
    public void actionPerformed(ActionEvent e) {  
        jindu.setValue(times++);  
        if (times > 100) {  
            timer.stop(); //定时器结束  
            for (int i = 0; i < 8; i++)  
                for (int j = 0; j < 8; j++) {  
                    button[i][j].setEnabled(false); //图形按钮无效  
                }  
            buttona.setEnabled(true);  
        }  
    }  
}
```

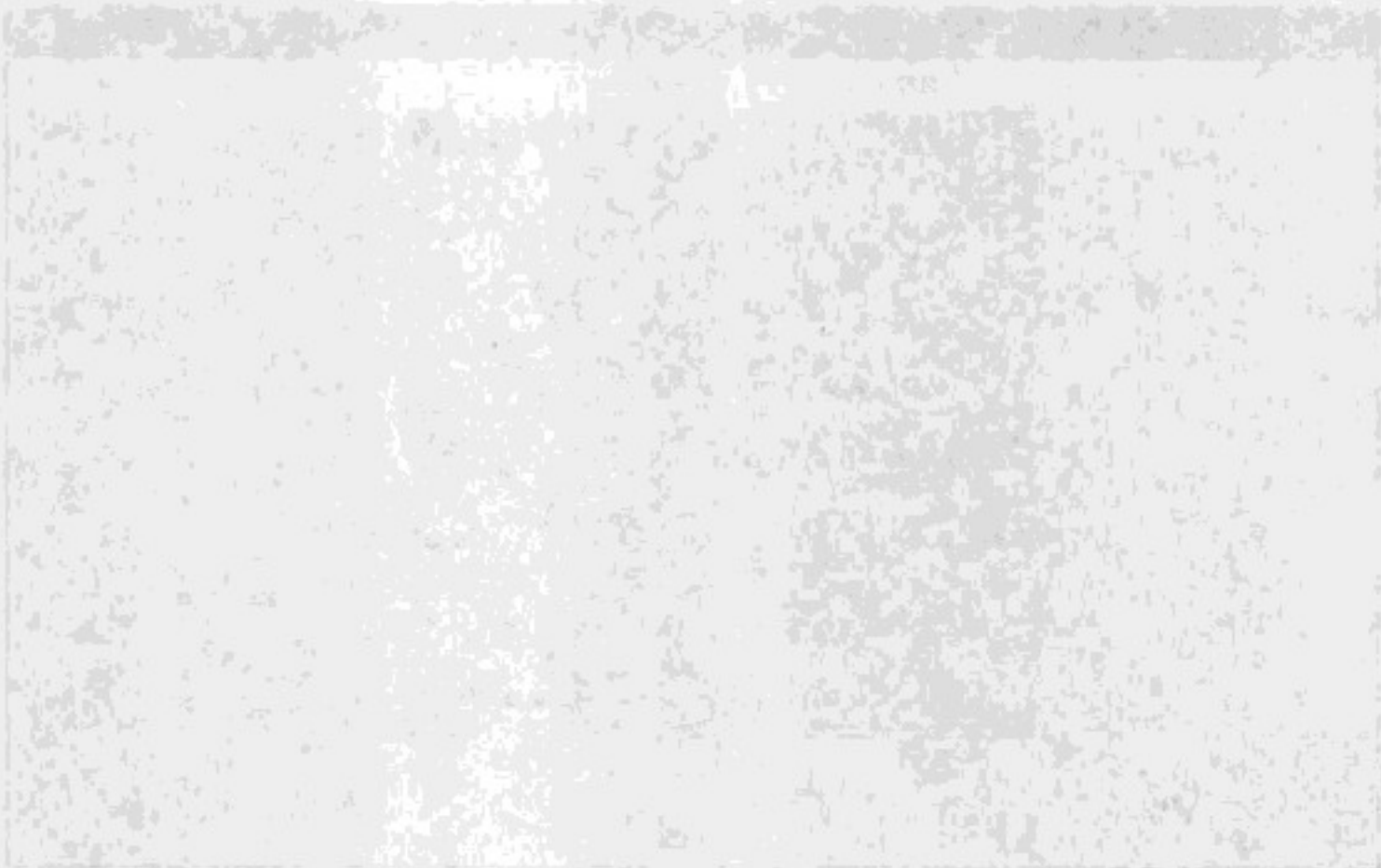


图 10-11 对对碰游戏（按钮版）主窗体界面（1/1）图

对对碰游戏（按钮版）主窗体界面（1/1）图

对对碰游戏（按钮版）主窗体界面（1/1）图

第 11 章

对对碰游戏（图形版）

11.1 对对碰游戏介绍

对对碰游戏在 8×8 格子的游戏池中进行，每个格子中有一个图像。玩家用鼠标连续选中两个相邻的图像，它们的位置会互换，互换后如果横排或竖排有 3 个以上相同的图像，则可以消去该图像并得分。游戏基本规则与第 10 章相同。

本章开发的游戏开始界面如图 11-1 所示，用户开始游戏后，直到窗口上方的时间值为 0 秒时结束。消掉图像方块可以增加用户的得分（每消去一块用户的得分增加 10 分）。

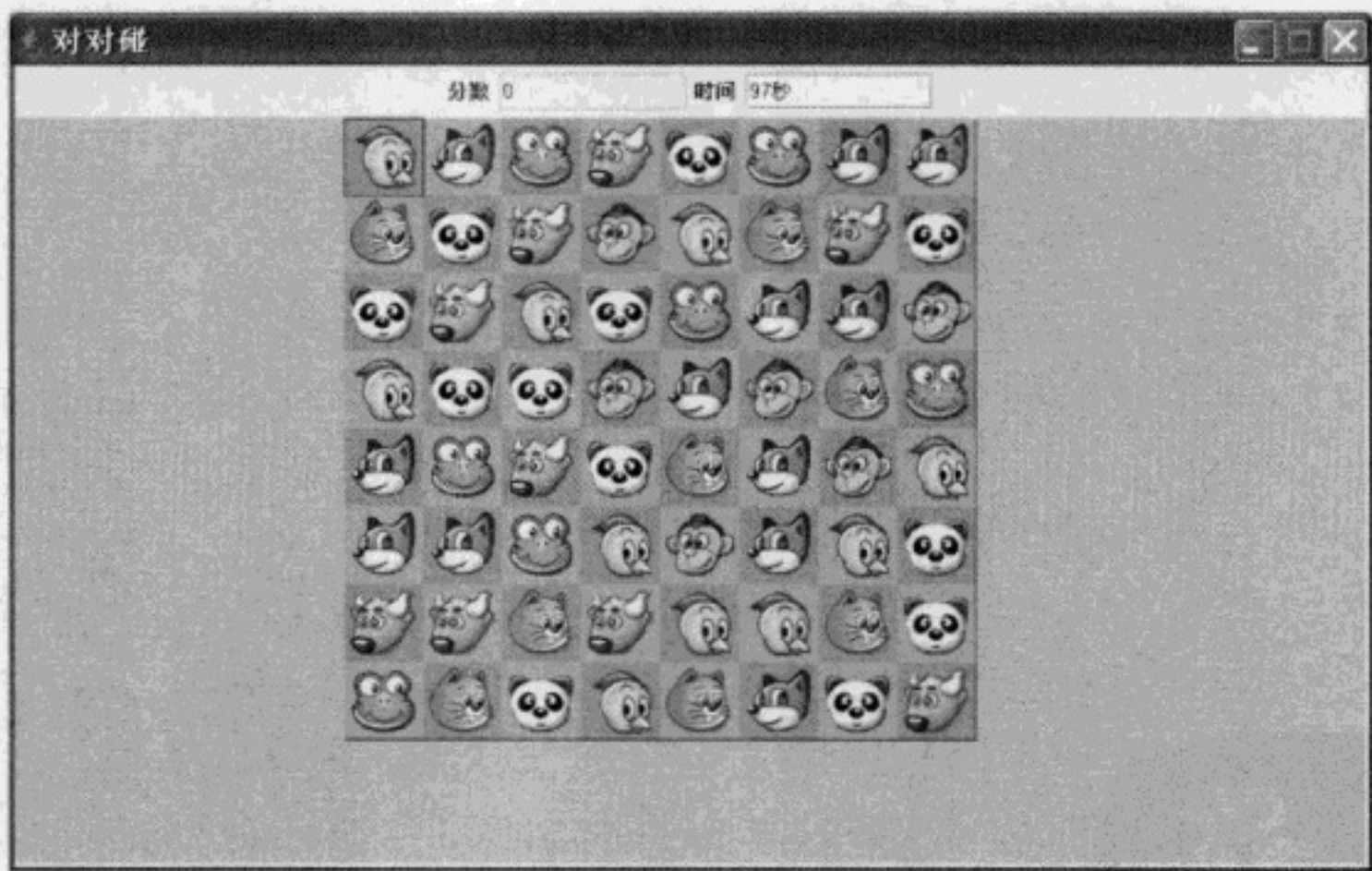


图 11-1 对对碰游戏（图形版）开始界面

11.2 对对碰游戏设计思路

11.2.1 游戏素材

对对碰游戏中用到的游戏池、动物等图案，分别使用图 11-2 所示的图片表示。

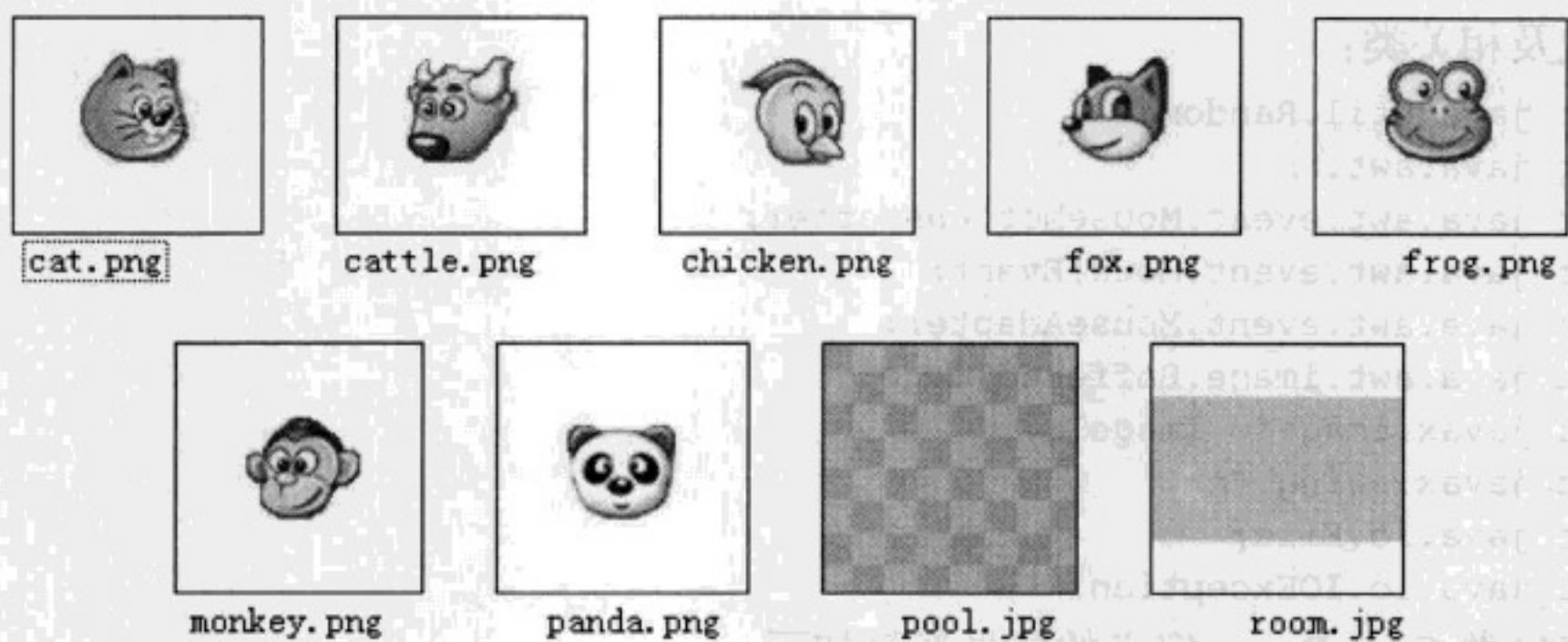


图 11-2 相关图片素材

11.2.2 设计思路

游戏屏幕由 8 行 8 列的方块组成，方块的动物图案各不相同。为了存储游戏画面中方块动物的图案，这里采用二维数组 `animal [8,8]`，来存储对应按钮的动物图案 ID（0~6 的数字）。在定时器 `timer` 的控制下，不停地更新剩余时间，如果剩余时间为 0 秒，则游戏结束，出现图 11-3 所示的游戏结束对话框提示用户是否继续。

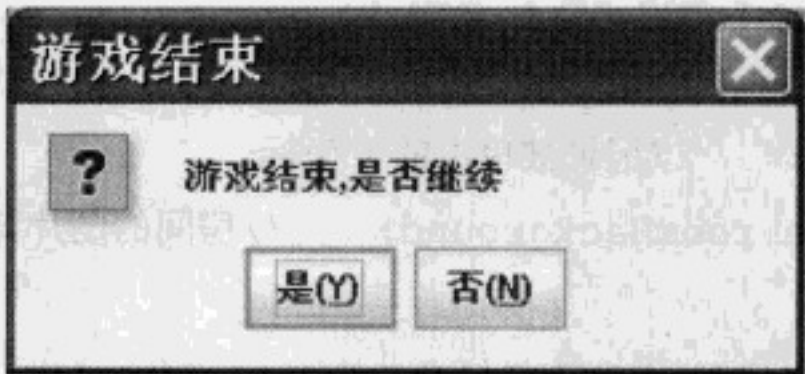


图 11-3 游戏结束对话框

当用户用鼠标连续选中两个相邻的方块时，这里不是按钮，识别是哪个动物方块通过鼠标单击处像素的坐标计算出棋盘坐标来实现。由于有两次选择，因此通过 `(x1, y1)` 记录第一次位置，`(x2, y2)` 记录第二次位置。交换二维数组 `animal` 中 `(x1, y1)` 和 `(x2, y2)` 元素两个方块的动物图案 ID，交换以后调用 `isThreeLinked(y2, x2)` 和 `isThreeLinked(y1, x1)` 方法检测屏幕上是否有符合消去规则的方块，如果有被消去的方块，则 `removeLinked(y2, x2)` 方法修改并记录要绘制方块的动物图案 ID 的 `animal` 数组对应元素的值，不需要绘制记为 `EMPTY`（即 7，因为动物图案 ID 是 0~6 的数字），并调用 `updateAnimal()` 方法从游戏屏幕该列上方重新随机产生新的动物图案 ID，更新动物图案 ID 数组 `animal[8][8]`。最后 `repaint()` 方法刷新显示需要绘制的所有方块图形，从而得到动态游戏效果。

11.3 程序设计的步骤

11.3.1 设计游戏窗口类（GameRoom.java）

游戏窗口类 `GameRoom` 实现游戏的全部功能，继承 `JFrame` 组件实现，由上方 `Panel1` 和中间 `Panel2` 组成。

导入包及相关类:

```
import java.util.Random;
import java.awt.*;
import java.awt.event.MouseMotionAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseAdapter;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;
import java.io.File;
import java.io.IOException;
```

游戏窗口类 GameRoom 定义的成员变量如下:

```
class GameRoom extends JFrame {
    private final int ANIMAL_WIDTH = 48;    //动物图片的宽度
    private final int ANIMAL_HEIGHT = 48;    //动物图片的高度
    private final int ANIMAL_NUM = 7;        //动物数量
    private final int DDP_MATRIX_ROW = 8;
    private final int DDP_MATRIX_COL = 8;
    private final int EMPTY = 7;            //对碰矩阵值空时的标记
    private boolean isThreeLinked;          //标记是否有 3 个以上的连接
    //标记单击次数
    private boolean isDoubleClicked;
    private int xClicked;    //记录被单击的 x 坐标
    private int yClicked;    //记录被单击的 y 坐标
    private BufferedImage roomBackground;    //房间的图片背景
    //随机数
    private Random rand = new Random();
    //动物数组
    private int[][] animal =
        new int[DDP_MATRIX_ROW][DDP_MATRIX_COL];
    private BufferedImage pool;
    private BufferedImage[] animalImage = new BufferedImage[ANIMAL_NUM];
    private RoomPanel roomPanel = new RoomPanel(); //游戏区面板
    JTextField textareal = new JTextField(10);    //显示分数
    JTextField textarea2 = new JTextField(10);    //显示剩余时间
    private int grade = 0;    //得分
    private Timer timer;    //定时器
```

游戏窗口类 GameRoom 的构造方法加载所有动物图片形成 Iocn[7]数组,同时上方 Panel1 加入“分数”文本框和“时间”文本框,中间 Panel2 动态显示动物图案。代码如下:

```
public GameRoom(){
    JLabel label1 = new JLabel("分数");
    JLabel label2 = new JLabel("时间");
    JPanel panell = new JPanel();
    this.setLayout(new BorderLayout());
    panell.setLayout(new FlowLayout());
    panell.add(label1);
    panell.add(textareal);
    textareal.setEditable(false);
    textareal.setText(Integer.toString(grade)); //显示得分
```



```

        textArea2.setText(Integer.toString(100)+"秒");
        panel1.add(label2);
        panel1.add(textArea2);
        this.add(panel1, BorderLayout.NORTH);
        this.add(roomPanel, BorderLayout.CENTER);
    }

```

init()方法不断初始化存储动物图案的数组 animal, 直到产生的动物图案不出现 3 个一相连的情况。图标动物图案 ID 记录在 animal 数组中。添加鼠标移动事件和鼠标单击事件。在鼠标移动事件中判断鼠标位置是否在图形区, 是则将光标设置为“手”状。在鼠标单击事件中实现相邻动物图案的交换。最后添加定时器实现定时功能。代码如下:

```

public void init() {
    do {
        System.out.println("重新初始化");
        initAnimalMatrix();
    } while (globalSearch(1)); //直到不出现 3 个一相连的情况
    print(); //打印出数组 animal 的信息

    //设置房间最佳大小
    roomPanel.setPreferredSize
        (new Dimension(roomBackground.getWidth(),
            roomBackground.getHeight()));

    addMouseMotionListener(new MouseMotionAdapter() {
        @Override
        public void mouseMoved(MouseEvent e) {
            int x = e.getX();
            int y = e.getY();
            setMouseCurHand(x, y);
        }
    });
    addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            System.out.println("x = " + e.getX());
            System.out.println("y = " + e.getY());
            swapAnimal(e.getX(), e.getY());
        }
    });
    setTitle("对对碰");
    pack();
    setResizable(false);
    setVisible(true);
    timer = new Timer(800, new TimeListener());
    timer.start();
    grade = 0;
    initImage();
}

```

setMouseCurHand(int x, int y)方法设置鼠标进入某区域内鼠标显示的图标。代码如下:

```

//设置鼠标进入某区域内鼠标显示的图标
private void setMouseCurHand(int x, int y) {
    //进入动物图案区域

```



```

        if ((x >= 203 && x <= 587)
            && (y >= 65 && y <= 445)) {
            //设置光标为“手”状
            setCursor(new Cursor(Cursor.HAND_CURSOR));
        } else
            setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }

```

由于随机产生动物图案会出现横排或竖排有 3 个以上连续相同的图像,而 `initAnimalMatrix()` 方法可以再次重新产生动物图案。代码如下:

```

//初始化动物数组
private void initAnimalMatrix() {
    for (int i = 0; i < DDP_MATRIX_ROW; i++) {
        for (int j = 0; j < DDP_MATRIX_COL; j++) {
            //随机选取动物图案
            animal[i][j] = rand.nextInt(7);
        }
    }
}

```

`initImage()`方法用来加载背景图片和所有动物图案。代码如下:

```

private void initImage() {
    try {

        //加载背景图片
        roomBackground = ImageIO.read(new File("image/room.jpg"));
        //加载所有动物图案
        animalImage[0] = ImageIO.read(new File("image/monkey.png"));
        animalImage[1] = ImageIO.read(new File("image/fox.png"));
        animalImage[2] = ImageIO.read(new File("image/frog.png"));
        animalImage[3] = ImageIO.read(new File("image/chicken.png"));
        animalImage[4] = ImageIO.read(new File("image/cattle.png"));
        animalImage[5] = ImageIO.read(new File("image/cat.png"));
        animalImage[6] = ImageIO.read(new File("image/panda.png"));
        //加载动物池图片
        pool = ImageIO.read(new File("image/pool.jpg"));

    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}

```

`isThreeLinked(int x, int y)`方法判断(x, y)处附近是否有 3 个以上连续相同的动物图案。

```

private boolean isThreeLinked(int x, int y) { //判断是否有 3 个以上连续相同的方块按钮
    ...//代码同第 10 章
}

```

`removeLinked(int x, int y)`方法将(x, y)处附近有 3 个以上连续相同的方块按钮的图案置为空 (EMPTY), 并根据数量计算用户的得分, 每消去一块加 10 分。代码如下:

```

private void removeLinked(int x, int y) {
    ...//代码同第 10 章
}

```

`globalSearch(int flag)`方法全盘扫描是否有 3 个以上连续相同的方块按钮, 当参数 `flag=1` 时仅扫描; 当参数 `flag=2` 时则将 3 个相连的相同的方块按钮图案置为空。代码如下:


```
private boolean globalSearch(int flag) {
    ...//代码同第 10 章
}
```

downAnimal()方法从游戏屏幕该列底部依次下移上方的方块来填充被消去的方块 (EMPTY)。代码如下:

```
//动物下降
private void downAnimal() {
    ...//代码同第 10 章
}
```

print()方法的代码与第 10 章不同,第 10 章通过 print()方法重新显示按钮图形,而此处仅在控制台打印数组信息。而刷新所有方块按钮图形是通过 repaint 调用 Paint()方法来实现的。代码如下:

```
private void print() {
    for (int i = 0; i < DDP_MATRIX_ROW; i++) {
        for (int j = 0; j < DDP_MATRIX_COL; j++) {
            System.out.print(animal[i][j]);
            if (j == 7) {
                System.out.println();
            }
        }
    }
}
```

对对碰游戏中 swapAnimal(int x, int y)方法交换选中的两个相邻方块,参数(x1,y1)为第一个选定方块按钮数组的坐标,参数(x2,y2)为第二个选定方块按钮数组的坐标。因此 animal [x1, y1]和 animal [x2, y2]的交换即可以实现图案交换。

对对碰游戏中需要检测在交换两个被选中的相邻方块后,横排或竖排是否有 3 个以上方块对象有相同的图像。isThreeLinked(y2, x2)和 isThreeLinked(y1, x1)方法完成检测是否有可以消除的方块对象。如果可以消去方块,则 removeLinked()方法修改并记录要绘制方块的动物图案 ID 的 animal[8][8]数组对应元素的值,不需要绘制记为 EMPTY(即 7,因为动物图案 ID 是 0~6 的数字),并调用 updateAnimal()方法从游戏屏幕该列上方重新随机产生新的动物图案 ID,更新动物图案 ID 数组 animal[8][8]。最后通过 repaint 调用 Paint()方法实现动态游戏效果。

swapAnimal(int x, int y) 与第 10 章的不同之处在于坐标的换算,由于传递参数是像素坐标,因此需要转换成棋盘坐标。

```
//交换动物图案
private void swapAnimal(int x, int y) {
    if ((x >= 203 && x <= 587)
        && (y >= 95 && y <= 479)) {

        //被单击的动物图案坐标的绝对值
        int xAbs;
        int yAbs;
        xAbs = (x - 203) / ANIMAL_WIDTH;
        yAbs = (y - 66) / ANIMAL_HEIGHT;
        System.out.println("xAbs = " + xAbs);
        System.out.println("yAbs = " + yAbs);

        if (!isDoubleClicked) {
            isDoubleClicked = true;
            x1 = xAbs;
            y1 = yAbs;
        }
    }
}
```



```

        System.out.println("被单击的点的 X 坐标 = " + x1);
        System.out.println("被单击的点的 Y 坐标 = " + y1);
        repaint();
    } else {
        int x2 = xAbs;
        int y2 = yAbs;
        isDoubleClicked = false;
        //两点的坐标绝对值等于 1 时视为相邻的两点
        if (1 == Math.abs(x2 - x1) && y2 == y1
            || 1 == Math.abs(y2 - y1) && x2 == x1) {
            //-----交换矩阵中相邻两点的值-----
            int tmp;
            tmp = animal[y2][x2];
            animal[y2][x2] = animal[y1][x1];
            animal[y1][x1] = tmp;
            //-----

            if (isThreeLinked(y2, x2) || isThreeLinked(y1, x1)) {
                System.out.println("消除点");
                if (isThreeLinked(y2, x2))
                    removeLinked(y2, x2);
                if (isThreeLinked(y1, x1))
                    removeLinked(y1, x1);
                downAnimal(); //被消除上方的动物图案下降
                //该列上方重新随机产生新的动物图案, 同时更新动物矩阵
                updateAnimal();
                print();
                repaint();
                //全局扫描判断是否有新的 3 个以上相连的点, 有则删除
                while (globalSearch(1)) {
                    //全局扫描消除 3 个以上相连的点
                    globalSearch(2);
                    //动物图案再次下落
                    downAnimal();
                    //再次更新动物矩阵
                    updateAnimal();
                    repaint();
                }
            }
            else { //没有 3 个以上相连的点, 则交换回来
                System.out.println("交换回来");
                tmp = animal[y1][x1];
                animal[y1][x1] = animal[y2][x2];
                animal[y2][x2] = tmp;
                print();
                //选中第二次单击的动物图案
                isDoubleClicked = true;
                x1 = xAbs;
                y1 = yAbs;
                repaint();
            }
        }
    }
}

```



```

    }
}

```

11.3.2 设计内部游戏面板类

游戏面板类 RoomPanel 的设计是图形化实现对对碰游戏的关键, 在面板中重载 Paint 事件将背景图片、动物池和所有动物绘制到面板中。代码如下:

```

class RoomPanel extends JPanel {
    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;
        //绘制背景图片
        g2d.drawImage(roomBackground, 0, 0
            , roomBackground.getWidth()
            , roomBackground.getHeight(), null);

        //绘制动物池
        g2d.drawImage(pool, 200, 0, 384, 384, null);
        //在动物池中绘制所有动物
        for (int i = 0; i < DDP_MATRIX_ROW; i++) {
            for (int j = 0; j < DDP_MATRIX_COL; j++) {
                g2d.drawImage(animalImage[animal[i][j]]
                    , 200 + j * 48, i * 48
                    , ANIMAL_WIDTH, ANIMAL_HEIGHT, null);
            }
        }
        g2d.drawRect(200 + x1 * 48, y1 * 48, 48, 48);
    }
}

```

11.3.3 设计内部定时器类

内部定时器类 TimeListener 修改剩余时间, 并判断是否达到最大值 100, 如果达到则定时器结束, 出现对话框提示是否继续游戏, 如果用户单击“是”按钮则可以开始新游戏, 单击“否”按钮则退出程序。代码如下:

```

class TimeListener implements ActionListener {
    int times = 0;
    public void actionPerformed(ActionEvent e) {
        times++;
        if (times > 100) {
            timer.stop();
            //定时器结束
            int answer=JOptionPane.showConfirmDialog(null,
                "游戏结束,是否继续", "游戏结束",
                JOptionPane.YES_NO_OPTION);
            if(answer== JOptionPane.YES_OPTION)
                init();
            else
                System.exit(0);
        }else
            textarea2.setText(Integer.toString(100-times)+"秒");
    }
}

```


第 12 章

俄罗斯方块游戏

12.1 俄罗斯方块游戏介绍

俄罗斯方块是一款风靡全球的电视游戏机和掌上游戏机游戏，它曾经引起的轰动与造就的经济价值可以说是游戏史上的一件大事。这款游戏最初是由前苏联的游戏制作人 Alex Pajitnov 制作的，它看似简单却变化无穷。游戏过程中仅需要玩家将不断下落的各种形状的方块进行移动和翻转，如果某一行被方块充满，就将该行消除；而当窗口中无法再容纳下落的方块时，则宣告游戏的结束。

通过上述介绍可见俄罗斯方块游戏的需求如下：

- (1) 由移动的方块和不能移动的固定方块组成；
- (2) 一行排满则消除该行；
- (3) 能产生多种方块；
- (4) 玩家可以看到游戏的积分和下一方块的形状；
- (5) 下一方块可以逆时针旋转。

俄罗斯方块游戏的界面如图 12-1 所示。

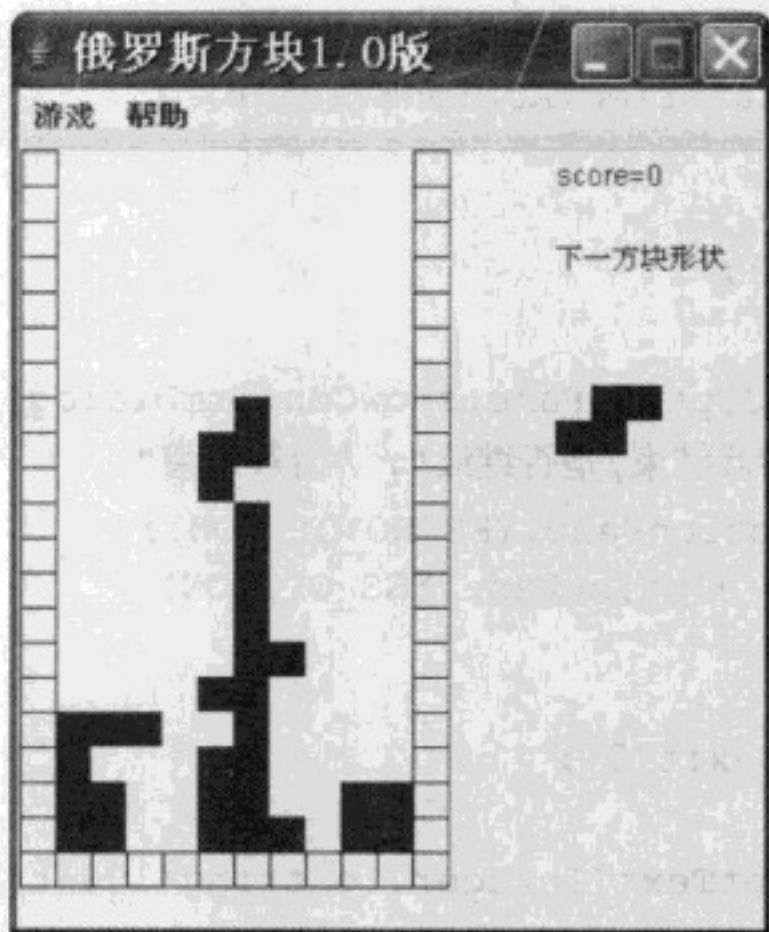


图 12-1 俄罗斯方块游戏的界面

12.2 程序设计的思路

12.2.1 俄罗斯方块形状的设计

游戏中下落的方块有着各种不同的形状，要在游戏中绘制不同形状的方块，就需要使用合理的数据表示方式。目前常见的俄罗斯方块拥有 7 种基本的形状以及它们旋转以后的变形体，具体的形状如图 12-2 所示。

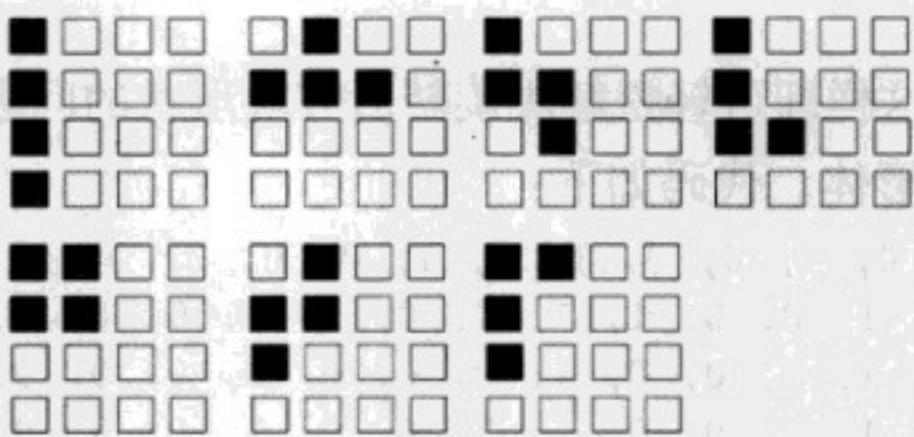


图 12-2 俄罗斯方块的形状

每种形状都是由不同的黑色小方格组成，在屏幕上只需要显示必要的黑色小方格就可以表现出各种形状，它们的数据逻辑可以使用一个 4×4 的 2 维数组表示，数组的存储值为 0 或者 1，如果值为 1 则表示需要显示一个黑色方块，为 0 则表示不显示黑色方块。

例如，⊥字形方块的数组存储格式如下：

```
int[][] shapes = new int[][]{
    {0, 1, 0, 0 },
    { 1, 1, 1, 0 },
    { 0, 0, 0, 0 },
    { 0, 0, 0, 0 }},
```

每种形状逆时针转动就会形成一个新的形状，为了使程序处理起来简单，可以把这些基本形状的变形体都使用二维数组定义好，这样就不需要编写每个方块的旋转函数了。

注意本游戏将每种形状采用一维数组存储也是可行的。例如，⊥字形方块最初为 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0；由于是 4×4 的方阵，因此可以分解如下：

```
0, 1, 0, 0,
1, 1, 1, 0,
0, 0, 0, 0,
0, 0, 0, 0
```

可见这是“⊥”字形。假如旋转 1 次，则为 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0；由于是 4×4 的方阵，因此可以分解如下：

```
0, 1, 0, 0,
1, 1, 0, 0,
0, 1, 0, 0,
0, 0, 0, 0
```

可见这是“└”字形。假如旋转 2 次，则为 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0；由于是 4×4 的方阵，因此可以分解如下：


```
1, 1, 1, 0,  
0, 1, 0, 0,  
0, 0, 0, 0,  
0, 0, 0, 0
```

可见这是“T”字形。假如旋转 3 次，则为 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0；由于是 4×4 的方阵，因此可以分解如下：

```
0, 1, 0, 0,  
0, 1, 1, 0,  
0, 1, 0, 0,  
0, 0, 0, 0
```

可见这是“┣”字形。这样即可轻松解决旋转后方块形状的问题。因此可以定义一个二维数组来存储这种形状及所有变形体。代码如下：

```
int[][] shapes = { { 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
                   { 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 },  
                   { 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
                   { 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0 } }
```

由于二维数组 shapes 仅能保存一种形状及其变形，因此可以用三维数组存储 7 种形状及其变形。

12.2.2 俄罗斯方块游戏的面板屏幕

游戏的面板由一定的行数和列数的单元格组成，游戏的面板屏幕可以看成由图 12-3 所示的屏幕网格组成。

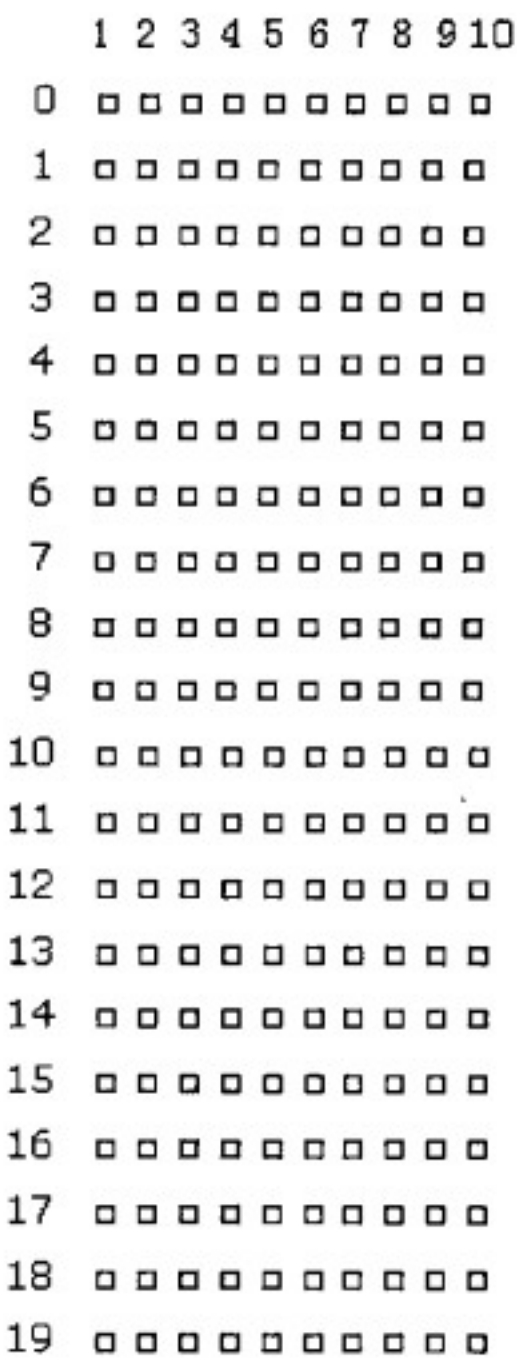


图 12-3 屏幕网格

屏幕由 20 行 10 列的网格组成，为了存储游戏画面中的已固定方块，这里采用二维数组 map，

当相应的数组元素值为 1 则绘制一个黑色小方块。一个俄罗斯方块形状在面板中的显示只需要把面板中相应的单元格绘制为黑色方块即可，如图 12-4（a）所示，面板中显示一个“L”型方块，只需要按照“L”型方块一维数组的定义，将它的一维数组的数据在 Paint()方法中绘制到面板即可。

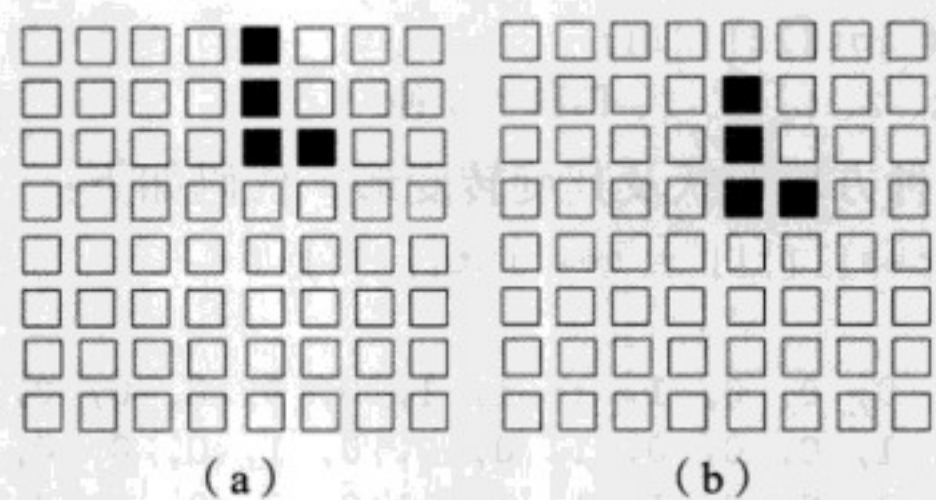


图 12-4 “L”型方块下落前和下落后的示意图

方块的下落的基本处理方式就是当前方块下移一行的位置，然后根据当前方块的数组数据和存储固定方块的面板二维数组 map，重新绘制一次屏幕即可，如图 12-4（b）所示。因此需要使用一个坐标来记录当前方块形状所在的行号 y 和列号 x。

12.2.3 俄罗斯方块游戏的运行流程

俄罗斯方块游戏就是在一个线程或者定时器控制下产生重绘事件，用户利用键盘输入改变游戏状态。在每隔一定的时间就重画当前下落的方块和 map 存储的固定方块，从而看到动态游戏效果。俄罗斯方块下落的过程中可能遇到多种情况，比如是否需要消行，是否需要终止下落的方块并且产生新形状的方块，等等。具体的判断流程如下。

首先判断方块是否可以继续下落，可以下落则 y++即可。如果方块不能继续下落，则将当前形状的方块添加到面板二维数组 map 中，界面产生新形状的方块且判断是否需要消行。最后请求重新绘制屏幕。

12.3 俄罗斯方块设计的步骤

12.3.1 设计游戏界面类（Tetrisblok.java）

在项目中创建一个继承 JPanel 的俄罗斯方块类 Tetrisblok，用于实现游戏界面，完成方块的自动下落、满行消除、计算得分以及方块的旋转移动等功能。

导入包及相关类：

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.Timer;
```

俄罗斯方块类 Tetrisblok 继承 JPanel，同时实现键盘事件接口 KeyListener。代码如下：

```
class Tetrisblok extends JPanel implements KeyListener {
    private int blockType; //blockType 代表方块类型
    private int turnState; //turnState 代表方块旋转的状态
```



```

private int score = 0;
private int nextblockType=-1,nextturnState=-1; //下一方块的状态
private int x,y; //当前方块位置
private Timer timer; //定时器
//游戏地图，存储已经放下的方块（1）及围墙（2）。空白处为（0）
int[][] map = new int[12][21];
//方块的形状，有倒Z、Z、L、J、I、田和T 7种

```

三维数组 shapes 存储 7 种方块形状及其旋转变形，代码如下：

```

private final int shapes[][][] = new int[][][] {
    //长条 I 形
    {
        { 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0 },
        { 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0 } },
    //倒 Z 字形
    {
        { 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 },
        { 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 } },
    //Z 字形
    {
        { 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 } },
    //J 字形
    {
        { 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0 },
        { 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 } },
    //田字形
    {
        { 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } },
    //L 字形
    {
        { 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 } },
    //I 字形
    {
        { 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 },
        { 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0 } } };

```

三维数组 shapes 存储 Z、倒 Z、L、J、I、田和 I 7 种方块的形状，每个二维数组存储一种方块旋转变形的形状。

生成新方块的新方法 newblock() 方法中判断是否已有下一方块，如果没有则同时产生当前方块和下一方块的形状代号和旋转状态。如果已有下一方块，则将已有的下一方块作为当前方块，再随机产生下一方块的形状代号 nextblockType（0~5 之间）和旋转状态 nextturnState（0~3 之间）。例

如旋转状态为 1 则是旋转 1 次后的方块, 旋转状态为 2 则是旋转 2 次后的方块。代码如下:

```
public void newblock() {
    //没有下一方块
    if(nextblockType == -1 && nextturnState == -1){
        blockType = (int) (Math.random() * 1000) % 7;
        turnState = (int) (Math.random() * 1000) % 4;
        nextblockType=(int) (Math.random() * 1000) % 7;
        nextturnState=(int) (Math.random() * 1000) % 4;
    }
    else{//已有下一方块
        blockType = nextblockType;
        turnState = nextturnState;
        nextblockType=(int) (Math.random() * 1000) % 7;
        nextturnState=(int) (Math.random() * 1000) % 4;
    }
    x = 4;        y = 0;//屏幕上方中央
    if (gameover(x, y) == 1) { //游戏结束
        newmap();
        drawwall();
        score = 0;
        JOptionPane.showMessageDialog(null, "GAME OVER");
    }
}
```

drawwall()方法完成在 map 数组中保存围墙的信息。围墙在 0 列和 11 列, 以及底部第 20 行。游戏区域在 0~19 行, 1~10 列范围中。代码如下:

```
public void drawwall() {
    int i, j;
    for (i = 0; i < 12; i++) { //底部第 20 行
        map[i][20] = 2;
    }
    for (j = 0; j < 21; j++) { //在 0 列和 11 列
        map[11][j] = 2;
        map[0][j] = 2;
    }
}
```

newmap()方法用来初始化地图, 将游戏区游戏清空 (置零)。代码如下:

```
//初始化地图
public void newmap() {
    int i, j;
    for (i = 0; i < 12; i++) {
        for (j = 0; j < 21; j++) {
            map[i][j] = 0;
        }
    }
}
```

Tetrisblok()构造方法产生一个新的下落方块, 同时启动定时器。定时器每隔 0.5s 触发一次。在定时器触发事件中完成屏幕重画, 同时判断当前方块是否可以下落。如果不可以下落则固定当前方块, 并消去可能的满行, 同时产生新的当前方块。代码如下:

```
Tetrisblok() {
    newblock();
}
```



```

newmap();
drawwall();
timer = new Timer(500, new TimerListener()); // 0.5s
timer.start();
}
// 定时器监听事件
class TimerListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (blow(x, y + 1, blockType, turnState) == 1) { // 可以下落
            y = y + 1; // 当前方块下移
        }
        if (blow(x, y + 1, blockType, turnState) == 0) { // 不可以下落
            add(x, y, blockType, turnState); // 固定当前方块
            delline(); // 消去满行
            newblock(); // 产生新的方块
        }
        repaint(); // 屏幕重画
    }
}

```

以下是菜单事件的代码。通过定时器的启动、停止来达到游戏的暂停和继续。代码如下：

```

public void newGame() // 新游戏
{
    newblock();
    newmap();
    drawwall();
}
public void pauseGame() // 暂停游戏
{
    timer.stop();
}
public void continueGame() // 继续游戏
{
    timer.start();
}

```

turn()是旋转当前方块的方法。仅将旋转次数 turnState 加 1 后, blow(x, y, blockType, turnState) 方法判断是否可以旋转, 如果不可以则将旋转次数恢复为原来的值。代码如下:

```

// 旋转当前方块的方法
public void turn() {
    int tempturnState = turnState;
    turnState = (turnState + 1) % 4;
    if (blow(x, y, blockType, turnState) == 1) { // 可以旋转
    }
    if (blow(x, y, blockType, turnState) == 0) { // 不可以旋转
        turnState = tempturnState; // 将旋转次数恢复为原来的值
    }
    repaint();
}

```

left()是左移当前方块的方法。blow(x-1, y, blockType, turnState)方法判断是否可以左移, 如果可以则将当前方块的 x 减 1 后重画。方法 right()右移的原理相似。代码如下:

```

// 左移的方法

```



```

public void left() {
    if (blow(x - 1, y, blockType, turnState) == 1) {
        x = x - 1;
    }
    repaint();
}

```

//右移的方法

```

public void right() {
    if (blow(x + 1, y, blockType, turnState) == 1) {
        x = x + 1;
    }
    repaint();
}

```

down()是当前方块下落的方法。blow(x, y+1, blockType, turnState)方法判断是否可以下落，可以下落则将当前方块的 y 加 1 后重画。如果不可以下落则固定当前方块，并消去可能的满行，同时产生新的当前方块。代码如下：

//下落的方法

```

public void down() {
    if (blow(x, y + 1, blockType, turnState) == 1) { //可以下落
        y = y + 1;
    }
    if (blow(x, y + 1, blockType, turnState) == 0) { //不可以下落
        add(x, y, blockType, turnState);
        newblock();
        delline();
    }
    repaint();
}

```

blow(int x, int y, int blockType, int turnState)方法判断当前方块的位置 (x, y) 是否合法，如果 (shapes[blockType][turnState][a * 4 + b] == 1) && (map[x + b + 1][y + a] == 1) 即当前方块和地图中固定方块重叠或者 (shapes[blockType][turnState][a * 4 + b] == 1) && (map[x + b + 1][y + a] == 2) 表示碰到围墙则返回 0，表示不合法；否则返回 1 表示合法。代码如下：

//判断移动或旋转后当前方块的位置是否合法

```

public int blow(int x, int y, int blockType, int turnState) {
    for (int a = 0; a < 4; a++) {
        for (int b = 0; b < 4; b++) {
            if (((shapes[blockType][turnState][a * 4 + b] == 1) && (map[x + b + 1][y + a] == 1))
                || ((shapes[blockType][turnState][a * 4 + b] == 1) && (map[x + b + 1][y + a] == 2))) {
                return 0;
            }
        }
    }
    return 1;
}

```

delline()是消去满行的方法。如果第 d 行满行，则上方方块下移。代码如下：

//消去满行的方法

```

public void delline() {
    int c = 0;
    for (int b = 0; b < 21; b++) {
        for (int a = 0; a < 12; a++) {
            if (map[a][b] == 1) {

```



```

        c = c + 1;
        if (c == 10) { //该行满行
            score += 10;
            for (int d = b; d > 0; d--) {
                for (int e = 0; e < 12; e++) { //则上方方块下移
                    map[e][d] = map[e][d - 1];
                }
            }
        }
    }
    c = 0;
}
}

```

gameover(int x, int y)是判断游戏结束的方法。代码如下:

```

public int gameover(int x, int y) {
    if (blow(x, y, blockType, turnState) == 0) {
        return 1;
    }
    return 0;
}

```

add(int x, int y, int blockType, int turnState)方法将当前方块添加到游戏地图 **map** 中。代码如下:

```

public void add(int x, int y, int blockType, int turnState) {
    int j = 0;
    for (int a = 0; a < 4; a++) {
        for (int b = 0; b < 4; b++) {
            if (shapes[blockType][turnState][j] == 1) {
                map[x + b + 1][y + a] = shapes[blockType][turnState][j];
            }
            j++;
        }
    }
}

```

paint(Graphics g)是屏幕重画的方法。代码如下:

```

public void paint(Graphics g) {
    super.paint(g); //调用父类的 paint() 方法, 实现初始化清屏
    int i, j;
    //画当前方块
    for (j = 0; j < 16; j++) {
        if (shapes[blockType][turnState][j] == 1) {
            g.fillRect((j % 4 + x + 1) * 15, (j / 4 + y) * 15, 15, 15);
        }
    }
    //画已经固定的方块和围墙
    for (j = 0; j < 21; j++) {
        for (i = 0; i < 12; i++) {
            if (map[i][j] == 1) { //画已经固定的方块
                g.fillRect(i * 15, j * 15, 15, 15);
            }
            if (map[i][j] == 2) { //画围墙
                g.drawRect(i * 15, j * 15, 15, 15);
            }
        }
    }
}

```



```

    }
}
g.drawString("score=" + score, 225, 15);
g.drawString("下一方块形状", 225, 50);
//在窗口右侧区域绘制下一方块
for (j = 0; j < 16; j++) {
    if (shapes[nextblockType][nextturnState][j] == 1) {
        g.fillRect(225+(j % 4) * 15, (j / 4) * 15+100, 15, 15);
    }
}
}
}

```

以下是键盘监听事件，完成左右键左右移动方块，向上键旋转方块，向下键向下移动方块的功能。代码如下：

```

//键盘监听事件
public void keyPressed(KeyEvent e) {
    switch (e.getKeyCode()) {
        case KeyEvent.VK_DOWN:
            down();
            break;
        case KeyEvent.VK_UP:
            turn();
            break;
        case KeyEvent.VK_RIGHT:
            right();
            break;
        case KeyEvent.VK_LEFT:
            left();
            break;
    }
}
//无用
public void keyReleased(KeyEvent e) {
}
//无用
public void keyTyped(KeyEvent e) {
}
}

```

12.3.2 设计游戏窗口类 (TetrisFrame.java)

在项目中创建一个继承 JFrame 的 TetrisFrame 类，用于显示自定义游戏面板 Tetrisblok 界面，同时加入菜单及菜单事件监听。代码如下：

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
public class TetrisFrame extends JFrame implements ActionListener{
    static JMenu game = new JMenu("游戏");
    JMenuItem newgame = game.add("新游戏");
    JMenuItem pause = game.add("暂停");
    JMenuItem goon = game.add("继续");
    JMenuItem exit = game.add("退出");
}

```



```

static JMenu help = new JMenu("帮助");
JMenuItem about = help.add("关于");
Tetrisblok a = new Tetrisblok();
public TetrisFrame () {
    addKeyListener(a);
    this.add(a);
    newgame.addActionListener(this); // “新游戏”菜单项
    pause.addActionListener(this); // “暂停”菜单项
    goon.addActionListener(this); // “继续”菜单项
    about.addActionListener(this); // “关于”菜单项
    exit.addActionListener(this); // “退出”菜单项
}
public void actionPerformed(ActionEvent e) {
    if(e.getSource()==newgame) // “新游戏”菜单项
    {
        a.newGame();
    }else if(e.getSource()==pause) // “暂停”菜单项
    {
        a.pauseGame();
    }else if(e.getSource()==goon) // “继续”菜单项
    {
        a.continueGame();
    }else if(e.getSource()==about) // “关于”菜单项
    {
        DisplayToast("左右键移动, 向上键旋转");
    } else if(e.getSource()==exit) // “退出”菜单项
    {
        System.exit(0);
    }
}
public void DisplayToast(String str) {
    JOptionPane.showMessageDialog(null, str, "提示",
        JOptionPane.ERROR_MESSAGE);
}
public static void main(String[] args) {
    TetrisFrame frame = new TetrisFrame ();
    JMenuBar menu = new JMenuBar();
    frame.setJMenuBar(menu);
    menu.add(game);
    menu.add(help);
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // “结束”按钮可用
    frame.setSize(320, 375);
    frame.setTitle("俄罗斯方块 1.0 版");
    //frame.setUndecorated(true);
    frame.setVisible(true);
    frame.setResizable(false);
}
}

```


第 13 章

开心农场

13.1 开心农场游戏介绍

开心农场是一款以种植为主的社交游戏。用户可以扮演一个农场的农场主，在自己农场里开垦土地、种植各种蔬菜和水果。本章开发了一个开心农场游戏，游戏运行界面如图 13-1 所示，单击“播种”按钮，可以播种种子；单击“生长”按钮，可以让农作物处于生长阶段；单击“开花”按钮，可以让农作物处于开花阶段；单击“结果”按钮，可以让农作物结果；单击“收获”按钮，可以收获果实到仓库中。



图 13-1 开心农场游戏运行界面

13.2 程序设计的思路

13.2.1 游戏素材

开心农场游戏中用到的播种、生长、开花、结果等农作物状态，分别使用图 13-2 所示的图片来表示。

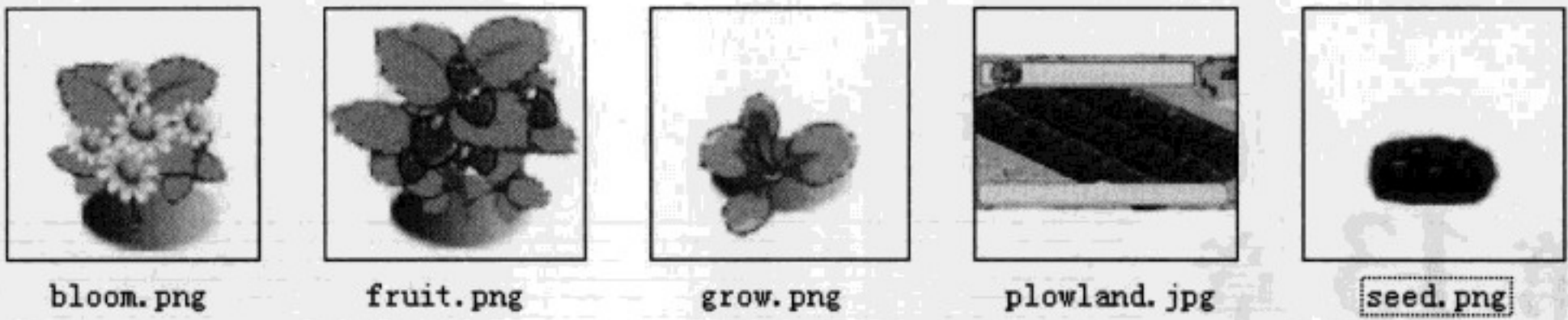


图 13-2 农作物状态相关图片素材

开心农场游戏中用到的播种、生长、开花、结果和收获的 5 个按钮的图片，分别使用图 13-3 所示的图片来表示。每个按钮均有两种图片，分别表示正常和鼠标经过的效果。



图 13-3 5 个按钮的相关图片素材

13.2.2 设计思路

使用一个带背景（plowland.jpg）的面板作为土地，其上显示播种、生长、开花、结果和收获这 5 个按钮和 1 个继承 JLabel 的表示农作物的 Crop 对象。在 5 个按钮的单击事件中改变 Crop 对象的图片就可以实现农作物各种状态的改变。

13.3 关键技术

13.3.1 实现图形按钮

本章游戏的按钮是圆形按钮，Java 能实现这种图形化按钮，仅使用 JButton 的相关方法就可以实现。主要代码如下：

```
JButton button= new JButton( );
ImageIcon exitedImageIcon =new ImageIcon("res / exited.png");
ImageIcon enteredImageIcon =new ImageIcon("res / roll.png");
ImageIcon pressedImageIcon =new ImageIcon("res / down.png");
button.setIcon(exitedImageIcon); //设置鼠标不在按钮上时的图标
button.setRolloverIcon(enteredImageIcon); //设置鼠标移到按钮上时的图标
button.setPressedIcon(pressedImageIcon); //设置鼠标单击时的图标
button.setContentAreaFilled(false); //是否显示外围矩形区域，设置与否
```



```
button.setFocusable(false);           //去掉按钮的聚焦框
button.setBorderPainted(false);       //去掉边框
```

13.4 开心农场设计的步骤

13.4.1 设计农作物类 (Crop.java)

农作物类 Crop 实现农作物各种状态的改变,由继承 JLabel 组件且改变 JLabel 组件的 Icon 组件实现。代码如下:

```
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JLabel;

public class Crop extends JLabel {
    Icon icon = null;           //图标对象
    public Crop() {
        super();
    }
    public void setIcon(String picture){
        icon=new ImageIcon(picture);    //获取图片
        setIcon(icon);                 //设置组件要显示的图标,用于显示农作物的状态
    }
}
```

13.4.2 设计背景的面板 (BackgroundPanel.java)

导入包及相关类:

```
import java.awt.Graphics;
import java.awt.Image;
import javax.swing.JPanel;
/**
 * 带背景的面板组件
 */
public class BackgroundPanel extends JPanel {
    private Image image;       //背景图片
    /**
     * 构造方法
     */
    public BackgroundPanel() {
        super();
        setOpaque(false);
        setLayout(null);
    }
    /**
     * 设置图片的方法
     */
    public void setImage(Image image) {
        this.image = image;
    }
}
```



```
protected void paintComponent(Graphics g) { //重写绘制组件的外观
    if (image != null) {
        int width = getWidth(); //获取组件大小
        int height = getHeight();
        g.drawImage(image,0,0,width,height, this); //绘制的图片与组件大小相同
    }
    super.paintComponent(g); //执行超类方法
}
```

13.4.3 设计农场类 (Farml.java)

编写一个农场类，名称为 Farm，在该类中编写 seed()方法，用于实现播种操作。在该方法中，如果农作物的状态为未播种，则进行播种，将农作物显示为播种状态，并修改成员变量 state 的值为 1 (表示已播种)；否则，设置提示信息为不能播种。代码如下：

```
public class Farm {
    public int state = 0; //农作物的状态：0 表示未播种；1 表示播种；
                        //2 表示生长；3 表示开花；4 表示结果

    /**
     * 播种
     */
    public String seed(Crop crop, String picture) {
        String returnValue = ""; //返回值变量
        if (state == 0) { //判断农作物的状态是否为未播种
            crop.setIcon(picture); //播种农作物
            state = 1; //设置状态属性为播种
        } else {
            returnValue = getMessage() + "，不能播种！"; //设置提示信息
        }
        return returnValue;
    }
}
```

在农场类 Farm 中编写 grow()方法，用于实现生长操作。在该方法中，如果农作物的状态为已播种，则置其处于生长阶段，并修改成员变量 state 的值为 2 (表示已生长)；否则，设置提示信息为不能生长。代码如下：

```
/**
 * 生长
 */
public String grow(Crop crop, String picture) {
    String returnValue = ""; //返回值变量
    if (state == 1) { //判断农作物的状态是否为播种
        crop.setIcon(picture); //设置农作物为生长状态
        state = 2; //设置状态属性为生长
    } else {
        returnValue = getMessage() + "，不能生长！"; //设置提示信息
    }
    return returnValue;
}
```

在农场类 Farm 中编写 bloom()方法，用于实现开花操作。实现过程与前面方法相似。代码如下：


```

/**
 * 开花
 */
public String bloom(Crop crop, String picture) {
    String returnValue = ""; //返回值变量
    if (state == 2) { //判断农作物的状态是否为生长
        crop.setIcon(picture); //设置农作物为开花状态
        state = 3; //设置状态属性为开花
    } else {
        returnValue = getMessage() + ", 不能开花! "; //设置提示信息
    }
    return returnValue;
}

```

在农场类 Farm 中编写 fruit()方法,用于实现结果操作。实现过程与前面方法相似。代码如下:

```

/**
 * 结果
 */
public String fruit(Crop crop, String picture) {
    String returnValue = ""; //返回值变量
    if (state == 3) { //判断农作物的状态是否为开花
        crop.setIcon(picture); //设置农作物为结果状态
        state = 4; //设置状态属性为结果
    } else {
        returnValue = getMessage() + ", 不能结果! "; //设置提示信息
    }
    return returnValue;
}

```

在农场类 Farm 中编写 harvest ()方法,用于实现收获操作。在该方法中,如果农作物的状态为已结果,则收获果实,让农作物处于未播种状态,并修改成员变量 state 的值为 0(表示未播种);否则,设置提示信息为不能收获。代码如下:

```

/**
 * 收获
 */
public String harvest(Crop crop, String picture) {
    String returnValue = ""; //返回值变量
    if (state == 4) { //判断农作物的状态是否为结果
        crop.setIcon(picture); //设置农作物为未播种状态
        state = 0; //设置状态属性为未播种
    } else {
        returnValue = getMessage() + ", 不能收获! "; //设置提示信息
    }
    return returnValue;
}

```

编写 getMessage()方法,用于根据农作物的状态属性,确定对应的提示信息。代码如下:

```

/**
 * 根据农作物的状态属性,确定对应的提示信息
 */
public String getMessage() {

```



```

        String message = "";
        switch (state) {
            case 0:
                message = "农作物还没有播种";
                break;
            case 1:
                message = "农作物刚刚播种";
                break;
            case 2:
                message = "农作物正在生长";
                break;
            case 3:
                message = "农作物正处于开花期";
                break;
            case 4:
                message = "农作物已经结果";
                break;
        }
        return message;
    }
}

```

13.4.4 设计窗体类 (MainFrame.java)

编写一个继承 JFrame 类的 MainFrame 窗体类，用于完成农作物的播种、生长、开花、结果和收获等操作。

导入包及相关类：

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

```

MainFrame 窗体类实例化一个农场 Farm 类的对象 farm，实例化一个农作物 Crop 类的对象 crop，并且定义果实数量 fruitNumber。代码如下：

```

public class MainFrame extends JFrame implements MouseListener{
    int fruitNumber = 0;        //果实数量
    Farm farm = new Farm();     //实例化 Farm 类的对象
    final Crop crop = new Crop();
    public static void main(String args[]) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    MainFrame frame = new MainFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}

```

MainFrame 窗体类的构造方法创建 5 个按钮，并且以图 13-2 所示的图片形式显示。“播种”按钮单击事件中调用农场类的 seed()方法；“生长”按钮单击事件中调用农场类的 grow()方法；“开花”按钮单击事件中调用农场类的 bloom()方法；“结果”按钮单击事件中调用农场类的 fruit()方

法以及“收获”按钮单击事件中调用农场类的 harvest()方法。如果有返回信息则说明未达到该状态,则提示错误信息。代码如下:

```

/**
 * Create the frame
 */
public MainFrame() {
    super();
    setTitle("打造自己的开心农场");
    setBounds(500, 200, 466, 276);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    addMouseListener(this);
    final BackgroundPanel backgroundPanel = new BackgroundPanel();
    Image bk=Toolkit.getDefaultToolkit().getImage("images/plowland.jpg");
    backgroundPanel.setImage(bk);
    backgroundPanel.setBounds(0, 0, 458, 242);
    getContentPane().add(backgroundPanel);

    crop.setBounds(180, 55, 106, 96);
    backgroundPanel.add(crop);

    final JButton button = new JButton();
    //设置当光标移动到按钮上方时显示的图片
    button.setRolloverIcon(new ImageIcon("images/播种1.png"));
    //设置为不绘制按钮的边框,当设为 False 时表示不绘制,默认为绘制
    button.setBorderPainted(false);
    //设置按钮的背景为透明,当设为 False 时表示不绘制,默认为绘制
    button.setContentAreaFilled(false);
    button.setIcon(new ImageIcon("images/播种.png"));
    button.addActionListener(new ActionListener() {
        public void actionPerformed(final ActionEvent e) {
            String message = farm.seed(crop, "images/seed.png");//播种
            if (!message.equals("")) { //当提示信息不为空时
                JOptionPane.showMessageDialog(null, message); //弹出提示对话框
            }
        }
    });
    button.setBounds(29, 185, 56, 56);
    backgroundPanel.add(button);

    final JButton button_1 = new JButton();
    button_1.setContentAreaFilled(false);
    button_1.setBorderPainted(false);
    button_1.setRolloverIcon(new ImageIcon(
        "images/生长1.png"));
    button_1.setIcon(new ImageIcon(
        "images/生长.png"));
    button_1.addActionListener(new ActionListener() {
        public void actionPerformed(final ActionEvent e) {
            String message = farm.grow(crop, "images/grow.png");//生长
            if (!message.equals("")) {
                JOptionPane.showMessageDialog(null, message); //弹出提示对话框
            }
        }
    });
}

```



```

    }
    });
    backgroundPanel.add(button_1);
    button_1.setBounds(114, 185, 56, 56);

    final JButton button_2 = new JButton();
    button_2.setBorderPainted(false);
    button_2.setContentAreaFilled(false);
    button_2.setRolloverIcon(new ImageIcon("images/开花1.png"));
    button_2.setIcon(new ImageIcon("images/开花.png"));
    button_2.addActionListener(new ActionListener() {
        public void actionPerformed(final ActionEvent e) {
            String message = farm.bloom(crop, "images/bloom.png"); //开花
            if (!message.equals("")) {
                JOptionPane.showMessageDialog(null, message); //弹出提示对话框
            }
        }
    });
    button_2.setBounds(199, 185, 56, 56);
    backgroundPanel.add(button_2);

    final JButton button_3 = new JButton();
    button_3.setBorderPainted(false);
    button_3.setContentAreaFilled(false);
    button_3.setRolloverIcon(new ImageIcon("images/结果1.png"));
    button_3.setIcon(new ImageIcon("images/结果.png"));
    button_3.addActionListener(new ActionListener() {
        public void actionPerformed(final ActionEvent e) {
            String message = farm.fruit(crop, "images/fruit.png"); //结果
            if (!message.equals("")) {
                JOptionPane.showMessageDialog(null, message); //弹出提示对话框
            }
        }
    });
    button_3.setBounds(284, 185, 56, 56);
    backgroundPanel.add(button_3);

    final JLabel storage = new JLabel();
    storage.setHorizontalAlignment(SwingConstants.CENTER);
    storage.setText("您的仓库没有任何果实，快快播种吧！");
    storage.setBounds(80, 15, 253, 28);
    backgroundPanel.add(storage);

    final JButton button_4 = new JButton();
    button_4.setRolloverIcon(new ImageIcon("images/收获1.png"));
    button_4.setIcon(new ImageIcon("images/收获.png"));
    button_4.setContentAreaFilled(false);
    button_4.setBorderPainted(false);
    button_4.addActionListener(new ActionListener() {
        public void actionPerformed(final ActionEvent e) {
            String message = farm.harvest(crop, ""); //收获

```

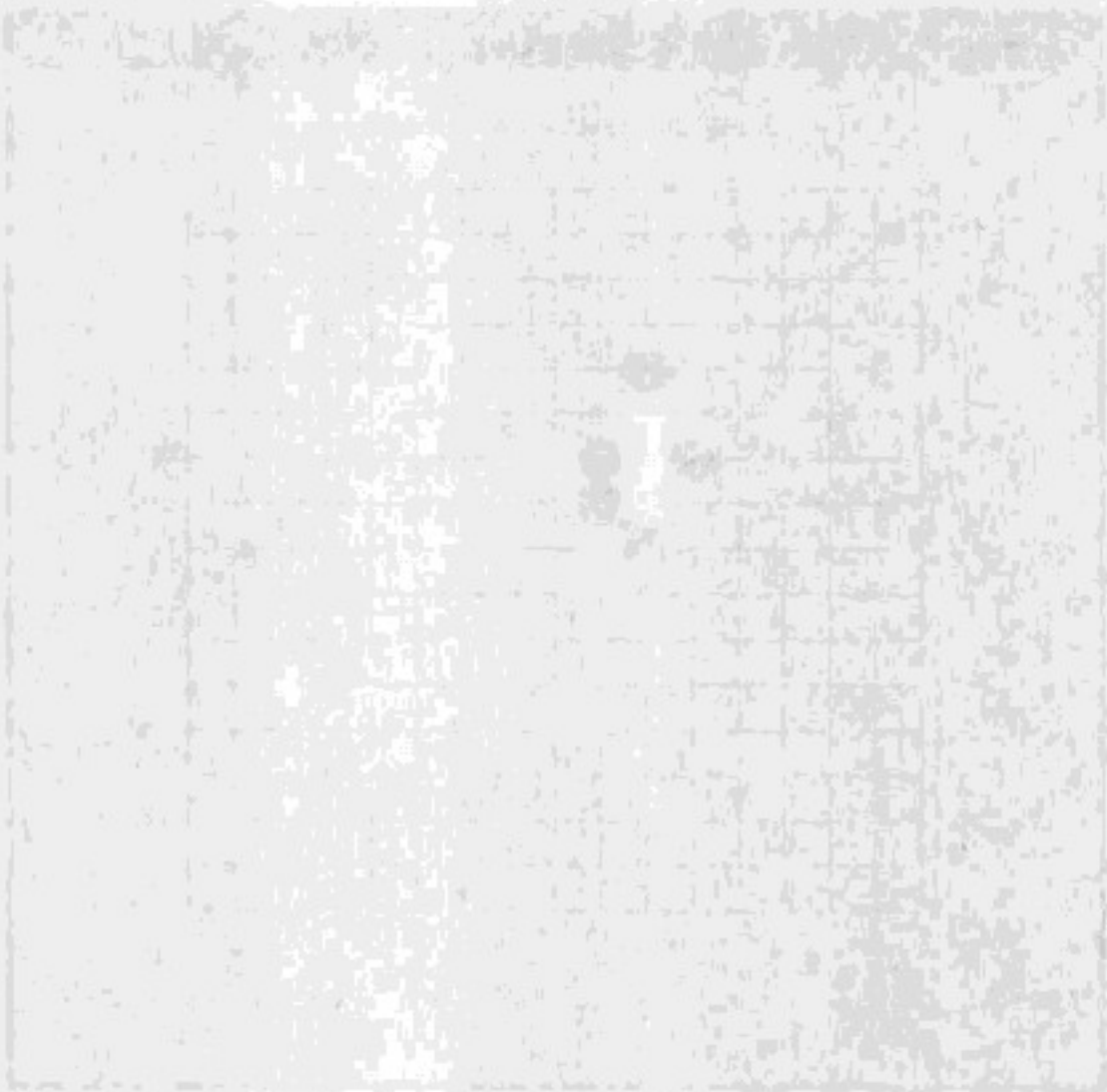


```
        if (!message.equals("")) {
            JOptionPane.showMessageDialog(null, message);    //弹出提示对话框
        } else {
            fruitNumber++;
            storage.setText("您的仓库现在有" + fruitNumber + "个果实!");
        }
    }
});
button_4.setBounds(369, 185, 56, 56);
backgroundPanel.add(button_4);
}
```

使用鼠标的单击事件可以改变种植的地块，此处实现比较简单，仅将农作物 crop 的位置移到单击的某块地附近就可以了。代码如下：

```
//单击选择某块地
public void mouseClicked(MouseEvent e) {
    int x = e.getX();    //获取 x 坐标
    int y = e.getY();    //获取 y 坐标
    //选择某块地
    crop.setBounds(x-55, y-85, 106, 96);
}
}
```

当然本章实现的游戏功能比较简单，未能引入时间判断。例如，种植 2 个小时才能开花，3 个小时才能结果，以及种植信息的保存（例如使用文件）均没有实现，这些功能读者可以自行完善。



第 14 章

单机版五子棋游戏

14.1 单机版五子棋游戏简介

五子棋是一款家喻户晓的棋类游戏，它的多变性吸引了无数的玩家。下面给大家介绍单机版五子棋程序。本章五子棋游戏为简易五子棋程序，棋盘大小为 15×15，白子先落。可以单击鼠标右键悔棋，最多悔 3 步。在每次下棋子前，需先判断该处有无棋子，有则不能落子，超出边界不能落子。任何一方有达到横向、竖向、斜向、反斜向连到 5 个棋子则胜利。本章五子棋游戏运行界面如图 14-1 所示。

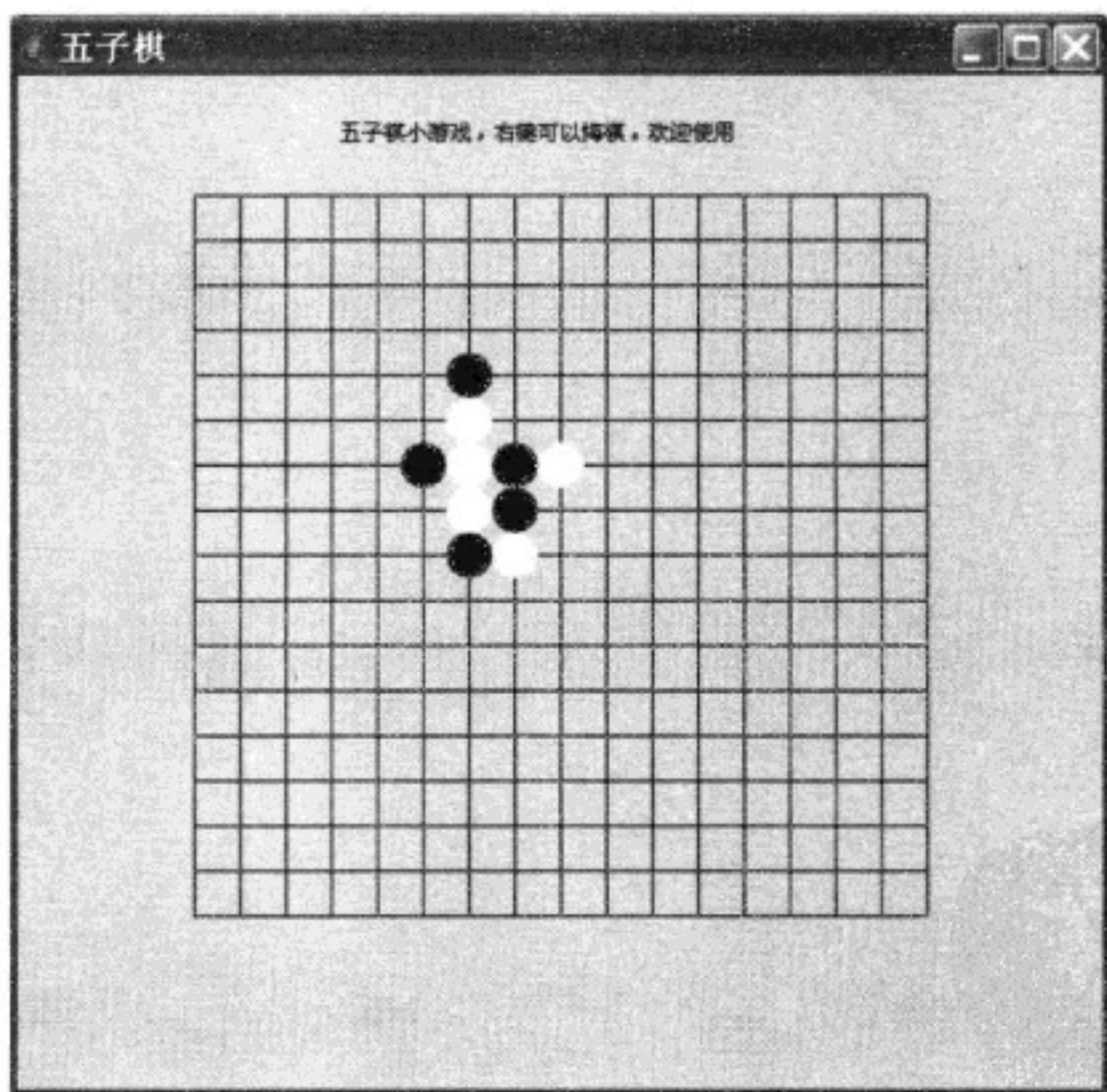


图 14-1 五子棋游戏运行界面

14.2 五子棋设计思想

在下棋过程中，为了保存下过的棋子位置使用了 Vector 向量 v ， v 存储双方的每步走棋信息，

每步走棋信息以“x 坐标—y 坐标”形式存储。同时黑白方各自也使用了 Vector 向量 white, black 保存各自的走棋信息, 便于统计是否五子相连。

在游戏运行过程中, 在鼠标单击事件中判断单击位置是否合法, 既不能在已有棋子的位置单击, 也不能超出游戏棋盘边界, 如果合法则将此位置信息加入 Vector 向量 v 及各自的走棋信息向量中, 同时调用 this.repaint() 方法刷新屏幕并判断游戏的输赢。

14.3 关键技术

14.3.1 Vector 向量容器

向量 (Vector) 是 java.util 包提供的一个用来实现不同类型元素共存的变长数组的工具类。Vector 不但可以保存顺序的一系列数据, 而且还封装了许多有用的方法来操作和处理这些数据, 比数组功能强大。

适合使用 Vector 类的情况包括:

- (1) 需要处理的对象数目不确定, 序列中的元素都是对象, 或者可以表示为对象;
- (2) 需要将不同类的对象组合成一个数据系列;
- (3) 需要频繁地做对象序列中元素的插入和删除操作;
- (4) 经常需要定位序列中的对象或其他查找操作;
- (5) 在不同类之间传递大量的数据。

1. 创建向量类的对象

Vector 类有 3 个构造函数, 最复杂的是:

```
Public Vector (int initCapacity, int capacityIncrement);
```

其中, initCapacity: 表示刚创建时 Vector 序列包含的元素个数。

capacityIncrement: 表示每次向 Vector 序列中追加元素时的增量。

例如: Vector MyVector = new Vector (10, 5);

表示创建的 MyVector 向量序列初始有 10 个元素, 以后不够用时, 按 5 为单位递增。创建时不需要指明元素类型, 使用时再确定。

2. 向量中添加元素

方法 1: 用 addElement() 方法将新元素添加在向量序列的尾部。

格式:

```
addElement (Object obj);
```

方法 2: 用 insertElement() 方法将新元素插入在向量序列的指定位置处。

格式:

```
insertElement (Object obj, int index);
```

其中 index 为插入位置, 0 表示第 1 个位置。

例如:

```
Vector MyVector=new Vector();
```

```
for ( int i=0; i<10;i++)
```

```
{
```

```
MyVector.addElement( new D200_Card(200180000+i, 1111, 50.0, "200", 0.10));
```



```

}
MyVector. insertElement(new IP_Card(123000,22,10.0,"200"),0);

```

3. 修改或删除向量序列中的元素

(1) void setElementAt(Object obj, int index);

将向量序列 index 位置处的对象元素设置为 obj, 如果此位置原来有元素则被覆盖。

(2) boolean removeElement (Object obj);

删除向量序列中第一个与指定的 obj 对象相同的元素, 同时将后面的元素前移。

(3) void removeElementAt(int index);

删除 index 指定位置处的元素, 同时将后面的元素前移。

(4) void removeAllElements();

清除向量序列中的所有元素。

4. 查找向量序列中的元素

(1) Object elementAt(int index);

返回指定位置处的元素。通常需要进行强制类型转换。

(2) boolean contains (Object obj);

检查向量序列中是否包含与指定的 obj 对象相同的元素, 是则返回 True 否则 False。

(3) int indexOf(Object obj,int start_index);

从指定的 start_index 位置开始向后搜索, 返回所找到的第一个与指定对象相同元素的下标位置, 若指定对象不存在则返回-1。

(4) int lastindexOf(Object obj,int start_index);

从指定的 start_index 位置开始向前搜索, 返回所找到的第一个与指定对象相同元素的下标位置, 若指定对象不存在则返回-1。

14.3.2 判断输赢的算法

本游戏的关键技术是判断输赢的算法, 对于算法的具体实现大致分为以下几种情况。

- 判断 $x=y$ 轴上是否形成五子连珠。
- 判断 $x=-y$ 轴上是否形成五子连珠。
- 判断 x 轴上是否形成五子连珠。
- 判断 y 轴上是否形成五子连珠。

以上 4 种情况只要任何一种情况成立, 就可以判断输赢。判断输赢实际上不用扫描整个棋盘, 如果能得到刚下的棋子位置(int x,int y), 就无需扫描整个棋盘, 而仅在此棋子附近横竖斜方向均判断一遍即可。

程序中用 victory(int x,int y,Vector contain) 方法来判断输赢。victory(int x,int y, Vector contain) 方法中前两个参数为走棋位置, 第 3 个参数为保存该方所有的走棋信息 Vector 向量。分别计算以 (int x,int y) 为中心的 4 个方向上棋子的数量, 由于 contain 保存的仅是己方的棋子, 因此在某个方向上判断时只需判断 contain 是否包含此位置, 如果包含此位置, 即说明此处有己方棋子。

判断 4 种情况下是否连成五子, 返回 True 或 False。

本程序中每下一步棋子, 就调用 victory(int x,int y,Vector contain) 方法判断是否已经连成五子, 如果返回 True, 则说明已经连成五子, 将显示输赢结果对话框。

14.4 程序设计的步骤

14.4.1 设计窗口类 (wuziqi2.java)

编写一个继承 JFrame 类的 wuziqi2 窗体类, 用于完成游戏的各种操作。

导入包及相关类:

```
import java.awt.*;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.util.Vector;
import javax.swing.*;
```

wuziqi2 窗体类实现鼠标监听接口, 并定义一些成员变量。代码如下:

```
public class wuziqi2 extends JFrame implements MouseListener{
    Vector v=new Vector();           //所有每步走棋的信息
    Vector white=new Vector();        //白方走棋信息
    Vector black=new Vector();        //黑方走棋信息
    boolean b;                       //判断白棋还是黑棋
    int blackcount,whitecount;        //计算悔棋步数
    int w=25;                         //间距大小, 是双数
    int px=100,py=100;               //棋盘的坐标
    int pxw=(px+w), pyw=(py+w);
    int width=w*16,height=w*16;
    int vline=(width+px);            //垂直线的长度
    int hline=(height+py);           //水平线的长度
```

wuziqi2 窗体类构造方法, 添加鼠标监听器, 设置窗体背景颜色为 Color.orange。代码如下:

```
/**
 * 构造方法
 */
public wuziqi2(){
    super("五子棋");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //关闭按钮
    Container con=this.getContentPane();
    con.setLayout(new BorderLayout());
    this.addMouseListener(this); //添加鼠标监听器
    this.setSize(600,600); //设置窗体大小
    this.setBackground(Color.orange);
    this.setVisible(true);
}
```

窗体 paint(Graphics g)事件中重画棋盘及所有下过的棋子, 这些棋子信息保存在 Vector 向量 v 中。代码如下:

```
/**
 * 画棋盘及棋子
 */
public void paint(Graphics g){
```



```

g.clearRect(0, 0, this.getWidth(), this.getHeight()); //清除面板
//URL url = getClass().getResource("qipian.jpg"); //指定图片路径
//ImageIcon image = new ImageIcon(url); //创建 ImageIcon 对象
//g.drawImage(image.getImage(), 100, 100, this); //将图片绘制到面板上
g.setColor(Color.BLACK); //设置网格颜色
g.drawRect(px, py, width, height); //网格大小
g.drawString("五子棋小游戏, 右键可以悔棋, 欢迎使用", 180, 70);
for(int i=0;i<15;i++){
    g.drawLine(pxw+i*w,py,pxw+i*w,hline); //每条横线和竖线
    g.drawLine(px,pyw+i*w,vline,pyw+i*w);
}
for(int x=0;x<v.size();x++){
    String str=(String)v.get(x);
    String tmp[]=str.split("-");
    int a=Integer.parseInt(tmp[0]);
    int b=Integer.parseInt(tmp[1]);
    a=a*w+px;
    b=b*w+py;
    if(x%2==0){
        g.setColor(Color.WHITE);
    }else{
        g.setColor(Color.BLACK);
    }
    g.fillArc(a-w/2, b-w/2, w, w,0,360);
}
}

```

鼠标单击事件中判断单击位置是否合法, 既不能在已有棋子的位置单击, 也不能超出游戏棋盘边界, 如果合法则将此位置信息加入 Vector 向量 v 及各自的走棋信息向量中, 同时调用 this.repaint() 方法刷新屏幕并判断游戏的输赢。下子时白子先落, 因此判断轮到哪方走棋通过 v.size() 值的奇偶来判断, 如果为偶数则轮到黑方下棋, 如果为奇数则轮到白方下棋。

悔棋实现也很简单, 仅需要从保存下过的棋子的位置 Vector 向量 v 中移除最后一项 (即刚走的棋子的位置信息) 即可。这样在重画时刚走的棋子就不会再显示出来了, 因为重画 paint 事件是根据 Vector 向量 v 中保存的棋子位置信息进行重画的。代码如下:

```

public void mouseClicked(MouseEvent e) {
    if(e.getButton()==e.BUTTON1){
        int x=e.getX();
        int y=e.getY();
        x=(x-x%w)+(x%w>w/2?w:0);
        y=(y-y%w)+(y%w>w/2?w:0);
        x=(x-px)/w;
        y=(y-py)/w;
        if(x>=0&&y>=0&&x<=16&&y<=16){
            if(v.contains(x+"-"+y)){
                System.out.println("已有棋了");
            }
            else{
                v.add(x+"-"+y);
                this.repaint();
                if(v.size()%2==0){
                    black.add(x+"-"+y);
                }
            }
        }
    }
}

```



```

        this.victory(x, y, black);
        System.out.println("黑棋");
    }
    else{
        white.add(x+"-"+y);
        this.victory(x, y, white);
        System.out.println("白棋");
    }
    System.out.println(e.getX()+"-"+e.getY());
}
}
else{
    System.out.println(e.getX()+"-"+e.getY()+"|"+ x+"-"+y+"\t 超出边界");
}
}
if(e.getButton()==e.BUTTON3){ //单击右键的悔棋方法
    System.out.println("鼠标右键-悔棋");
    if(v.isEmpty()){
        JOptionPane.showMessageDialog(this, "没有棋可以悔");
    }
    else{
        if(v.size()%2==0){ //判断是白方悔棋还是黑方悔棋
            blackcount++;
            if(blackcount>3){
                JOptionPane.showMessageDialog(this, "黑棋已经悔了三步");
            }
            else{
                v.remove(v.lastElement());
                this.repaint();
            }
        }
        else{
            whitecount++;
            if(whitecount>3){
                JOptionPane.showMessageDialog(this, "白棋已经悔了三步");
            }
            else{
                v.remove(v.lastElement());
                this.repaint();
            }
        }
    }
}
}
}
}

```

本游戏的关键地方在于判断游戏的输赢。判断输赢的方法 victory(int x,int y, Vector contain)中前两个参数为走棋位置,第 3 个参数为保存该方所有的走棋信息 Vector 向量。分别计算以(int x,int y)为中心的 4 个方向上棋子的数量,由于 contain 保存的仅是己方的棋子,因此在某个方向上判断时只需判断 contain 是否包含此位置,如果包含此位置,即说明此处有己方棋子。

例如,以(int x,int y)为中心计算水平方向棋子数量时,首先向右最多 4 个位置,判断 contain 是否包含此位置,如果包含则 ch 加 1。然后向左最多 4 个位置,判断 contain 是否包含此位置,如果包含则 ch 加 1。代码如下:


```

for(int i=1;i<5;i++){                                //向右
    if(contain.contains((x+i)+"-"+y))
        ch++;
    else
        break;
}
for(int i=1;i<5;i++){                                //向左
    if(contain.contains((x-i)+"-"+y))
        ch++;
    else
        break;
}

```

统计完成后如果 $ch \geq 4$ 则说明水平方向已连成五子, 因为下子处(int x,int y)还有己方一个棋子。其他方向同理。代码如下:

```

public void victory(int x,int y,Vector contain){      //判断输赢的方法
    int cv=0;                                          //计算垂直方向棋子数量的变量
    int ch=0;                                          //计算水平方向棋子数量的变量
    int cil=0;                                         //计算斜面方向棋子数量的变量 1
    int ci2=0;                                         //计算斜面方向棋子数量的变量 2
    //计算水平方向棋子的数量
    for(int i=1;i<5;i++){
        if(contain.contains((x+i)+"-"+y))
            ch++;
        else
            break;
    }
    for(int i=1;i<5;i++){
        if(contain.contains((x-i)+"-"+y))
            ch++;
        else
            break;
    }
    //计算垂直方向棋子的数量
    for(int i=1;i<5;i++){
        if(contain.contains(x+"-"+(y+i)))
            cv++;
        else
            break;
    }
    for(int i=1;i<5;i++){
        if(contain.contains(x+"-"+(y-i)))
            cv++;
        else
            break;
    }
    //计算 45°斜面方向棋子的数量
    for(int i=1;i<5;i++){
        if(contain.contains((x+i)+"-"+(y+i)))
            cil++;
        else
            break;
    }
}

```



```

        for(int i=1;i<5;i++){
            if(contain.contains((x-i)+"-"+(y-i)))
                ci1++;
            else
                break;
        }
        //计算 135°斜面方向棋子的数量
        for(int i=1;i<5;i++){
            if(contain.contains((x-i)+"-"+(y+i)))
                ci2++;
            else
                break;
        }
        for(int i=1;i<5;i++){
            if(contain.contains((x+i)+"-"+(y-i)))
                ci2++;
            else
                break;
        }
        if(ch>=4||cv>=4||ci1>=4||ci2>=4){
            System.out.println(v.size()+"步棋");
            if(v.size()%2==0){
                //判断向量 v.size() 的值, 结果为偶数则黑棋方胜利, 结果为奇数则白棋方胜利
                JOptionPane.showMessageDialog(null,"恭喜你, 黑棋方赢了");
            }
            else{
                JOptionPane.showMessageDialog(null,"恭喜你, 白棋方赢了");
            }
            this.v.clear();
            this.black.clear();
            this.white.clear();
            this.repaint();
        }
    }
}

```

由于下子时白子先落, 判断输赢时如果向量 `v.size()` 的值结果为偶数则黑棋方胜利, 结果为奇数则白棋方胜利。

第 15 章

网络五子棋游戏

15.1 网络五子棋游戏简介

本章介绍应用 Java 语言的 Socket 编程方法开发的“网络五子棋”程序。网络五子棋采用 C/S 架构，分为服务器端和客户端。服务器端运行界面如图 15-1 所示，开始游戏时服务器首先启动，单击“侦听”按钮启动服务器，侦听是否有客户端连接，如果有连接则进入聊天和下棋功能，同时“侦听”按钮文字变成“正在聊天”。

用户根据提示信息，轮到自己下棋才可以在棋盘上落子，“悔棋”按钮可以在对方还没落子前使用。在下棋过程中服务器端用户和客户端用户之间可以聊天，服务器端用户通过“发送”按钮发送聊天信息。

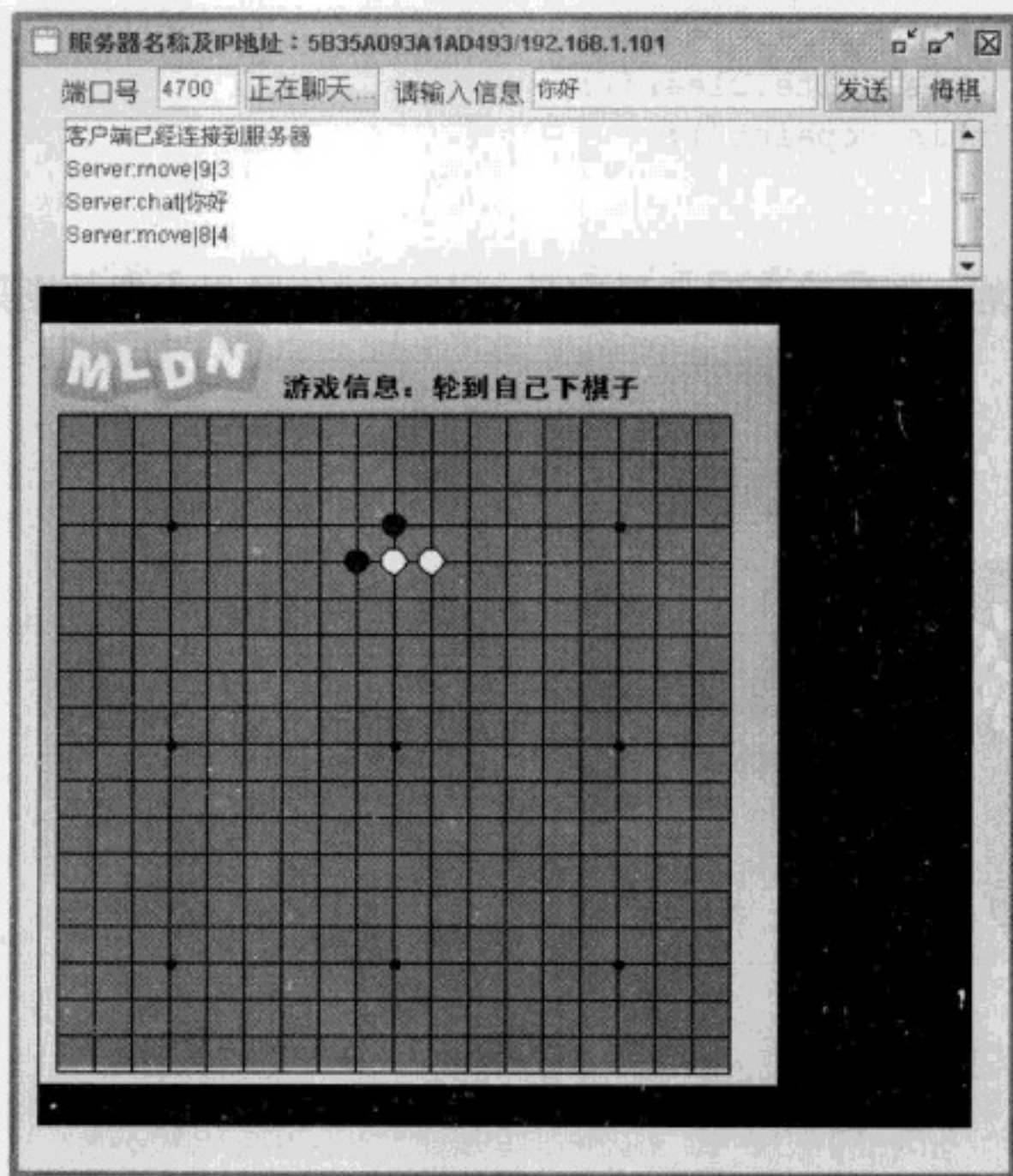


图 15-1 网络五子棋游戏服务器端界面

客户端运行界面如图 15-2 所示，需要输入服务器 IP 地址，如果正确且服务器已经启动则可以“连接”服务器。连接成功则“连接”按钮文字变成“正在聊天”。

用户根据提示信息，轮到自己下棋才可以在棋盘上落子，“悔棋”按钮可以在对方还没落子前使用。在下棋过程中客户端通过“发送”按钮发送聊天信息。

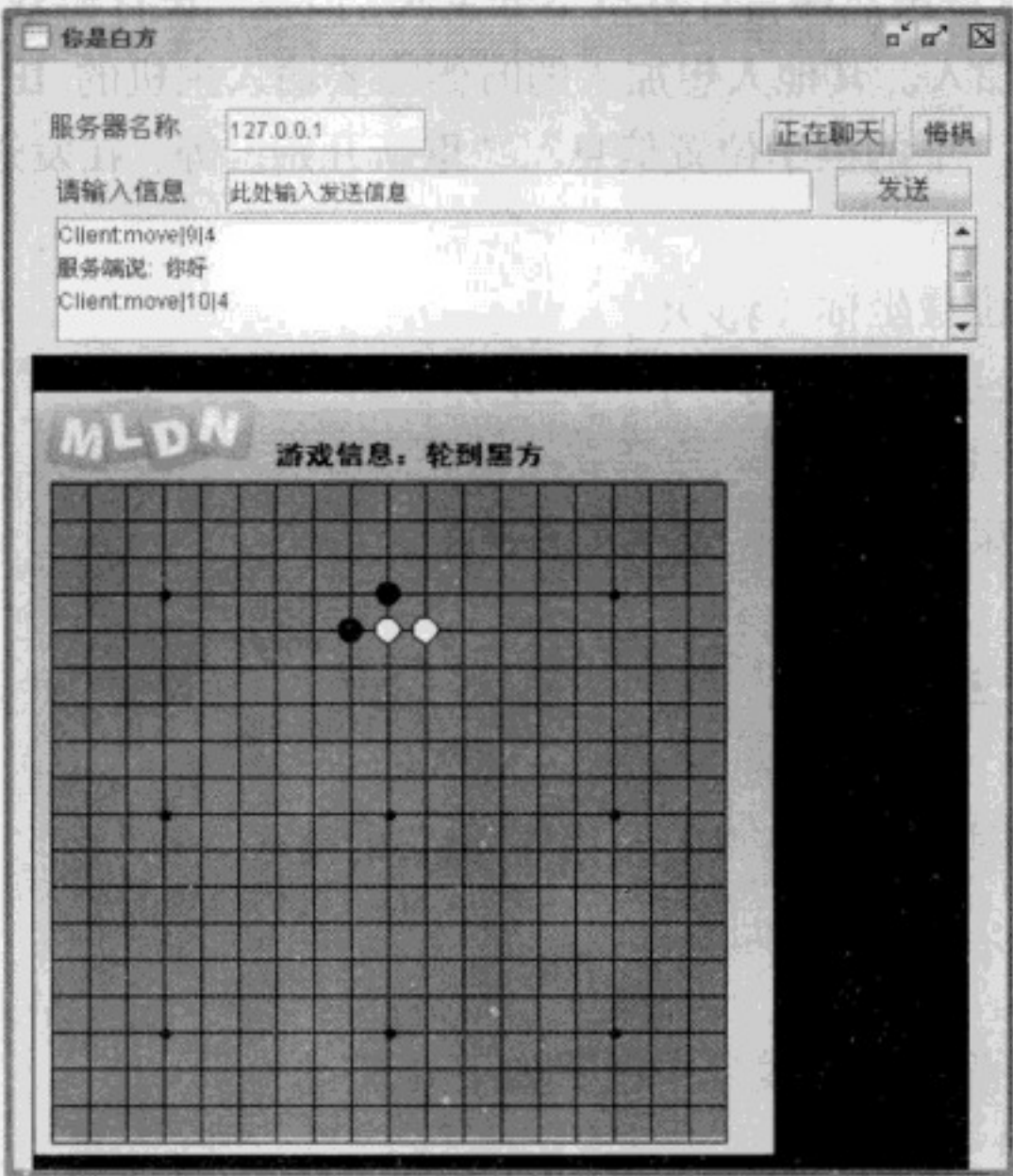


图 15-2 网络五子棋游戏客户端界面

15.2 五子棋设计思想

15.2.1 界面设计

下棋需要有棋盘，程序中通过继承 JPanel 面板类 GobangPanel 显示图 15-3 所示的棋盘背景图片。而棋盘线条、准星点位及双方的落子是绘制出来的。游戏界面中要求用户输入服务器 IP、端口等。

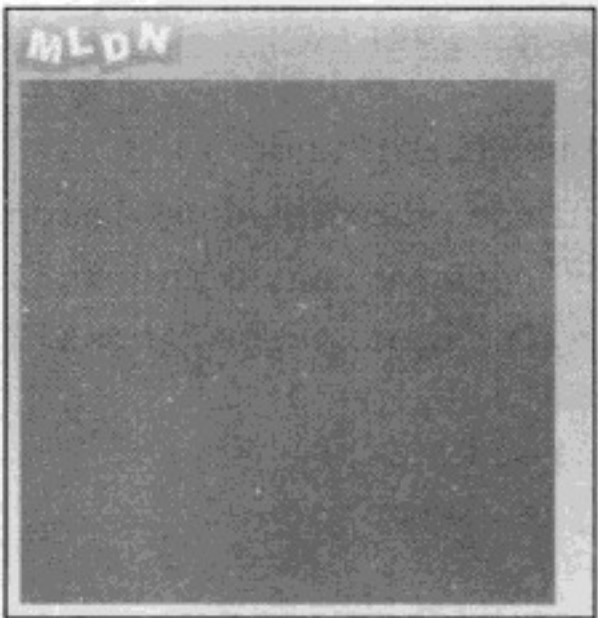


图 15-3 棋盘背景

15.2.2 通信协议

设计的难点在于与对方需要通信。这里使用了 UDP (User Data Protocol)。UDP 是用户数据报文协议的简称,两台计算机之间的传输类似于传递邮件;两台计算机之间没有明确的连接,使用 UDP 建立对等通信。这里虽然两台计算机不分主次,但我们设计游戏时假设一台计算机作为主机(黑方),等待其他人加入。其他人想加入的时候需要输入主机的 IP。为了区分通信中传送的信息,例如“输赢信息”,“下的棋子位置信息”,“重新开始”等,在发送信息的首部要加上代号。因此定义了如下协议:

(1) move|下的棋子位置坐标(x,y);

例如:“move|1|1”表示对方下的棋子位置坐标(1,1)。

(2) over|哪方赢的信息;

例如:“over|游戏结束,黑方胜”表示黑方赢了。

(3) quit|

表示游戏结束,对方离开了。

(4) undo|x|y;

悔棋命令,表示撤销刚才自己在(x,y)坐标位置的落子。

(5) chat|聊天内容;

文字聊天协议。

基于以上协议,在接收信息的线程中做了如下处理:

```
public void run() {
    try {
        while (true) {
            this.sleep(100);
            instr = new BufferedReader(new InputStreamReader(socket
                .getInputStream()));
            if (instr.ready()) { //检查是否有数据
                String cmd = instr.readLine();
                //在每个字符处进行分解
                ss = cmd.split("\\|");
                if (cmd.startsWith("move")) {
                    message = "轮到自己下棋子";
                    int x = Integer.parseInt(ss[1]);
                    int y = Integer.parseInt(ss[2]);
                    allChess[x][y] = 2;
                    panel2.repaint();
                    canPlay = true;
                }
                if (cmd.startsWith("undo")) {
                    JOptionPane.showMessageDialog(null, "对方撤销上步棋");
                    int x = Integer.parseInt(ss[1]);
                    int y = Integer.parseInt(ss[2]);
                    allChess[x][y] = 0;
                    panel2.repaint();
                    canPlay = false;
                }
                if (cmd.startsWith("over")) {
                    JOptionPane.showMessageDialog(null, message);
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



```

        panel2.setEnabled(false);
        canPlay = false;
    }
    if (cmd.startsWith("quit")) {
        JOptionPane.showMessageDialog(null, "游戏结束, 对方离开了");
        panel2.setEnabled(false);
        canPlay = false;
    }
    if (cmd.startsWith("chat")) {
        jTextArea1.append("客户端说: " + ss[1] + "\n");
    }
}
} catch (Exception ex) {
    System.out.print("error: " + ex);
}
}

```

在下棋过程中, 为了保存下过的棋子的位置使用了 allChess 数组, allChess 数组初始值为 0, 表示此处无棋子。allChess 数组还可以存储值 1 和 2, 其中 1 表示该点是黑子, 2 表示该点是白子。

15.3 关键技术

15.3.1 Socket 技术

基于 TCP/IP 网络的 Java 程序与其他程序通信中, Java 程序依靠 Socket 进行通信。Socket 可以看成为在两个程序进行通信连接中的一个端点, 一个程序将一段信息写入 Socket 中, 该 Socket 将这段信息发送给另外一个 Socket, 使这段信息能传送到其他程序中。

无论何时, 在两个网络应用程序之间发送和接收信息时都需要建立一个可靠的连接, 流套接字依靠 TCP 来保证信息正确地到达目的地。实际上, IP 包有可能在网络中丢失或者在传送过程中发生错误, 任何一种情况发生, 作为接收方的 TCP 将联系发送方 TCP 重新发送这个 IP 包。这就是所谓的在两个流套接字之间建立可靠的连接。

流套接字在 C/S 程序中扮演一个必需的角色, 客户机程序 (需要访问某些服务的网络应用程序) 创建一个扮演服务器程序的主机的 IP 地址和服务器程序 (为客户端应用程序提供服务的网络应用程序) 的端口号的流套接字对象。

客户端流套接字的初始化代码将 IP 地址和端口号传递给客户端主机的网络管理软件, 管理软件将 IP 地址和端口号通过 NIC 传递给服务器端主机; 服务器端主机读到通过 NIC 传递来的数据, 然后查看服务器程序是否处于监听状态, 这种监听依然是通过套接字和端口来进行的。如果服务器程序处于监听状态, 那么服务器端网络管理软件就向客户机网络管理软件发出一个积极的响应信号, 接收到响应信号后, 客户端流套接字初始化代码就给客户程序建立一个端口号, 并将这个端口号传递给服务器程序的套接字 (服务器程序将使用这个端口号识别传来的信息是否属于客户程序), 同时完成流套接字的初始化。

如果服务器程序没有处于监听状态, 那么服务器端网络管理软件将给客户端传递一个消极信号, 收到这个消极信号后, 客户程序的流套接字初始化代码将抛出一个异常对象并且不建立通信

连接，也不创建流套接字对象。这种情形就像打电话一样，当有人的时候通信建立，否则电话将被挂断。

这部分的工作包括相关联的 3 个类：InetAddress、Socket 和 ServerSocket。InetAddress 对象描绘了 32 位或 128 位 IP 地址，Socket 对象代表客户程序流套接字；ServerSocket 对象代表服务程序流套接字，这 3 个类均位于 java.net 包中。

15.3.2 InetAddress 类简介

InetAddress 类在网络 API 套接字编程中扮演了一个重要角色。InetAddress 描述了 32 位或 128 位 IP 地址，要完成这个功能，InetAddress 类主要依靠 Inet4Address 和 Inet6Address 两个支持类。这 3 个类是继承关系，InetAddress 是父类，Inet4Address 和 Inet6Address 是子类。

由于 InetAddress 类只有一个构造函数，而且不能传递参数，因此不能直接创建 InetAddress 对象，比如下面的语句就是错误的：

```
InetAddress ia = new InetAddress();
```

但我们可以通过下面的 5 个静态方法来创建一个 InetAddress 对象或 InetAddress 数组。

(1) getAllByName (String host) 方法：返回一个 InetAddress 对象数组的引用，每个对象包含一个表示相应主机名的单独的 IP 地址，这个 IP 地址是通过 host 参数传递的，对于指定的主机，如果没有 IP 地址存在，那么该方法将抛出一个 UnknownHostException 异常对象。

(2) getByAddress (byte [] addr) 方法：返回一个 InetAddress 对象的引用，这个对象包含了一个 IPv4 地址或 IPv6 地址，IPv4 地址是一个 4 字节地址数组，IPv6 地址是一个 16 字节地址数组，如果返回的数组既不是 4 字节的也不是 16 字节的，那么该方法将会抛出一个 UnknownHostException 异常对象。

(3) getByAddress (String host, byte [] addr) 方法：返回一个 InetAddress 对象的引用，这个 InetAddress 对象包含了一个由 host 和 4 字节的 addr 数组指定的 IP 地址，或者是 host 和 16 字节的 addr 数组指定的 IP 地址，如果这个数组既不是 4 字节的也不是 16 字节的，那么该方法将抛出一个 UnknownHostException 异常对象。

(4) getName (String host) 方法：返回一个 InetAddress 对象，该对象包含了一个与 host 参数指定的主机相对应的 IP 地址，对于指定的主机，如果没有 IP 地址存在，那么该方法将抛出一个 UnknownHostException 异常对象。

(5) getLocalHost()方法：返回一个 InetAddress 对象，这个对象包含了本地主机的 IP 地址，考虑到本地主机既是客户程序主机又是服务器程序主机，为避免混乱，我们将客户程序主机称为客户主机，将服务器程序主机称为服务器主机。

InetAddress 和它的子类型对象处理主机名到主机 IPv4 或 IPv6 地址的转换，完成这个转换需要使用域名系统，下面的代码示范了如何通过调用 getName (String host) 方法获得 InetAddress 子类对象的方法，这个对象包含了与 host 参数相对应的 IP 地址：

```
InetAddress ia = InetAddress.getName ("www.sun.com");
```

一旦获得了 InetAddress 子类对象的引用就可以调用 InetAddress 的各种方法来获得 InetAddress 子类对象中的 IP 地址信息。例如，可以通过调用 getCanonicalHostName()方法从域名服务中获得标准的主机名；getHostAddress()方法获得 IP 地址；getHostName()方法获得主机名；isLoopbackAddress()方法判断 IP 地址是否是一个 loopback 地址。

下面程序使用 InetAddress 获取本机 IP 及主机名等信息。


```
import java.net.*;
class InetAddressDemo{
    public static void main (String [] args) throws UnknownHostException{
        String host = "localhost";
        InetAddress ia = InetAddress.getByName (host);
        System.out.println ("Canonical Host Name = " +
            ia.getCanonicalHostName ());
        System.out.println ("Host Address = " +ia.getHostAddress ());
        System.out.println ("Host Name = " +   ia.getHostName ());
        System.out.println ("Is Loopback Address = " +ia.isLoopbackAddress ());
    }
}
```

在 Eclipse 中进行调试时, 控制台窗口输出的结果如下:

```
Canonical Host Name = localhost
Host Address = 127.0.0.1
Host Name = localhost
Is Loopback Address = true
```

InetAddressDemo 通过调用 `getByName(String host)` 方法获得一个 `InetAddress` 子类对象的引用, 通过这个引用获得了标准主机名、主机地址、主机名以及 IP 地址是否是 loopback 地址的输出。

15.3.3 ServerSocket 类

由于 `SocketDemo` 使用了流套接字, 因此服务程序也要使用流套接字, 这就要创建一个 `ServerSocket` 对象。`ServerSocket` 有几个构造函数, 最简单的是:

```
ServerSocket (int port);
```

当使用 `ServerSocket (int port)` 创建一个 `ServerSocket` 对象时, `port` 参数传递端口号, 这个端口就是服务器监听连接请求的端口。如果在这时出现错误将抛出 `IOException` 异常对象, 否则将创建 `ServerSocket` 对象并开始准备接收连接请求。

接下来服务程序进入无限循环之中。无限循环从调用 `ServerSocket` 的 `accept()` 方法开始, 在调用开始后 `accept()` 方法将导致调用线程阻塞直到连接建立。在建立连接后 `accept()` 方法返回一个最近创建的 `Socket` 对象, 该 `Socket` 对象绑定了客户程序的 IP 地址或端口号。

由于存在单个服务程序与多个客户程序通信的可能, 因此服务程序响应客户程序不应该花很多时间, 否则客户程序在得到服务前有可能花很多时间来等待通信的建立, 然而服务程序和客户程序的会话有可能是很长的 (这与电话类似)。因此为加快对客户程序连接请求的响应, 典型的方法是服务器主机运行一个后台线程, 这个后台线程处理服务程序和客户程序的通信。

为了示范我们在上面谈到的概念并完成 `SocketDemo` 程序, 下面我们创建一个 `ServerDemo` 程序。该程序将创建一个 `ServerSocket` 对象来监听端口 10000 的连接请求, 如果成功, 服务程序将等待连接输入, 开始一个线程处理连接, 并响应来自客户程序的命令。

ServerSocket 示例程序 ServerDemo.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class ServerDemo{
    public static void main (String [] args) throws IOException{
        System.out.println ("Server starting...\n");
        ServerSocket server = new ServerSocket (10000);
        while (true) {
            Socket s = server.accept ();
```



```

        System.out.println ("Accepting Connection...\n");
        new ServerThread (s).start ();
    }
}

class ServerThread extends Thread{
    private Socket s;
    ServerThread (Socket s){
        this.s = s;
    }
    public void run (){
        BufferedReader br = null;
        PrintWriter pw = null;
        try{
            InputStreamReader isr;
            isr = new InputStreamReader (s.getInputStream ());
            br = new BufferedReader (isr);
            pw = new PrintWriter (s.getOutputStream (), true);
            Calendar c = Calendar.getInstance ();
            do{
                String cmd = br.readLine ();
                if (cmd == null)
                    break;
                cmd = cmd.toUpperCase ();
                if (cmd.startsWith ("BYE"))
                    break;
                if (cmd.startsWith ("DATE") || cmd.startsWith ("TIME"))
                    pw.println (c.getTime ().toString ());
                if (cmd.startsWith ("DOM"))
                    pw.println (" " + c.get (Calendar.DAY_OF_MONTH));
                if (cmd.startsWith ("DOW"))
                    switch (c.get (Calendar.DAY_OF_WEEK)) {
                        case Calendar.SUNDAY : pw.println ("SUNDAY");
                        break;
                        case Calendar.MONDAY : pw.println ("MONDAY");
                        break;
                        case Calendar.TUESDAY : pw.println ("TUESDAY");
                        break;
                        case Calendar.WEDNESDAY: pw.println ("WEDNESDAY");
                        break;
                        case Calendar.THURSDAY : pw.println ("THURSDAY");
                        break;
                        case Calendar.FRIDAY : pw.println ("FRIDAY");
                        break;
                        case Calendar.SATURDAY : pw.println ("SATURDAY");
                    }
                if (cmd.startsWith ("DOY"))
                    pw.println (" " + c.get (Calendar.DAY_OF_YEAR));
                if (cmd.startsWith ("PAUSE"))
                    try{
                        Thread.sleep (3000);
                    }
                    catch (InterruptedException e){}
            } while (true);
        }
        catch (IOException e){

```



```

        System.out.println (e.toString ());
    }
    finally{
        System.out.println ("Closing Connection...\n");
        try{
            if (br != null)
                br.close ();
            if (pw != null)
                pw.close ();
            if (s != null)
                s.close ();
        }
        catch (IOException e){}
    }
}
}

```

15.3.4 Socket 类

当客户程序需要与服务器程序通信时，客户程序在客户机创建一个 Socket 对象。Socket 类有几个构造函数，常用的两个构造函数是：

```
Socket (InetAddress addr, int port);
```

```
Socket (String host, int port);
```

两个构造函数都创建了一个基于 Socket 的连接服务器端流套接字的流套接字。对于第一个构造函数，InetAddress 子类对象通过 addr 参数获得服务器主机的 IP 地址；对于第二个构造函数，host 参数包被分配到 InetAddress 对象中，如果没有 IP 地址与 host 参数相一致，那么将抛出 UnknownHostException 异常对象。两个函数都通过参数 port 获得服务器的端口号。假设已经建立了连接，网络 API 将在客户端基于 Socket 的流套接字中捆绑客户程序的 IP 地址和任意一个端口号，否则两个函数都会抛出一个 IOException 对象。

如果创建了一个 Socket 对象，那么它可能通过调用 Socket 的 getInputStream()方法从服务程序获得输入流读传送来的信息，也可能通过调用 Socket 的 getOutputStream()方法获得输出流来发送消息。在读写活动完成之后，客户程序调用 close()方法关闭流和流套接字。下面的代码创建了一个服务程序主机地址为 198.163.227.6，端口号为 13 的 Socket 对象，然后从这个新创建的 Socket 对象中读取输入流，然后再关闭流和 Socket 对象。

```

Socket s = new Socket ("198.163.227.6", 13);
InputStream is = s.getInputStream ();    // 从 Socket 流中读入
is.close ();
s.close ();

```

接下来我们将示范一个流套接字的客户程序。这个程序将创建一个 Socket 对象，Socket 将访问运行在指定主机端口 10000 上的服务程序，如果访问成功，客户程序将给服务程序发送一系列命令并打印服务程序的响应。Socket 使用示例程序 SocketDemo.Java，源代码如下：

```

import java.io.*;
import java.net.*;
class SocketDemo{
    public static void main (String [] args){
        String host = "localhost";
        if (args.length == 1)
            host = args [0];
        BufferedReader br = null;

```



```

PrintWriter pw = null;
Socket s = null;
try{
    s = new Socket (host, 10000);
    InputStreamReader isr;
    isr = new InputStreamReader (s.getInputStream ());
    br = new BufferedReader (isr);
    pw = new PrintWriter (s.getOutputStream (), true);
    pw.println ("DATE");
    System.out.println (br.readLine ());
    pw.println ("PAUSE");
    pw.println ("DOW");
    System.out.println (br.readLine ());

    pw.println ("DOM");
    System.out.println (br.readLine ());
    pw.println ("DOY");
    System.out.println (br.readLine ());
}
catch (IOException e){
    System.out.println (e.toString ());
}
finally{
    try{
        if (br != null)
            br.close ();
        if (pw != null)
            pw.close ();
        if (s != null)
            s.close ();
    }
    catch (IOException e){}
}
}
}

```

运行这段程序将会得到图 15-4 所示的结果，图 15-5 是运行 ServerDemo.java 后服务器端程序运行结果发生的变化。这里必须保证服务器端的程序已经运行了，否则会显示服务器不能连接的错误。



图 15-4 ServerDemo.java 程序的运行结果



图 15-5 ServerDemo.java 运行界面的变化

SocketDemo.Java 创建了一个 Socket 对象与运行在主机端口 10000 的服务程序联系，主机的 IP 地址由 host 变量确定。程序将获得 Socket 的输入输出流，围绕 BufferedReader 的输入流和 PrintWriter 的输出流对字符串进行读写操作就变得非常容易。程序向服务程序发出各种 date/time 命令并得到响应，每个响应均被打印，一旦最后一个响应被打印，将执行 try/catch/finally 结构的

finally 子串, finally 子串将在关闭 Socket 之前关闭 BufferedReader 和 PrintWriter。(说明: 该程序的服务程序在本节的第三部分介绍, 这里先简单介绍一下上面程序用到的简单 date/time 命令。“DATE”命令指示传送服务器时间; “PAUSE”命令指示服务器线程暂停 3s; “DOW”命令指示传送服务器当前日期是一周的第几天; “DOM”命令指示传送服务器当前日期是当月的第几天; “DOY”命令指示传送服务器当前日期是当年的第几天。)

另外, Socket 类包含了许多有用的方法, 如 getLocalAddress()方法将返回一个包含客户程序 IP 地址的 InetAddress 子类对象的引用; getLocalPort()方法将返回客户程序的端口号; getInetAddress()方法将返回一个包含服务器 IP 地址的 InetAddress 子类对象的引用; getPort()方法将返回服务程序的端口号。

15.4 程序设计的步骤

15.4.1 设计服务器端类 (Server.java)

编写一个继承 JFrame 类的 Server 窗体类, 用于完成游戏的服务器端各种操作。

导入包及相关类:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import javax.imageio.ImageIO;
import java.awt.image.*;
```

Server 窗体类实现鼠标侦听接口, 并定义一些成员变量。

```
public class Server extends JFrame implements ActionListener {
    JPanel contentPane;
    JLabel jLabel2 = new JLabel();
    JTextField jTextField2 = new JTextField("4700");
    JButton jButton1 = new JButton();           //侦听按钮
    JLabel jLabel3 = new JLabel();
    JTextField jTextField3 = new JTextField();
    JButton jButton2 = new JButton();           //发送按钮
    JButton jButton3 = new JButton();           //悔棋按钮
    JScrollPane jScrollPane1 = new JScrollPane();
    JTextArea jTextArea1 = new JTextArea();     //显示聊天内容
    ServerSocket server = null;
    Socket socket = null;
    BufferedReader instr = null;
    PrintWriter os = null;
    public static String[] ss = new String[10];
    //保存刚下的棋子的坐标
    int x = 0;
    int y = 0;
```

双方的落子信息保存在二维数组 allChess[19][19]中, 其中 0 表示这个位置没有棋子; 1 表示这个位置是黑子; 2 表示这个位置是白子。由于服务器作为黑方, 因此 isBlack = true。


```

int[][] allChess = new int[19][19];
boolean isBlack = true;           //自己是黑方
//标识当前游戏是否可以继续
boolean canPlay = true;
//保存显示的提示信息
String message = "";              // "自己是黑方先行";
JPanel panell = new JPanel();
GobangPanel panel2 = new GobangPanel(); // 实例化 GobangPanel 对象

```

在 Server 窗体类的构造方法中, 添加动作监听器, 设置窗体中各个组件的位置, 主要有两个 Panel 组成, 一个 Panel 中放置“侦听”按钮 jButton1、“发送”按钮 jButton2、“悔棋”按钮 jButton3 及显示聊天内容的文本区域 jTextArea1。另一个 GobangPanel 实例 panel2 中, 放置棋盘背景图片, 并能接收鼠标单击事件。在窗口关闭事件中向对方发送离开信息 quit。代码如下:

```

/**
 * 服务器端构造方法
 */
public Server() {
    jbInit();
}
//各种组件的初始化
private void jbInit() {
    contentPane = (JPanel) this.getContentPane();
    this.setSize(new Dimension(540, 640));
    this.setTitle("服务器");
    jLabel2.setBounds(new Rectangle(22, 0, 72, 28));
    jLabel2.setText("端口号");
    jLabel2.setFont(new java.awt.Font("宋体", 0, 14));
    jTextField2.setBounds(new Rectangle(73, 0, 45, 24));
    jButton1.setBounds(new Rectangle(120, 0, 73, 25));
    jButton1.setFont(new java.awt.Font("Dialog", 0, 14));
    jButton1.setBorder(BorderFactory.createEtchedBorder());
    jButton1.setActionCommand("jButton1");
    jButton1.setText("侦听");
    jLabel3.setBounds(new Rectangle(200, 0, 87, 28));
    jLabel3.setText("请输入信息");
    jLabel3.setFont(new java.awt.Font("宋体", 0, 14));
    jTextField3.setBounds(new Rectangle(274, 0, 154, 24));
    jTextField3.setText("");
    jButton2.setText("发送");
    jButton2.setActionCommand("jButton1");
    jButton2.setBorder(BorderFactory.createEtchedBorder());
    jButton2.setFont(new java.awt.Font("Dialog", 0, 14));
    jButton2.setBounds(new Rectangle(430, 0, 43, 25));
    jButton3.setText("悔棋");
    jButton3.setActionCommand("jButton1");
    jButton3.setBorder(BorderFactory.createEtchedBorder());
    jButton3.setFont(new java.awt.Font("Dialog", 0, 14));
    jButton3.setBounds(new Rectangle(480, 0, 43, 25));
    jScrollPane1.setBounds(new Rectangle(23, 28, 493, 89));
    jTextField3.setText("此处输入发送信息");
    jTextArea1.setText("聊天内容");
}

```



```

panel1.setLayout(null); // panel1.setLayout(flayout);
panel1.add(jLabel2);
panel1.add(jTextField2);
panel1.add(jButton1);
panel1.add(jLabel3);
panel1.add(jTextField3);
panel1.add(jButton2);
panel1.add(jButton3);
panel1.add(jScrollPane1);

jScrollPane1.getViewport().add(jTextArea1);
contentPane.setLayout(null);
contentPane.add(panel1);
contentPane.add(panel2);
panel1.setBounds(0, 0, 540, 120);
panel2.setBounds(10, 120, 540, 460);
jButton1.addActionListener(this);
jButton2.addActionListener(this);
jButton3.addActionListener(this);
this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) { //窗口关闭事件
        try {
            sendData("quit|"); //向对方发送离开信息
            socket.close();
            instr.close();
            System.exit(0);
        } catch (Exception ex) {
        }
    }
});
}

```

`actionPerformed()`方法在表示对象发生操作时调用。通过 `e.getSource()`来区别哪个按钮发生了单击事件,如果是“侦听”按钮 `jButton1`,则启动服务器侦听;如果是“发送”按钮 `jButton2`,则调用 `sendData()`方法发送文字;如果是“悔棋”按钮 `jButton3`,则发送悔棋信息。代码如下:

```

public void actionPerformed(ActionEvent e) {
    if (e.getSource() == jButton1) { // “侦听”按钮
        int port = Integer.parseInt(jTextField2.getText().trim());
        listenClient(port);
        System.out.print("侦听...");
    }
    if (e.getSource() == jButton2) { // “发送”按钮
        String s = this(jTextField3.getText().trim());
        sendData(s);
        System.out.print("发送文字");
    }
    if (e.getSource() == jButton3) { // “悔棋”按钮
        if (canPlay != true) { //该对方走棋
            allChess[x][y] = 0;
            panel2.repaint();
            canPlay = true;
            String s = "undo|" + x + "|" + y;

```



```

        sendData(s);
        System.out.print("发送悔棋信息");
    } else//对方已走棋
    {
        message = "对方已走棋，不能悔棋了";
        JOptionPane.showMessageDialog(this, message);
        System.out.print("对方已走棋，不能悔棋了");
    }
}
}

```

如果单击了“侦听”按钮 jButton1，则启动服务器侦听，同时修改文字为“正在侦听...”，程序阻塞直到接收到 Socket 连接。这时向对方发送“已经成功连接...”提示，在聊天区域加入文字“客户端已经连接到服务器”。由于服务器作为黑方，因此棋盘上的信息提示为“自己是黑方先行”。最后启动接收聊天信息及下棋信息的线程 MyThread t，线程 t 不断接收相关信息。代码如下：

```

private void listenClient(final int port) { //侦听
    try {
        if (jButton1.getText().trim().equals("侦听")) {
            new Thread(new Runnable() {
                public void run() {
                    // TODO Auto-generated method stub
                    try {
                        server = new ServerSocket(port);
                        jButton1.setText("正在侦听...");
                        socket = server.accept();
                    } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    } //等待，一直到客户端连接才继续执行
                    //this.setTitle("你是黑方");
                    sendData("已经成功连接...");
                    jButton1.setText("正在聊天...");
                    jTextArea1.append("客户端已经连接到服务器\n");
                    message = "自己是黑方先行";
                    panel2.repaint();
                    MyThread t = new MyThread();
                    t.start();
                }
            }).start();
        }
    } catch (Exception ex) {
    }
}

```

内部线程类 MyThread 负责接收数据。当接收到数据时区分是何种数据，如果是“move|x|y”信息，则是对方（白方）走棋信息，修改记录棋盘信息的 allChess 数组，并刷新自己屏幕。如果是“undo|x|y”信息，则是对方（白方）撤销上步棋信息，修改记录棋盘信息的 allChess 数组，并刷新自己屏幕。如果是“over”信息，则是游戏一方胜利了，对话框显示输赢信息。如果是“quit”信息，则是对方离开了，游戏结束。代码如下：

//内部线程类

```
class MyThread extends Thread { //该线程负责接收数据
    public void run() {
        try {
            while (true) {
                this.sleep(100);
                instr = new BufferedReader(new InputStreamReader(socket
                    .getInputStream()));
                if (instr.ready()) { //检查是否有数据
                    String cmd = instr.readLine();
                    JTextArea1.append("客户端: " + cmd + "\n");
                    //在每个“|”字符处进行分解
                    ss = cmd.split("\\|");
                    if (cmd.startsWith("move")) { //对方白子走棋信息
                        int x = Integer.parseInt(ss[1]);
                        int y = Integer.parseInt(ss[2]);
                        allChess[x][y] = 2; //白子
                        message = "轮到自己下棋子";
                        panel2.repaint();
                        canPlay = true;
                    }
                    if (cmd.startsWith("undo")) {
                        JOptionPane.showMessageDialog(null, "对方撤销上步棋");
                        int x = Integer.parseInt(ss[1]);
                        int y = Integer.parseInt(ss[2]);
                        allChess[x][y] = 0;
                        panel2.repaint();
                        canPlay = false;
                    }
                    if (cmd.startsWith("over")) {
                        JOptionPane.showMessageDialog(null, message);
                        panel2.setEnabled(false);
                        canPlay = false;
                    }
                    if (cmd.startsWith("quit")) {
                        JOptionPane.showMessageDialog(null, "游戏结束, 对方离开了!!");
                        panel2.setEnabled(false);
                        canPlay = false;
                    }
                    if (cmd.startsWith("chat")) {
                        JTextArea1.append("客户端说: " + ss[1] + "\n");
                    }
                }
            }
        } catch (Exception ex) {
            System.out.print("error: " + ex);
        }
    }
}
```

sendData(String s)方法负责发送数据给客户端。代码如下:

```
private void sendData(String s) { //发送数据
    try {
```



```

        os = new PrintWriter(socket.getOutputStream());
        os.println(s);
        os.flush();
        if (!s.equals("已经成功连接..."))
            this.jTextArea1.append("Server:" + s + "\n");
    } catch (Exception ex) {
    }
}

```

main()方法实例化 Server 类窗体,并显示服务器名称及 IP 地址。代码如下:

```

public static void main(String arg[]) {
    JFrame.setDefaultLookAndFeelDecorated(true);
    Server frm = new Server();
    frm.setVisible(true);
    try {
        InetAddress address = InetAddress.getLocalHost().getLocalHost();
        frm.setTitle(frm.getTitle()+"名称及 IP 地址: "+address.toString());
    } catch (Exception e) {
        //异常处理代码
    }
}

```

内部类 GobangPanel 继承面板 JPanel,首先显示棋盘背景图片。在面板的刷新方法 paintComponent(Graphics g)中采用双缓冲技术防止屏幕闪烁。双缓冲技术就是将要显示的信息首先绘制到缓冲图片 BufferedImage bi 中,绘制完成后将此图片一次显示到屏幕上,这样可以避免屏幕闪烁。代码如下:

```

class GobangPanel extends JPanel {
    BufferedImage bgImage = null;//棋盘背景图片
    GobangPanel() {
        this.addMouseListener(new MouseLis());
        String imagePath = "";
        try {
            imagePath = System.getProperty("user.dir") + "/background2.jpg";
            bgImage = ImageIO.read(new File(imagePath.replaceAll("\\\\", "/")));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        //双缓冲技术防止屏幕闪烁
        BufferedImage bi = new BufferedImage(500, 500,
            BufferedImage.TYPE_INT_RGB);
        Graphics g2 = bi.createGraphics();
        g2.setColor(Color.BLACK);
        //绘制背景
        g2.drawImage(bgImage, 1, 20, this);
        //输出标题信息
        g2.setFont(new Font("黑体", Font.BOLD, 15));
        g2.drawString("游戏信息: " + message, 130, 60);
    }
}

```



```

//绘制棋盘
for (int i = 0; i < 19; i++) {
    g2.drawLine(10, 70 + 20 * i, 370, 70 + 20 * i);
    g2.drawLine(10 + 20 * i, 70, 10 + 20 * i, 430);
}

//标注点位
g2.fillOval(68, 128, 6, 6);
g2.fillOval(308, 128, 6, 6);
g2.fillOval(308, 368, 6, 6);
g2.fillOval(68, 368, 6, 6);
g2.fillOval(308, 248, 6, 6);
g2.fillOval(188, 128, 6, 6);
g2.fillOval(68, 248, 6, 6);
g2.fillOval(188, 368, 6, 6);
g2.fillOval(188, 248, 6, 6);

//绘制全部棋子
for (int i = 0; i < 19; i++) {
    for (int j = 0; j < 19; j++) {
        if (allChess[i][j] == 1) {
            //黑子
            int tempX = i * 20 + 10;
            int tempY = j * 20 + 70;
            g2.fillOval(tempX - 7, tempY - 7, 14, 14);
        }
        if (allChess[i][j] == 2) {
            //白子
            int tempX = i * 20 + 10;
            int tempY = j * 20 + 70;
            g2.setColor(Color.WHITE);
            g2.fillOval(tempX - 7, tempY - 7, 14, 14);
            g2.setColor(Color.BLACK);
            g2.drawOval(tempX - 7, tempY - 7, 14, 14);
        }
    }
}
g.drawImage(bi, 0, 0, this);
}

```

内部类 MouseLis 继承 MouseAdapter 适配器, 实现 mousePressed(MouseEvent e) 事件。在此事件中首先判断单击位置是否在棋盘中, 并且此处无棋子, 则根据自己是哪方颜色来修改棋盘对应的数组 allChess 此处 allChess [x][y] 元素的值。落子后刷新自己的屏幕, 并调用自己类中的方法 checkWin() 判断这个棋子是否和其他的棋子连成 5 子即输赢判断, 如果赢了则发送游戏结束信息给对方。代码如下:

```

class MouseLis extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
        if (canPlay == true) {
            x = e.getX();
            y = e.getY();
            if (x >= 10 && x <= 370 && y >= 70 && y <= 430) {
                x = x / 20;
                y = (y - 60) / 20;
            }
        }
    }
}

```



```

        if (allChess[x][y] == 0) {
            //判断当前要下的是什么颜色的棋子
            if (isBlack == true) {
                allChess[x][y] = 1;
                isBlack = false;
                message = "轮到白方";
                sendData("move|" + String.valueOf(x) + "|"
                    + String.valueOf(y));
                canPlay = false;
                repaint();
            } else {
                allChess[x][y] = 2;
                isBlack = true;
                message = "轮到黑方";
                sendData("move|" + String.valueOf(x) + "|"
                    + String.valueOf(y));
                canPlay = false;
                //白子
                repaint();
            }
        }
        //判断这个棋子是否和其他的棋子连成 5 子, 即判断游戏是否结束
        boolean winFlag = this.checkWin();
        if (winFlag == true) {
            message = "游戏结束,"
                + (allChess[x][y] == 1 ? "黑方" : "白方")
                + "胜";
            sendData("over|" + message);
            JOptionPane.showMessageDialog(null, message);
            System.out.println(message);
            canPlay = false;
        }
        } else {
            message = "当前位置已经有棋子, 请重新落子! ";
            System.out.println(message);
        }
    }
    repaint();
} else {
    message = "该对方走棋! ";
    JOptionPane.showMessageDialog(null, message);
}
}
}

```

checkWin()方法判断这个棋子是否和其他的棋子连成 5 子即输赢判断。它是以 (x,y) 为中心横向、纵向、斜方向的判断来统计相同个数来实现的。代码如下:

```

private boolean checkWin() {
    boolean flag = false;
    //保存共有多少相同颜色的棋子相连
    int count = 1;
    //判断横向是否有 5 个棋子相连, 特点是纵坐标相同
    //即 allChess[x][y] 中 y 值相同
    int color = allChess[x][y];
    //通过循环来判断棋子是否相连

```



```

//横向的判断
int i = 1;
while (color == allChess[x + i][y + 0]) {
    count++;
    i++;
}
i = 1;
while (color == allChess[x - i][y - 0]) {
    count++;
    i++;
}
if (count >= 5) {
    flag = true;
}
//纵向的判断
int i2 = 1;
int count2 = 1;
while (color == allChess[x + 0][y + i2]) {
    count2++;
    i2++;
}
i2 = 1;
while (color == allChess[x - 0][y - i2]) {
    count2++;
    i2++;
}
if (count2 >= 5) {
    flag = true;
}
//斜方向的判断(右上 + 左下)
int i3 = 1;
int count3 = 1;
while (color == allChess[x + i3][y - i3]) {
    count3++;
    i3++;
}
i3 = 1;
while (color == allChess[x - i3][y + i3]) {
    count3++;
    i3++;
}
if (count3 >= 5) {
    flag = true;
}
//斜方向的判断(右下 + 左上)
int i4 = 1;
int count4 = 1;
while (color == allChess[x + i4][y + i4]) {
    count4++;
    i4++;
}
i4 = 1;
while (color == allChess[x - i4][y - i4]) {
    count4++;
    i4++;
}

```



```

    }
    if (count4 >= 5) {
        flag = true;
    }
    return flag;
}

```

15.4.2 设计客户端类 (Client.java)

编写一个继承 JFrame 类的 Client 窗体类，用于完成游戏客户端的各种操作。

导入包及相关类：

```

import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;
import java.io.*;
import java.net.*;

```

Server 窗体类实现鼠标侦听接口，并定义一些成员变量。代码如下：

```

public class Client extends JFrame implements ActionListener {
    JPanel contentPane;
    JLabel jLabel1 = new JLabel();
    JTextField jTextField1 = new JTextField("127.0.0.1");
    JLabel jLabel2 = new JLabel();
    JTextField jTextField2 = new JTextField("4700");
    JButton jButton1 = new JButton(); // “连接”按钮
    JLabel jLabel3 = new JLabel();
    JTextField jTextField3 = new JTextField();
    JButton jButton2 = new JButton(); // “发送”按钮
    JButton jButton3 = new JButton(); // “悔棋”按钮
    JScrollPane jScrollPane1 = new JScrollPane();
    JTextArea jTextArea1 = new JTextArea();
    BufferedReader instr = null;
    Socket socket = null;
    PrintWriter os = null;
    public static String[] ss = new String[10];
    //保存棋子的坐标
    int x = 0;
    int y = 0;
}

```

双方的落子信息保存在二维数组 allChess[19][19]中，其中 0 表示这个位置没有棋子；1 表示这个位置是黑子；2 表示这个位置是白子。由于客户端作为白方因此 isBlack = false。代码如下：

```

int[][] allChess = new int[19][19]; //保存之前下过的全部棋子的坐标
boolean isBlack = false; //自己是白棋方
//标识当前游戏是否可以继续
boolean canPlay = false;
//保存显示的提示信息
String message = ""; // “自己是白棋，黑方先行”；
JPanel panel1 = new JPanel();
GobangPanel panel2 = new GobangPanel();

```

在 Client 窗体类的构造方法中，添加动作监听器，设置窗体中各个组件的位置，主要有两个 Panel 组成，一个 Panel 中放置“连接”按钮 jButton1、“发送”按钮 jButton2、“悔棋”按钮 jButton3

及显示聊天内容的文本区域 `jTextArea1`。另一个 `GobangPanel` 实例 `panel2` 中, 放置棋盘背景图片, 并能接收鼠标单击事件。在窗口关闭事件中向对方发送离开信息 `quit`。代码如下:

```
/**
 * 客户端构造方法
 */
public Client() {
    jbInit();
}

private void jbInit() {
    contentPane = (JPanel) this.getContentPane();
    jLabel1.setFont(new java.awt.Font("宋体", 0, 14));
    jLabel1.setText("服务器名称");
    jLabel1.setBounds(new Rectangle(20, 22, 87, 28));
    // contentPane.setLayout(null);
    this.setSize(new Dimension(540, 640));
    this.setTitle("客户端");
    jTextField1.setBounds(new Rectangle(114, 26, 108, 24));
    jLabel2.setText("端口号");
    jLabel2.setFont(new java.awt.Font("宋体", 0, 14));
    jButton1.setBounds(new Rectangle(400, 28, 73, 25));
    jButton1.setFont(new java.awt.Font("Dialog", 0, 14));
    jButton1.setBorder(BorderFactory.createEtchedBorder());
    jButton1.setActionCommand("jButton1");
    jButton1.setText("连接");
    jLabel3.setBounds(new Rectangle(23, 57, 87, 28));
    jLabel3.setText("请输入信息");
    jLabel3.setFont(new java.awt.Font("宋体", 0, 14));
    jTextField3.setBounds(new Rectangle(114, 60, 314, 24));

    jButton2.setText("发送");
    jButton2.setActionCommand("jButton1");
    jButton2.setBorder(BorderFactory.createEtchedBorder());
    jButton2.setFont(new java.awt.Font("Dialog", 0, 14));
    jButton2.setBounds(new Rectangle(440, 58, 73, 25));
    jButton3.setText("悔棋");
    jButton3.setActionCommand("jButton1");
    jButton3.setBorder(BorderFactory.createEtchedBorder());
    jButton3.setFont(new java.awt.Font("Dialog", 0, 14));
    jButton3.setBounds(new Rectangle(480, 28, 43, 25));
    jScrollPane1.setBounds(new Rectangle(23, 85, 493, 69));
    jTextField3.setText("此处输入发送信息 ");
    jTextArea1.setText("");
    panel1.setLayout(null);
    panel1.add(jLabel1);
    panel1.add(jTextField1);
    panel1.add(jLabel2);
    panel1.add(jTextField2);
    panel1.add(jButton1);
    panel1.add(jLabel3);
    panel1.add(jTextField3);
    panel1.add(jButton2);
    panel1.add(jButton3);
}
```



```

panel1.add(jScrollPane1);
jScrollPane1.getViewport().add(jTextArea1);
contentPane.setLayout(null);
contentPane.add(panel1);
contentPane.add(panel2);
panel1.setBounds(0, 0, 540, 160);
panel2.setBounds(10, 160, 540, 460);
jButton1.addActionListener(this);
jButton2.addActionListener(this);
jButton3.addActionListener(this);
this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        try {
            sendData("quit|"); //向对方发送离开信息
            socket.close();
            instr.close();
            os.close();
            System.exit(0);
        } catch (Exception ex) {
        }
    }
});
}

```

在 `actionPerformed()` 是一个方法，在表示对象发生操作时调用。通过 `e.getSource()` 来区别哪个按钮发生了单击事件，如果是“连接”按钮 `jButton1`，则调用 `connectServer(ip, port)` 实现与服务器的连接；如果是“发送”按钮 `jButton2`，则调用 `sendData()` 方法发送文字；如果是“悔棋”按钮 `jButton3`，则发送悔棋信息。代码如下：

```

public void actionPerformed(ActionEvent e) {
    if (e.getSource() == jButton1) { //连接按钮
        String ip = jTextField1.getText().trim();
        int port = Integer.parseInt(jTextField2.getText().trim());
        connectServer(ip, port);
    }
    if (e.getSource() == jButton2) { //“发送”按钮
        String s = this(jTextField3.getText().trim());
        sendData(s);
    }
    if (e.getSource() == jButton3) { //“悔棋”按钮
        if (canPlay != true) { //该对方走棋
            allChess[x][y] = 0;
            panel2.repaint();
            canPlay = true;
            String s = "undo|" + x + "|" + y;
            sendData(s);
            System.out.print("发送悔棋信息");
        } else //对方已走棋
        {
            message = "对方已走棋，不能悔棋了";
            JOptionPane.showMessageDialog(this, message);
            System.out.print("对方已走棋，不能悔棋了");
        }
    }
}
}

```


“连接”按钮 jButton1 从文本框得到服务器 ip 及端口 port, 调用 connectServer(ip, port) 方法启动与服务器的连接, 同时修改文字为“连接服务器...”, 程序阻塞直到 Socket 连接成功。连接成功后“连接”按钮 jButton1 文字改为“正在聊天”。由于客户端作为白方, 因此棋盘上的信息提示为“自己是白棋, 黑方先行”。最后启动接收聊天信息及下棋信息的线程 MyThread t, 线程 t 不断接收相关信息。代码如下:

```
private void connectServer(String ip, int port) { // 连接
    try {
        if (jButton1.getText().trim().equals("连接")) {
            jButton1.setText("连接服务器...");
            socket = new Socket(ip, port);
            this.setTitle("你是白方");
            jButton1.setText("正在聊天");
            message = "自己是白棋, 黑方先行";
            panel2.repaint();
            MyThread t = new MyThread();
            t.start();
        }
    } catch (Exception ex) {
    }
}
```

内部线程类 MyThread 与服务器端代码完全一样, 该线程负责接收数据。当接收到数据时区分是何种数据, 如果是“move|x|y”信息, 则是对方(黑方)走棋信息, 修改记录棋盘信息的 allChess 数组, 并刷新自己屏幕。如果是“undo|x|y”信息, 则是对方(黑方)撤销上步棋信息, 修改记录棋盘信息的 allChess 数组, 并刷新自己屏幕。如果是“over”信息, 则是游戏一方胜利了, 对话框显示输赢信息。如果是“quit”信息, 则是对方离开了, 游戏结束。其代码与服务器端此部分代码一样, 如下所示:

//内部线程类

```
class MyThread extends Thread {
    public void run() {
        try {
            os = new PrintWriter(socket.getOutputStream());
            instr = new BufferedReader(new InputStreamReader(socket
                .getInputStream()));
            while (true) {
                this.sleep(100);
                if (instr.ready()) {
                    String cmd = instr.readLine();
                    //在每个空格字符处进行分解
                    ss = cmd.split("\\|");
                    if (cmd.startsWith("move")) { //对方黑子走棋信息
                        int x = Integer.parseInt(ss[1]);
                        int y = Integer.parseInt(ss[2]);
                        allChess[x][y] = 1; //黑子落在 (x, y) 位置
                        message="轮到自己下棋子";
                        panel2.repaint();
                        canPlay = true;
                    }
                    if (cmd.startsWith("undo")) {
```



```

        JOptionPane.showMessageDialog(null, "对方撤销上步棋");
        int x = Integer.parseInt(ss[1]);
        int y = Integer.parseInt(ss[2]);
        allChess[x][y] = 0;
        panel2.repaint();
        canPlay = false;
    }
    if (cmd.startsWith("over")) {
        JOptionPane.showMessageDialog(null, "游戏结束, 对方胜!!");
        panel2.setEnabled(false);
        canPlay = false;
    }
    if (cmd.startsWith("quit")) {
        JOptionPane.showMessageDialog(null, "游戏结束, 对方离开了!!");
        panel2.setEnabled(false);
        canPlay = false;
    }
    if (cmd.startsWith("chat")) {
        JTextAreal.append("服务器端说: " + ss[1] + "\n");
    }
}
} catch (Exception ex) {
    System.out.print("error: " + ex);
}
}
}

```

sendData(String s)方法负责发送数据给客户端。代码如下:

```

private void sendData(String s) { //发送数据
    try {
        os = new PrintWriter(socket.getOutputStream());
        os.println(s);
        os.flush();
        this.jTextAreal.append("Client:" + s + "\n");
    } catch (Exception ex) {
    }
}

```

main()方法实例化 Client 类窗体并显示, 代码如下:

```

public static void main(String arg[]) {
    JFrame.setDefaultLookAndFeelDecorated(true);
    Client frm = new Client();
    frm.setVisible(true);
}

```

内部类 GobangPanel 继承面板 JPanel, 首先显示棋盘背景图片。在面板的刷新方法 paintComponent(Graphics g)中采用双缓冲技术防止屏幕闪烁。双缓冲技术就是将要显示的信息首先绘制到缓冲图片 BufferedImage bi 中, 绘制完成后将此图片一次显示到屏幕上, 这样可以避免屏幕闪烁。其代码与服务器端此部分代码一样。

```

class GobangPanel extends JPanel {
    ...//见服务器端此部分代码
}

```

内部类 MouseLis 继承 MouseAdapter 适配器, 实现 mousePressed(MouseEvent e)事件。在此事

件中首先判断单击位置是否在棋盘中，并且此处无棋子，则根据自己是哪方颜色来修改棋盘对应的数组 allChess 此处 allChess [x][y]元素的值。落子后刷新自己的屏幕，并调用自己类中的方法 checkWin()判断这个棋子是否和其他的棋子连成 5 子即输赢判断，如果赢了则发送游戏结束信息给对方。其代码与服务器端此部分代码一样。

```
class MouseLis extends MouseAdapter {  
    public void mousePressed(MouseEvent e) {  
        ...//见服务器端此部分代码  
    }  
}
```

checkWin()方法判断这个棋子是否和其他的棋子连成 5 子即输赢判断。它是以 (x,y) 为中心横向、纵向、斜方向的判断（右下+左上）来统计相同个数来实现的。其代码与服务器端此部分代码一样。

```
private boolean checkWin() {  
    ...//见服务器端此部分代码  
}
```

当一方连成 5 子时，游戏结束的界面如图 15-6 所示。

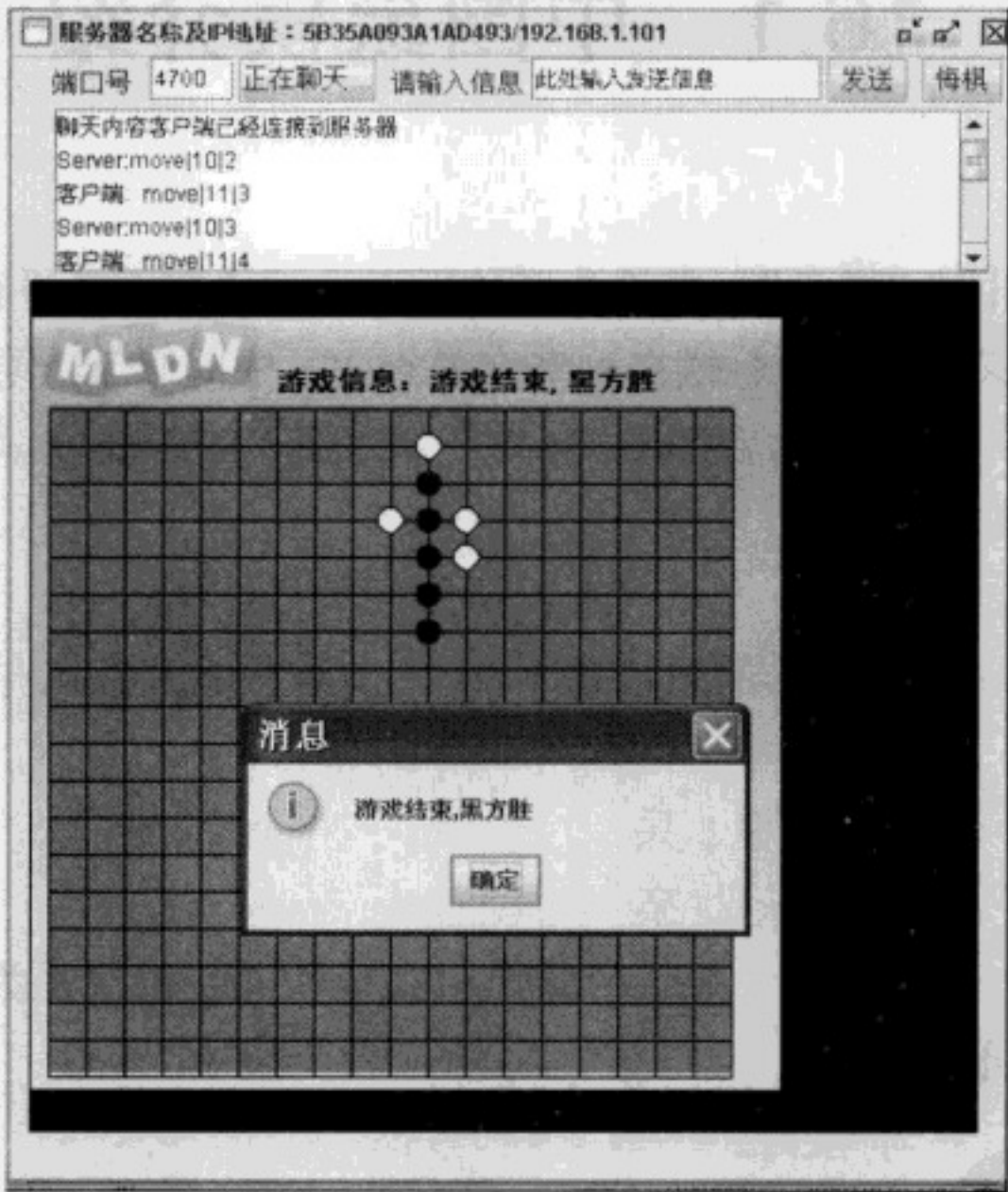


图 15-6 游戏结束的界面

第 16 章

网络中国象棋

中国象棋是一种家喻户晓的棋类游戏，它的趣味性吸引了无数的玩家。在信息化的今天，我们能否凭借革新精神，在电脑中下中国象棋呢？下面介绍“基于 UDP 的 P2P 网络中国象棋”的开发原理和过程。

16.1 中国象棋介绍

1. 棋盘

棋子活动的场所，叫作“棋盘”，在长方形的平面上，绘有 9 条平行的竖线和 10 条平行的横线相交组成，共 90 个交叉点，棋子就摆在这些交叉点上。中间第五、第六两条横线之间未画竖线的空白地带，称为“河界”，整个棋盘就以“河界”分为相等的两部分。两方将帅坐镇，画有“米”字方格的地方，叫作“九宫”。

2. 棋子

象棋的棋子共 32 个，分为红黑两组，各 16 个，由对弈双方各执一组，每组兵种是一样的，各分为 7 种：

红方：帅、仕、相、车、马、炮、兵

黑方：将、士、象、车、马、炮、卒

其中帅与将、仕与士、相与象、兵与卒的作用完全相同，仅仅是为了区分红棋和黑棋。

3. 各棋子的走法说明

(1) 将或帅

移动范围：它只能在王宫内移动。

移动规则：它每一步只可以水平或垂直移动一点。

(2) 士或仕

移动范围：它只能在王宫内移动。

移动规则：它每一步只可以沿对角线方向移动一点。

(3) 象或相

移动范围：河界的一侧。

移动规则：它每一步只可以沿对角线方向移动两点，另外，在移动的过程中不能穿越障碍。

(4) 马

移动范围：任何位置。

移动规则：每一步只可以水平或垂直移动一点，再按对角线方向向左或者向右移动。另外，在移动的过程中不能穿越障碍。

(5) 车

移动范围：任何位置。

移动规则：可以水平或垂直方向移动任意个无阻碍的点。

(6) 炮

移动范围：任何位置。

移动规则：移动起来与车很相似，但它必须跳过一个棋子才能吃掉对方的一个棋子。

(7) 兵或卒

移动范围：任何位置。

移动规则：每步只能向前移动一点。过河以后，它便增加了向左右移动的能力，不允许向后移动。

4. 关于输和赢

对局中，出现下列情况之一，本方算输，对方获胜：

(1) 己方的帅（将）被对方棋子吃掉；

(2) 己方发出认输请求；

(3) 己方走棋超出步时限制。

16.2 关键技术

16.2.1 UDP 简介

UDP 的全称是用户数据报 (User Datagram Protocol)，在网络中它与 TCP 一样用于处理数据报。在 OSI 模型中，UDP 位于第 4 层——传输层，处于 IP 协议的上一层。UDP 具有不提供数据报分组、组装以及不能对数据报排序的缺点。也就是说，当报文发送之后，是无法得知其是否安全完整到达的。

在选择使用协议的时候，选择 UDP 必须要谨慎。在网络质量令人不十分满意的环境下，UDP 的数据包丢失会比较严重。但是由于 UDP 不属于连接型协议的特性，具有资源消耗小，处理速度快的优点，因此通常音频、视频和普通数据在传送时使用 UDP 较多，因为它们即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。例如我们聊天用的 ICQ 和 OICQ 就是使用的 UDP。

使用 java.net 包下的 DatagramSocket 和 DatagramPacket 类，可以非常方便地控制用户的数据报文，下面就对这两个类进行介绍。

16.2.2 DatagramPacket 类

DatagramPacket 类用于处理报文，它将 Byte 数组、目标地址和目标端口等数据包装成报文或者将报文拆卸成 Byte 数组。应用程序在产生数据报时应该注意，TCP/IP 规定数据报文大小最多包含 65 507 个，通常主机接收 548 个字节，但大多数平台能够支持 8 192 字节大小的报文。

DatagramPacket 有多个构造方法，尽管这些构造方法的形式不同，但通常情况下它们都有两个共同的参数：byte [] buf 和 int length。其中参数 buf 包含了一个对保存自寻址数据报信息的字

节数组的引用，参数 `length` 表示字节数组的长度。

最简单的构造方法是：

```
DatagramPacket(byte [ ] buf, int length);
```

这个构造方法确定了数据报数组和数组的长度，但没有任何数据报的地址和端口信息，这些信息可以通过调用方法 `setAddress (InetAddress addr)` 和 `setPort (int port)` 添加上。下面的代码示范了这些函数和方法。

```
byte [ ] buffer = new byte [100];
DatagramPacket dgp = new DatagramPacket (buffer, buffer.length);
InetAddress ia = InetAddress.getByName ("www.disney.com");
dgp.setAddress (ia);
dgp.setPort (6000); //送数据报到端口 6000
```

如果用户更喜欢在调用构造方法的同时包括地址和端口号，则可以使用

```
DatagramPacket (byte [ ] buf, int length, InetAddress addr, int port);
```

下面的代码示范了另外一种选择。

```
byte [ ] buffer = new byte [100];
InetAddress ia = InetAddress.getByName ("www.disney.com");
DatagramPacket dgp = new DatagramPacket (buffer, buffer.length, ia, 6000);
```

有时候在创建了 `DatagramPacket` 对象后想改变字节数组和它的长度，这时可以通过调用下面两个方法来实现。

```
setData(byte [ ] buf);
setLength(int length);
```

在任何时候都可以通过调用 `getData()` 方法来得到字节数组的引用；通过调用 `getLength()` 方法来获得字节数组的长度。下面的代码示范了这些方法。

```
byte [ ] buffer2 = new byte [256];
dgp.setData (buffer2);
dgp.setLength (buffer2.length);
```

`DatagramPacket` 的常用方法如下。

`getAddress ()`、`setAddress (InetAddress)`：得到、设置数据报地址。

`getDate ()`、`setData (byte [] buf)`：得到、设置数据报内容。

`getLength ()`、`setLength (int length)`：得到、设置数据报长度。

`getPort ()`、`setPort (int port)`：得到、设置端口号。

16.2.3 DatagramSocket 类

`DatagramSocket` 类在客户端创建数据报套接字与服务器端进行通信连接，并发送和接收数据报套接字。虽然有多个构造方法可供选择，但创建客户端套接字最便利的选择是 `DatagramSocket ()` 函数，而服务器端则是 `DatagramSocket (int port)` 函数。如果未能创建套接字或绑定套接字到本地端口，那么这两个函数都将抛出一个 `SocketException` 对象。一旦程序创建了 `DatagramSocket` 对象，那么程序将分别调用 `send (DatagramPacket p)` 和 `receive (DatagramPacket p)` 来发送和接收数据报。

`Datagram` 构造方法如下。

`DatagramSocket ()`：创建数据报套接字，绑定到本地主机任意存在的端口。

`DatagramSocket (int port)`：创建数据报套接字，绑定到本地主机指定端口。

`DatagramSocket (int port, InetAddress laddr)`: 创建数据报套接字, 绑定到指定的本地主机地址。

常用方法如下。

`connect (InetAddress address, int port)`: 连接指定地址。

`disconnect ()`: 断开套接字连接。

`close ()`: 关闭数据报套接字。

`getInetAddress ()`: 得到套接字所连接的地址。

`getLocalAddress ()`: 得到套接字绑定的主机地址。

`getLocalPort ()`: 得到套接字绑定的主机端口号。

`getPort ()`: 得到套接字的端口号。

`reseive (DatagramPacket p)`: 接收数据报。

`send (DatagramPacket p)`: 发送数据报。

下面的例子示范了如何创建数据报套接字以及如何通过套接字处理发送和接收信息。例 1 是数据报套接字客户机示例的程序 `DatagramDemo.java`。

```
import java.io.*;
import java.net.*;
class DatagramDemo{//发送数据端
    public static void main (String [] args){
        String host = "localhost";
        DatagramSocket s = null;
        try{
            s = new DatagramSocket ();
            byte [] buffer;
            buffer = new String ("Send me a datagram").getBytes ();
            InetAddress ia = InetAddress.getByName (host);
            DatagramPacket dgp = new DatagramPacket (buffer, buffer.length, ia, 10000);
            s.send (dgp);
            byte [] buffer2 = new byte [100];
            dgp = new DatagramPacket (buffer2, buffer.length, ia, 10000);
            s.receive (dgp);
            System.out.println (new String (dgp.getData ()));
        }
        catch (IOException e){
            System.out.println (e.toString ());
        }
        finally{
            if (s != null)
                s.close ();
        }
    }
}
```

`DatagramDemo` 由创建一个绑定任意本地 (客户端) 端口号的 `DatagramSocket` 对象开始, 然后装入带有文本信息的数组 `buffer` 和描述服务器主机 IP 地址的 `InetAddress` 子类对象的引用, 接下来, 程序创建了一个 `DatagramPacket` 对象, 该对象加入了带文本信息的缓冲器的引用、`InetAddress` 子类对象的引用, 以及服务端口号 10000。`DatagramPacket` 的数据报通过方法 `sent()` 发送给服务器程序, 于是一个包含服务程序响应的新的 `DatagramPacket` 对象被创建, `receive()` 方法得到相应的数据报, 然后由 `getData()` 方法返回该数据报的一个引用, 最后关闭 `Datagram Socket`。

例 2 是相应的服务程序。

例 2 数据报套接字服务器程序示例。

```
import java.io.*;
import java.net.*;

class DatagramServerDemo{           //接收数据端
    public static void main (String [] args) throws IOException{
        System.out.println ("Server starting ...\n");
        DatagramSocket s = new DatagramSocket (10000);
        byte [] data = new byte [100];
        DatagramPacket dgp = new DatagramPacket (data, data.length);
        //进入一个无限循环中来接收数据包
        while (true){
            s.receive (dgp);           //接收数据包
            System.out.println (new String (data));
            s.send (dgp);
        }
    }
}
```

该程序创建了一个绑定端口 10000 的数据报套接字,然后创建一个字节数组容纳数据报信息,并创建数据报包。接着,程序进入一个无限循环以接收自寻址数据包、显示内容并将响应返回客户端,套接不会关闭,因为循环是无限的。

在编译 DatagramServerDemo 和 DatagramDemo 的源代码后,由输入 java DatagramServerDemo 开始运行 DatagramServerDemo,然后在同一主机上输入 java DatagramDemo 开始运行 DatagramDemo,如果 DatagramServerDemo 与 DatagramDemo 运行于不同主机,在输入时注意要在命令行加上服务程序的主机名或 IP 地址,例如,IP 地址 202.196.32.97 或主机名 DatagramDemo www.yesky.com。

16.2.4 P2P 知识

Peer-To-Peer 即点对点,通常简称为 P2P。所谓网络中的点对点,其实可以看成是一种对等的网络模型。P2P 其实是实现网络上不同计算机之间,不经过中继设备直接交换数据或服务的一种技术。P2P 由于允许网络中任一台计算机可以直接连接到网络中的其他计算机,并与之进行数据交换,这样既消除了中间环节,也使得网络上的沟通变得更加容易、更直接。

P2P 作为一种网络的模型,它有别于传统的客户/服务器模型。客户/服务器模型一般都有预定义的客户机和服务器。而在 P2P 模型中并没有明确的客户端和服务端,但其实在 P2P 模型中,每一台计算机既可以看成是服务器,也可以看成是客户机。在网络中,传统上的客户机/服务器通信模型中,发送服务请求或者发送数据的计算机,一般称为客户机;而接收、处理服务或接收数据的计算机称为服务器。而在 P2P 网络模型中,计算机不仅接收数据,还发送数据;不仅提出服务请求,还接收对方的服务请求。

P2P 是一种用于不同 PC 用户之间,不经过中继设备直接交换数据或服务的技术,它允许 Internet 用户直接使用对方的文件。每个人可以直接连接到其他用户的计算机,并进行文件的交换,而不需要连接到服务器上再进行浏览与下载。因为消除了中间环节,P2P 技术使得网络上的沟通变得更加容易、更直接。P2P 改变了 Internet 现在的以大网站为中心的状态,重返“非中心化”,并把权力交还给用户。

16.3 网络中国象棋设计思路

16.3.1 棋盘表示

棋盘表示就是使用一种数据结构来描述棋盘及棋盘上的棋子，我们使用一个二维数组 Map。一个典型的中国象棋棋盘是使用 9×10 的二维数组来表示的。每一个元素代表棋盘上的一个交点。一个没有棋子的交点所对应的元素是-1。一个二维数组 Map 保存了当前棋盘的布局。当 $\text{Map}[x,y]=i$ 时说明此处是棋子 i，否则为-1 说明此处为空。程序中下棋的棋盘界面使用图 16-1 所示的图片资源。

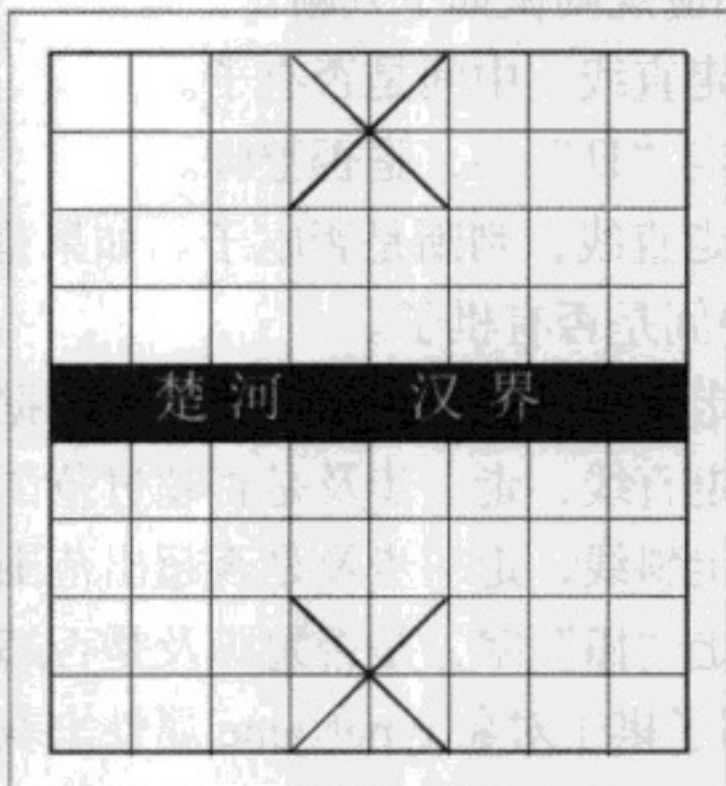


图 16-1 棋盘图片资源

16.3.2 棋子表示

为棋子设计相应的类，每种棋子图案用到的对应的图片资源如图 16-2 所示。



图 16-2 棋子图片资源

为了使设计程序方便，将 32 个棋子对象赋给了数组 chess。chess[i]中下标 i 的含义是：如果 i 小于 16，那么说明它属于黑方的棋子，否则是红方的棋子。这样就可以得到我们所需要的全部象棋棋子了。黑方棋子对应的是 0~15，红方棋子则用 16~31 表示。

具体下标含义如下。

0 将 1 士 2 士 3 象 4 象 5 马 6 马 7 车 8 车 9 炮 10 炮 11~15 卒
16 帅 17 士 18 士 19 象 20 象 21 马 22 马 23 车 24 车 25 炮 26 炮 27~31 兵

游戏开始时根据不同角色（红方或黑方）初始化棋盘。注意初始化棋盘时，无论游戏者是红方还是黑方，必须保证自己的棋子在下方（南边），对方的棋子在上方，这样才能使游戏者进入以正常角度看的游戏。而对方看到时在上方，这需要在传递棋子信息时，将棋子信息对调成对方角度的棋子。

16.3.3 走棋规则

对于象棋来说，有“马走日”、“象走田”等一系列复杂的规则。走法的产生是博弈程序中一个相当复杂而且耗费运算时间的方面。不过，通过良好的数据结构，可以显著地提高生成的速度。判断是否能走棋的算法如下。

根据棋子名称的不同，按相应规则做如下判断。

- 如果为“车”，检查是否走直线，中间是否有子。
- 如果为“马”，检查是否走“日”字，是否蹩脚。
- 如果为“炮”，检查是否走直线，判断是否吃子。如果是吃子，则检查中间是否只有一个棋子；如果不是吃子，则检查中间是否有棋子。
- 如果为“兵”，检查是否走直线，走一步及向前走，根据是否过河，检查是否横走。
- 如果为“将”，检查是否走直线，走一步及是否超过范围。
- 如果为“士”，检查是否走斜线，走一步及是否超出范围。
- 如果为“象”，检查是否走“田”字，是否蹩脚及是否超出范围。

如何分辨棋子？程序中采用了棋子对象 `typeName` 属性来获取。

程序中 `IsAbleToPut(firstchess, x, y)` 函数实现判断是否能走棋，返回逻辑值，这段代码最复杂。其中参数含义如下。

`firstchess` 代表走的棋子对象，参数 `x, y` 代表走棋的目标位置。走动棋子原始位置 (`oldx, oldy`) 可以通过 `firstchess.pos.x` 获取原 `x` 坐标 `oldx`，`firstchess.pos.y` 获取原 `y` 坐标 `oldy`。

`IsAbleToPut(idcx, x, y)` 函数实现走棋规则判断。

例如“将”或“帅”的走棋规则为只能走一格，所以原 `x` 坐标与新位置 `x` 坐标之差不能大于 1，`GetChessY(idcx)` 获取的原 `y` 坐标与新位置 `y` 坐标之差不能大于 1。

```
if (Math.Abs(x - oldx) > 1 || Math.Abs(y - oldy) > 1)
    return false;
```

由于不能走出九宫，因此 `x` 坐标为 4、5、6 且 $1 \leq y \leq 3$ 或 $8 \leq y \leq 10$ （实际上仅需判断是否 $8 \leq y \leq 10$ 即可，因为走棋时自己的“将”或“帅”只能在下方的九宫中），否则此步违规，将返回 `False`。

```
if (x < 4 || x > 6 || (y > 3 && y < 8)) return false;
```

“士”的走棋规则为只能走斜线一格，因此原 `x` 坐标与新位置 `x` 坐标之差为 1 且原 `y` 坐标与新位置 `y` 坐标之差也同时为 1。

```
if ((x - oldx) * (y - oldy) == 0) return false;
if (Math.Abs(x - oldx) > 1 || Math.Abs(y - oldy) > 1) return false;
```

由于不能走出九宫，因此 `x` 坐标为 4、5、6 且 $1 \leq y \leq 3$ 或 $8 \leq y \leq 10$ ，否则此步违规，将返回 `False`。

```
if (x < 4 || x > 6 || (y > 3 && y < 8)) return false;
```


“炮”的走棋规则为只能走直线，因此 x, y 不能同时改变，即 $(x - oldx) \times (y - oldy) = 0$ 保证走直线。然后判断如果 x 坐标改变了，是否原位置 $oldx$ 到目标位置其间是否有棋子，如果有子则累加其间棋子个数 c 。通过 c 是否为 1 且目标处为非己方棋子，可以判断是否可以走棋。

“兵”或“卒”的走棋规则为只能向前走一步，根据是否过河，检查是否可以横走。因此 x 与原坐标 $oldx$ 改变的值不能大于 1，同时 y 与原坐标 $oldy$ 改变的值也不能大于 1。如果过河即是 $y < 6$ 。代码如下：

```
if ((x - oldx) * (y - oldy) != 0)
    return false;
if (Math.Abs(x - oldx) > 1 || Math.Abs(y - oldy) > 1)
    return false;
if (y >= 6 && (x - oldx) != 0)
    return false;
if (y - oldy > 0)
    return false;
return true;
```

其余的棋子判断方法类似，这里不再一一介绍。

16.3.4 坐标转换

走棋过程中，需要将鼠标像素坐标转换成棋盘坐标，用到 `analyse()` 方法，解析出棋盘坐标 (`tempx, tempy`)；如果单击处有棋子则返回棋子对象，如果无棋子将返回 `null`。代码如下：

```
//解析鼠标之下的棋子对象
private Chess analyse(int x, int y)
{
    tempx = (int)Math.floor((double)x / 40)+1;
    tempy = (int)Math.floor((double)(y-20) / 40)+1;
    //防止超出范围
    if (tempx > 9 || tempy > 10 || tempx < 1 || tempy < 1)
    {
        return null;
    }
    else
    {
        int idx = Map[tempx][tempy];
        if (idx == -1)
        {
            return null;
        }
        return chess[idx];
    }
}
```

16.3.5 通信协议设计

网络程序设计的难点在于与对方需要通信。这里使用了 UDP (User Data Protocol)。UDP 是用户数据报文协议的简称，两台计算机之间的传输类似于传递邮件；两台之间没有明确的连接，使用 UDP 建立对等通信。这里虽然两台计算机不分主次，但我们设计时假设一台做主机（红方），等待其他人加入。其他人加入的时候需要输入主机的 IP。为了区分通信中传送的是“输赢信息”、“下的棋子位置信息”、“重新开始”等信息，在发送信息的首部加上了代号。因此定义了如下协议。

命令|参数|参数……

(1) 联机功能

join|

用户如果联机则发此命令，并处于不断接收对方联机的状态。

(2) 联机成功信息

conn|

收到对方联机命令，发出此联机成功信息。

(3) 认输信息

lose|

如果游戏者发出认输信息，则发此命令。

(4) 对方棋子移动信息

move| idx, x, y

其中，棋子移动的目标位置坐标 (x,y)，棋子移动的起始位置坐标 (old_x,old_y) 可以从棋盘布局 Map 中获取；idx 是被移动棋子的数组索引号。

注意本程序在传递棋子移动信息时，这里采取了一个小技巧，在发送数据的时候我们把坐标做了颠倒（把自己的棋盘颠倒）。即 (x,y) 坐标以 ((10-x), (11-y)) 的坐标形式发给对方。

(5) 游戏结束

succ|+赢方代号（赢了此局）

(6) 表示退出游戏

quit|

16.3.6 网络通信传递棋子信息

游戏开始后，创建一个线程 th:

```
th = new Thread(this);
```

// 创建一个线程 th

```
flag = true;
```

th.Start()方法启动线程后，通过重写 run()方法实现不断侦听本机设定的端口，得到对方发送过来的信息，根据自己定义的通信协议中传送的是“输赢信息”、“下的棋子位置信息”、“重新开始”等信息而分别进行处理。

```
public void run() {
    try {
        //指定接收端口
        DatagramSocket s = new DatagramSocket (receiveport);
        byte [] data = new byte [100];
        DatagramPacket dgp = new DatagramPacket (data, data.length);
        //进入一个无限循环中来接收数据包
        while (flag==true){
            s.receive (dgp);           //接收数据包
            String strData=new String (data);
            String[] a = new String[6];
            a = strData.split("\\|");
            System.out.println("接收数据信息:"+strData+"分割命令: "+a[0]);
            if (a[0].equals("join"))
                //与对方联机
            else if (a[0].equals("conn"))
                //联机成功信息
```



```
else if (a[0].equals("succ"))
    //输赢信息
else if (a[0].equals("move"))
    //对方的走棋信息, move|棋子索引号|X|Y
else if (a[0].equals("quit"))
    //对方退出
else if (a[0].equals("lose"))
    //对方认输
}
} catch (SocketException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}
```

掌握以上关键技术，就可以开发两人对战的网络中国象棋程序了。

16.4 网络中国象棋实现的步骤

16.4.1 设计棋子类（chess.java）

棋子的类图如图 16-3 所示。棋子类代码中首先定义棋子所属玩家、坐标位置、棋子图案、棋子种类成员变量。代码如下：

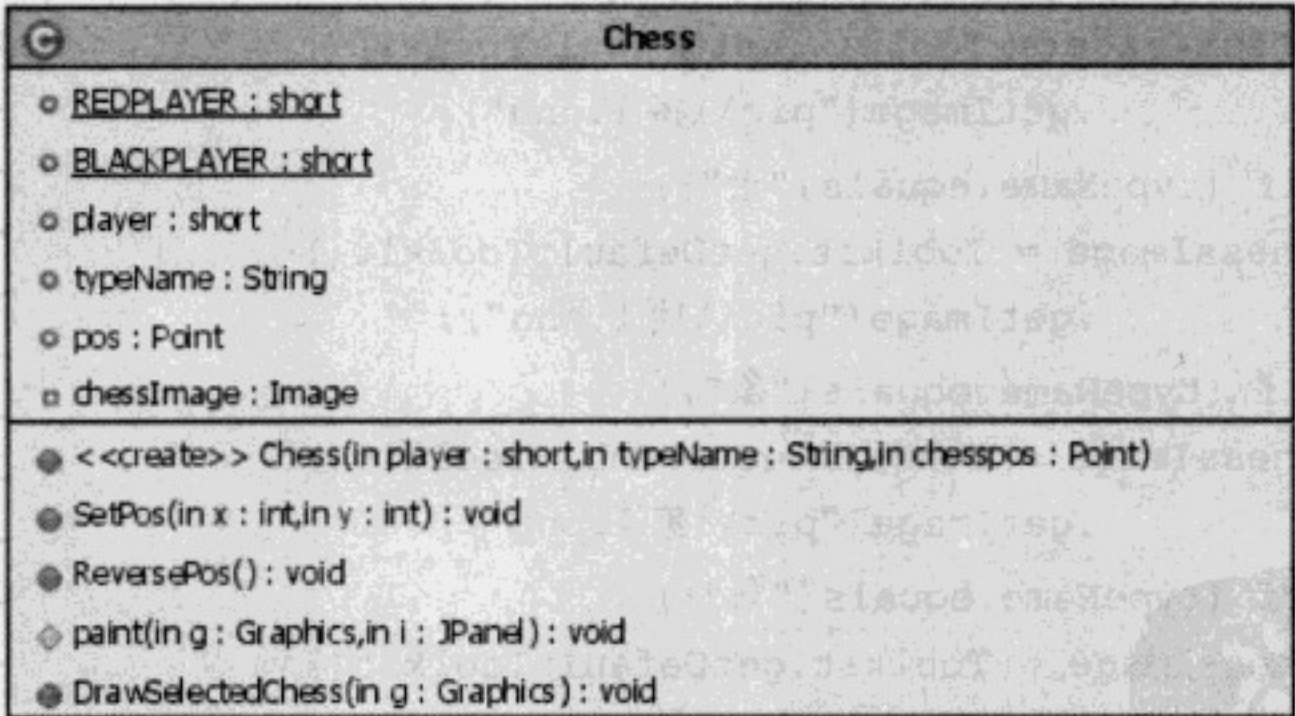


图 16-3 棋子的类图

```
class Chess // 棋子类
{
    public static final short REDPLAYER = 1;
    public static final short BLACKPLAYER = 0;
    public short player; // 红子为 REDPLAYER，黑子为 BLACKPLAYER
    public String typeName; // 帅、士……
    public Point pos; // 位置
    private Image chessImage; // 棋子图案
}
```


棋子类构造方法的 3 个参数分别代表哪方、棋子名称、棋子所在棋盘位置。该构造方法中，多分支 if 语句段根据棋子种类设置相应棋子的图案。代码如下：

```
public Chess(short player, String typeName, Point chesspos) {
    this.player = player;
    this.typeName = typeName;
    this.pos = chesspos;
    //初始化棋子图案
    //不能使用 switch (typeName) 方法，因为 switch 不支持 String 类型
    if (player == REDPLAYER) {
        if (typeName.equals("帅"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\帅.png");
        else if (typeName.equals("仕"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\士.png");
        else if (typeName.equals("相"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\相.png");
        else if (typeName.equals("马"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\马.png");
        else if (typeName.equals("车"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\车.png");
        else if (typeName.equals("炮"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\炮.png");
        else if (typeName.equals("兵"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\兵.png");
    } else //黑方棋子
    {
        if (typeName.equals("将"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\将 1.png");
        else if (typeName.equals("士"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\仕 1.png");
        else if (typeName.equals("象"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\象 1.png");
        else if (typeName.equals("马"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\马 1.png");
        else if (typeName.equals("车"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\车 1.png");
        else if (typeName.equals("炮"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\炮 1.png");
        else if (typeName.equals("卒"))
            chessImage = Toolkit.getDefaultToolkit().getImage("pic\\卒 1.png");
    }
}
```



```

    }
}

```

SetPos(int x,int y)方法用来设置棋子所在棋盘的位置。ReversePos()方法的作用是将棋子位置坐标颠倒(棋盘颠倒)。即(x,y)坐标变成(10-x, 11-y)坐标。代码如下:

```

public void SetPos(int x, int y) //设置棋子位置
{
    pos.x = x;
    pos.y = y;
}

public void ReversePos() //将棋子位置对调
{
    pos.x = 10 - pos.x;
    pos.y = 11 - pos.y;
}

```

构造方法创建好棋盘后, paint()方法绘制棋子到传递过来的棋盘游戏面板上。代码如下:

```

// 在指定的 JPanel 上画棋子
protected void paint(Graphics g, JPanel i) {
    g.drawImage(chessImage, pos.x * 40 - 40, pos.y * 40 - 20, 40, 40,
        (ImageObserver) i);
}

```

棋子类中提供另一个 DrawSelectedBlock(Graphics g)方法,它将在棋子周围画选中的示意边框线,这里是直接画在传递过来的游戏面板上。代码如下:

```

public void DrawSelectedChess(Graphics g) {
    //画选中棋子的示意边框线
    g.drawRect(pos.x * 40 - 40, pos.y * 40 - 20, 40, 40);
}

```

16.4.2 设计棋盘类 (ChessBoard.java)

棋盘类是游戏实例,首先定义一个数组 chess 存储双方 32 个棋子对象。二维数组 Map 保存了当前棋盘的棋子布局,当 Map[x,y]=i 时说明此处是棋子 i, 否则为-1 说明此处为空。以下是成员变量的定义:

```

public class ChessBoard extends JPanel implements Runnable { //棋盘类
    public static final short REDPLAYER = 1;
    public static final short BLACKPLAYER = 0;
    public Chess[] chess = new Chess[32]; //所有棋子
    public int[][] Map = new int[9 + 1][10 + 1]; //棋盘的棋子布局
    public Image bufferImage; //双缓存
    private Chess firstChess2 = null; //鼠标单击时选定的棋子
    private Chess secondChess2 = null;
    private boolean first = true; //区分第一次与第二次选中的棋子
    private int x1, y1, x2, y2;
    private int tempx, tempy;
    private int r; //棋子半径
    private boolean IsMyTurn = true; //IsMyTurn 判断是否该自己走棋
    public short LocalPlayer = REDPLAYER; //LocalPlayer 记录自己是红方还是黑方
    private String message = ""; //提示信息
}

```



```
//线程消亡的标识位
private boolean flag = false;
private int otherport;           //对方端口
private int receiveport;        //本机接收端口
```

将当前棋盘的棋子布局的二维数组 Map 进行初始化。由于棋子索引从 0 开始到 31，因此此处初始化为-1。代码如下：

```
private void cls_map() {
    int i, j;
    for (i = 1; i <= 9; i++) {
        for (j = 1; j <= 10; j++) {
            Map[i][j] = -1;
        }
    }
}
```

棋盘类的构造方法较简单，主要初始化并保存了当前棋盘的棋子布局的二维数组 Map。在构造方法中添加了鼠标监听事件。

游戏面板的鼠标单击事件处理用户走棋过程。代码如下：

```
public ChessBoard()           //构造方法
{
    r = 20;
    cls_map();                //初始化当前棋盘的棋子布局
    /**
     * 鼠标监听事件
     */
    addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            if (IsMyTurn == false) {
                message = "该对方走棋";
                repaint();
                return;
            }
            int x = e.getX();
            int y = e.getY();
            selectChess(e);
            System.out.println(x);
            repaint();
        }
    })
}
```

用户走棋时，首先应选中自己的棋子（第一次选择棋子），因此有必要判断是否单击成对方棋子了。如果是自己的棋子，则 firstChess2 记录用户选择的棋子，同时棋子被加上黑色框线示意被选中。

当用户选过己方棋子后，单击对方棋子（secondChess2 记录用户第二次选择的棋子），则是吃子，如果“将”或“帅”被吃掉，则游戏结束。当然第二次选择棋子有可能是用户改变主意，选择自己的另一棋子，则 firstChess2 重新记录用户选择的己方棋子。

当用户选过己方棋子后，单击的位置无棋子，则处理没有吃子的走棋过程。调用 IsAbleToPut(CurSelect, x, y)方法判断是否能走棋，如果符合走棋规则，则发送此步走棋信息。代码如下：

```
private void selectChess(MouseEvent e) {
```



```

int idx, idx2;                //保存第一次和第二次被单击棋子的索引号
if (first) {
    //第一次选择棋子
    firstChess2 = analyse(e.getX(), e.getY());
    x1 = tempX;
    y1 = tempY;
    if (firstChess2 != null) {
        if (firstChess2.player != LocalPlayer) {
            message = "单击成对方棋子了!";
            return;
        }
        first = false;
    }
} else {
    //第二次选择棋子
    secondChess2 = analyse(e.getX(), e.getY());
    x2 = tempX;
    y2 = tempY;
    //如果是自己的棋子, 则替换上次选择的棋子
    if (secondChess2 != null) {
        if (secondChess2.player == LocalPlayer) {
            //取消上次选择的棋子
            firstChess2 = secondChess2;
            x1 = tempX;
            y1 = tempY;
            secondChess2 = null;
            return;
        }
    }
}
if (secondChess2 == null)      //目标处没棋子, 移动棋子
{
    if (IsAbleToPut(firstChess2, x2, y2)) {
        //在 Map 中去掉原 CurSelect 棋子
        idx = Map[x1][y1];
        Map[x1][y1] = -1;
        Map[x2][y2] = idx;
        chess[idx].SetPos(x2, y2);
        //调用 send() 方法
        send("move" + "|" + idx + "|" + (10 - x2) + "|"
            + String.valueOf(11 - y2) + "|");
        first = true;
        repaint();
        SetMyTurn(false);      //轮到对方走棋了
        //toolStripStatusLabel1.Text = "";
    } else {
        //错误走棋
        message = "不符合走棋规则";
    }
    return;
}
if (secondChess2 != null
    && IsAbleToPut(firstChess2, x2, y2))    //可以吃子
{
    first = true;

```



```

//在 Map 中去掉原 CurSelect 棋子
idx = Map[x1][y1];
idx2 = Map[x2][y2];
Map[x1][y1] = -1;
Map[x2][y2] = idx;
chess[idx].SetPos(x2, y2);
chess[idx2] = null;
repaint();
if (idx2 == 0) //0 代表“将”
{
    message = "红方赢了";
    JOptionPane.showConfirmDialog(null, "红方赢了", "提示",
        JOptionPane.DEFAULT_OPTION);
    //调用 send() 方法
    send("move" + "|" + idx + "|" + (10 - x2) + "|"
        + String.valueOf(11 - y2) + "|");
    send("succ" + "|" + "红方赢了" + "|");
    //btnNew.setEnabled(true); //可以重新开始
    return;
}
if (idx2 == 16) //16 代表“帅”
{
    message = "黑方赢了";
    JOptionPane.showConfirmDialog(null, "黑方赢了", "提示",
        JOptionPane.DEFAULT_OPTION);
    send("move" + "|" + idx + "|" + (10 - x2) + "|"
        + String.valueOf(11 - y2) + "|");
    send("succ" + "|" + "黑方赢了" + "|");
    return;
}
//调用 send() 方法
send("move" + "|" + idx + "|" + (10 - x2) + "|"
    + String.valueOf(11 - y2) + "|");
SetMyTurn(false); //轮到对方走棋了
} else //不能吃子
{
    message = "不能吃子";
}
}
}); //构造方法结束

```

当用户开始游戏后, 向对方发送联机 send("join") 命令, 同时启动线程 th 接收对方发来的各种信息。代码如下:

```

public void startJoin(String ip, int otherport, int receiveport) // 开始联机
{
    flag = true;
    this.otherport = otherport;
    this.receiveport = receiveport;
    send("join");
    //创建一个线程 th
    Thread th = new Thread(this);
    //启动线程
}

```



```
th.start();
message = "程序处于等待联机状态! ";
```

```
}
```

当用户联机成功后,NewGame(short player)方法根据玩家的角色(黑方还是红方),用 InitChess()方法初始化棋子布局,布局时按黑方棋子在上,红方棋子在下进行设计。如果玩家的角色是黑方,为了便于玩家看棋,将所有棋子对调,即黑方棋子在下,红方棋子在上。布局后将所有棋子、棋盘重画并显示出来。代码如下:

```
public final void NewGame(short player) //棋子初始布局
{
    cls_map(); //清空存储棋子的信息数组
    InitChess(); //初始棋子布局
    if (player == BLACKPLAYER) {
        ReverseBoard(); //将所有棋子对调即黑方棋子在下,红方棋子在上
    }
    repaint();
}

private void InitChess() {
    //布置黑方棋子
    chess[0] = new Chess(BLACKPLAYER, "将", new Point(5, 1));
    Map[5][1] = 0;
    chess[1] = new Chess(BLACKPLAYER, "士", new Point(4, 1));
    Map[4][1] = 1;
    chess[2] = new Chess(BLACKPLAYER, "士", new Point(6, 1));
    Map[6][1] = 2;
    chess[3] = new Chess(BLACKPLAYER, "象", new Point(3, 1));
    Map[3][1] = 3;
    chess[4] = new Chess(BLACKPLAYER, "象", new Point(7, 1));
    Map[7][1] = 4;
    chess[5] = new Chess(BLACKPLAYER, "马", new Point(2, 1));
    Map[2][1] = 5;
    chess[6] = new Chess(BLACKPLAYER, "马", new Point(8, 1));
    Map[8][1] = 6;

    chess[7] = new Chess(BLACKPLAYER, "车", new Point(1, 1));
    Map[1][1] = 7;
    chess[8] = new Chess(BLACKPLAYER, "车", new Point(9, 1));
    Map[9][1] = 8;

    chess[9] = new Chess(BLACKPLAYER, "炮", new Point(2, 3));
    Map[2][3] = 9;
    chess[10] = new Chess(BLACKPLAYER, "炮", new Point(8, 3));
    Map[8][3] = 10;

    for (int i = 0; i <= 4; i++) {
        chess[11 + i] = new Chess(BLACKPLAYER, "卒", new Point(1 + i * 2, 4));
        Map[1 + i * 2][4] = 11 + i;
    }

    //布置红方棋子
```



```

    chess[16] = new Chess(REDPLAYER, "帅", new Point(5, 10));
    Map[5][10] = 16;
    chess[17] = new Chess(REDPLAYER, "仕", new Point(4, 10));
    Map[4][10] = 17;
    chess[18] = new Chess(REDPLAYER, "仕", new Point(6, 10));
    Map[6][10] = 18;
    chess[19] = new Chess(REDPLAYER, "相", new Point(3, 10));
    Map[3][10] = 19;
    chess[20] = new Chess(REDPLAYER, "相", new Point(7, 10));
    Map[7][10] = 20;
    chess[21] = new Chess(REDPLAYER, "马", new Point(2, 10));
    Map[2][10] = 21;
    chess[22] = new Chess(REDPLAYER, "马", new Point(8, 10));
    Map[8][10] = 22;

    chess[23] = new Chess(REDPLAYER, "车", new Point(1, 10));
    Map[1][10] = 23;
    chess[24] = new Chess(REDPLAYER, "车", new Point(9, 10));
    Map[9][10] = 24;

    chess[25] = new Chess(REDPLAYER, "炮", new Point(2, 8));
    Map[2][8] = 25;
    chess[26] = new Chess(REDPLAYER, "炮", new Point(8, 8));
    Map[8][8] = 26;

    for (int i = 0; i <= 4; i++) {
        chess[27 + i] = new Chess(REDPLAYER, "兵", new Point(1 + i * 2, 7));
        Map[1 + i * 2][7] = 27 + i;
    }
}

```

ReverseBoard()方法翻转棋子，将黑红方棋子对调。代码如下：

```

private void ReverseBoard()    //翻转棋子
{
    int x, y, c;
    //对调(x, y)与(10-x, 11-y)处棋子
    for (int i = 0; i < 32; i++)
        if (chess[i] != null)
        {
            chess[i].ReversePos();
        }
    //对调 Map 记录的棋子索引号
    for (x = 1; x <= 9; x++) {
        for (y = 1; y <= 5; y++) {
            if (Map[x][y] != -1) {
                c = Map[10 - x][11 - y];
                Map[10 - x][11 - y] = Map[x][y];
                Map[x][y] = c;
            }
        }
    }
}

```


上面方法使用 paint(Graphics g)事件重画游戏中的背景棋盘和所有棋子对象。代码如下:

//重画场景中的所有对象

```
public void paint(Graphics g) {
    g.clearRect(0, 0, this.getWidth(), this.getHeight());
    Image bgImage = Toolkit.getDefaultToolkit().getImage("pic\\qipan.jpg");
    //绘制背景棋盘
    g.drawImage(bgImage, 1, 20, this);
    //画棋子
    for (int i = 0; i < 32; i++) {
        if (chess[i] != null) {
            chess[i].paint(g, this);
        }
    }
    if (firstChess2 != null) {
        firstChess2.DrawSelectedChess(g);
    }
    if (secondChess2 != null) {
        secondChess2.DrawSelectedChess(g);
    }
    g.drawString(message, 0, 450);
}
```

IsAbleToPut(firstchess, x, y)方法实现判断是否能走棋,并返回逻辑值,这段代码最复杂。

//IsAbleToPut(firstchess, x, y)方法实现判断是否能走棋,并返回逻辑值,这段代码最复杂

```
public final boolean IsAbleToPut(Chess firstchess, int x, int y) {
    int i, j, c;
    int oldx, oldy;           //在棋盘原坐标
    oldx = firstchess.pos.x;
    oldy = firstchess.pos.y;
    String qi_name = firstchess.typeName;
    if (qi_name.equals("将") || qi_name.equals("帅")) {
        if ((x - oldx) * (y - oldy) != 0) {
            return false;
        }
        if (Math.abs(x - oldx) > 1 || Math.abs(y - oldy) > 1) {
            return false;
        }
        if (x < 4 || x > 6 || (y > 3 && y < 8)) {
            return false;
        }
        return true;
    }
    if (qi_name.equals("士") || qi_name.equals("仕")) {
        if ((x - oldx) * (y - oldy) == 0) {
            return false;
        }
        if (Math.abs(x - oldx) > 1 || Math.abs(y - oldy) > 1) {
            return false;
        }
        if (x < 4 || x > 6 || (y > 3 && y < 8)) {
            return false;
        }
        return true;
    }
}
```



```
if (qi_name.equals("象") || qi_name.equals("相")) {
    if ((x - oldx) * (y - oldy) == 0) {
        return false;
    }
    if (Math.abs(x - oldx) != 2 || Math.abs(y - oldy) != 2) {
        return false;
    }
    if (y < 6) {
        return false;
    }
    i = 0; // i, j 必须设置初始值
    j = 0;
    if (x - oldx == 2) {
        i = x - 1;
    }
    if (x - oldx == -2) {
        i = x + 1;
    }
    if (y - oldy == 2) {
        j = y - 1;
    }
    if (y - oldy == -2) {
        j = y + 1;
    }
    if (Map[i][j] != -1) {
        return false;
    }
    return true;
}

if (qi_name.equals("马") || qi_name.equals("马")) {
    if (Math.abs(x - oldx) * Math.abs(y - oldy) != 2) {
        return false;
    }
    if (x - oldx == 2) {
        if (Map[x - 1][oldy] != -1) {
            return false;
        }
    }
    if (x - oldx == -2) {
        if (Map[x + 1][oldy] != -1) {
            return false;
        }
    }
    if (y - oldy == 2) {
        if (Map[oldx][y - 1] != -1) {
            return false;
        }
    }
    if (y - oldy == -2) {
        if (Map[oldx][y + 1] != -1) {
            return false;
        }
    }
    return true;
}
```



```

}
if (qi_name.equals("车") || qi_name.equals("车")) {
    //判断是否为直线
    if ((x - oldx) * (y - oldy) != 0) {
        return false;
    }
    //判断是否隔有棋子
    if (x != oldx) {
        if (oldx > x) {
            int t = x;
            x = oldx;
            oldx = t;
        }
        for (i = oldx; i <= x; i += 1) {
            if (i != x && i != oldx) {
                if (Map[i][y] != -1) {
                    return false;
                }
            }
        }
    }
    if (y != oldy) {
        if (oldy > y) {
            int t = y;
            y = oldy;
            oldy = t;
        }
        for (j = oldy; j <= y; j += 1) {
            if (j != y && j != oldy) {
                if (Map[x][j] != -1) {
                    return false;
                }
            }
        }
    }
    return true;
}

if (qi_name.equals("炮") || qi_name.equals("炮")) {
    boolean swapflagx = false;
    boolean swapflagy = false;
    if ((x - oldx) * (y - oldy) != 0) {
        return false;
    }
    c = 0;
    if (x != oldx) {
        if (oldx > x) {
            int t = x;
            x = oldx;
            oldx = t;
            swapflagx = true;
        }
        for (i = oldx; i <= x; i += 1) {
            if (i != x && i != oldx) {
                if (Map[i][y] != -1) {
                    c = c + 1;

```



```

        }
    }
}
if (y != oldy) {
    if (oldy > y) {
        int t = y;
        y = oldy;
        oldy = t;
        swapflagy = true;
    }
    for (j = oldy; j <= y; j += 1) {
        if (j != y && j != oldy) {
            if (Map[x][j] != -1) {
                c = c + 1;
            }
        }
    }
}
if (c > 1) //与目标处间隔 1 个以上棋子
{
    return false;
}
if (c == 0) //与目标处无间隔棋子
{
    if (swapflagx == true) {
        int t = x;
        x = oldx;
        oldx = t;
    }
    if (swapflagy == true) {
        int t = y;
        y = oldy;
        oldy = t;
    }
    if (Map[x][y] != -1) {
        return false;
    }
}
if (c == 1) //与目标处间隔 1 个棋子
{
    if (swapflagx == true) {
        int t = x;
        x = oldx;
        oldx = t;
    }
    if (swapflagy == true) {
        int t = y;
        y = oldy;
        oldy = t;
    }
    if (Map[x][y] == -1) //如果目标处无棋子, 则不能走此步
    {
        return false;
    }
}

```



```

    }
    return true;
}
if (qi_name.equals("卒") || qi_name.equals("兵")) {
    if ((x - oldx) * (y - oldy) != 0) {
        return false;
    }
    if (Math.abs(x - oldx) > 1 || Math.abs(y - oldy) > 1) {
        return false;
    }
    if (y >= 6 && (x - oldx) != 0) {
        return false;
    }
    if (y - oldy > 0) {
        return false;
    }
    return true;
}
return false;
}

```

read()方法不断侦听本机设定的端口,得到对方发送过来的信息,根据自己定义的通信协议中传送的是“输赢信息”、“下的棋子位置信息”、“认输”等信息而分别进行处理。代码如下:

//线程执行的内容

```

public void run() {
    try {
        //指定接收端口
        DatagramSocket s = new DatagramSocket(receiveport);
        byte[] data = new byte[100];
        DatagramPacket dgp = new DatagramPacket(data, data.length);
        //进入一个无限循环来接收数据包
        while (flag == true) {
            s.receive(dgp); //接收数据包
            String strData = new String(data);
            String[] a = new String[6];
            a = strData.split("\\|");
            if (a[0].equals("join")) {
                LocalPlayer = BLACKPLAYER;
                //显示棋子
                NewGame(LocalPlayer);
                if (LocalPlayer == REDPLAYER) {
                    SetMyTurn(true); //能走棋
                } else {
                    SetMyTurn(false);
                }
                //发送联机成功信息
                send("conn|");
            } else if (a[0].equals("conn")) //联机成功信息
            {
                LocalPlayer = REDPLAYER;
                //显示棋子
                NewGame(LocalPlayer);
                if (LocalPlayer == REDPLAYER) {

```



```

        SetMyTurn(true);                //能走棋
    } else {
        SetMyTurn(false);
    }
} else if (a[0].equals("succ")) {
    //获取传送信息到本地端口号的远程计算机 IP 地址
    if (a[1].equals("黑方赢了")) {
        JOptionPane.showConfirmDialog(null,
"黑方赢了你可以重新开始!", "你输了", JOptionPane.DEFAULT_OPTION);
    }
    if (a[1].equals("红方赢了")) {
        JOptionPane.showConfirmDialog(null,
"红方赢了你可以重新开始!", "你输了", JOptionPane.DEFAULT_OPTION);
    }
    message = "你可以重新开局! ";
} else if (a[0].equals("move")) {
    //对方的走棋信息, move|棋子索引号|X|Y
    int idx = Short.parseShort(a[1]);
    x2 = Short.parseShort(a[2]);
    y2 = Short.parseShort(a[3]);
    String z = a[4];                //对方上步走棋的棋谱信息
    message = x2 + ":" + y2;
    Chess c = chess[idx];
    x1 = c.pos.x;                //(x1,y1)是被移动棋子在棋盘原处的坐标
    y1 = c.pos.y;

    //修改棋子位置, 显示对方走棋信息
    idx = Map[x1][y1];
    int idx2 = Map[x2][y2];
    Map[x1][y1] = -1;
    Map[x2][y2] = idx;
    chess[idx].SetPos(x2, y2);
    if (idx2 != -1) {
        chess[idx2] = null;
    }
    repaint();
    IsMyTurn = true;
    //SetMyTurn(true);
} else if (a[0].equals("quit")) {
    JOptionPane.showConfirmDialog(null, "对方退出了, 游戏结束!", "提示",
JOptionPane.DEFAULT_OPTION);
    message = "对方退出了, 游戏结束! ";
    flag = false;
} else if (a[0].equals("lose")) {
    JOptionPane.showConfirmDialog(null, "恭喜你, 对方认输了!", "你赢了",
JOptionPane.DEFAULT_OPTION);
    SetMyTurn(false);
}
System.out.println(new String(data));
}
} catch (SocketException e) {
    //TODO Auto-generated catch block

```



```

        e.printStackTrace();
    } catch (IOException e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

发送信息 send(String str)方法实现较为简单, 主要实现创建 UDP 网络服务, 传送信息到指定计算机的 otherport 端口号后, 关闭 UDP 网络服务。代码如下:

```

public void send(String str)                //发送信息
{
    //message=str;
    DatagramSocket s = null;
    try {
        s = new DatagramSocket();
        byte[] buffer;
        buffer = new String(str).getBytes();
        InetAddress ia = InetAddress.getLocalHost();    //本机地址
        //目的主机地址
        DatagramPacket dgp = new DatagramPacket(buffer, buffer.length, ia, otherport);
        s.send(dgp);
        System.out.println("发送信息:" + str);
    } catch (IOException e) {
        System.out.println(e.toString());
    } finally {
        if (s != null)
            s.close();
    }
}

```

16.4.3 设计游戏窗体 (Frmchess.java)

游戏窗口类 Frmchess 实现游戏的全部功能, 继承 JFrame 组件实现。由上方 Panel1、中间 Panel2 和下方 Panel3 组成。上方 Panel1 仅显示“帅”字图片; 中间 Panel2 是游戏区, 是棋盘类 ChessBoard 对象, 实现游戏走棋功能; 下方 Panel3 添加了“认输按钮”、“开始”按钮及输入对方 IP、端口的文本框。代码如下:

```

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;

public class Frmchess extends JFrame {
    ChessBoard panel2 = new ChessBoard();
    JButton button1 = new JButton("认输");
    JButton button2 = new JButton("开始");
    JTextField jTextField1 = new JTextField();    //输入 IP
    JTextField jTextField2 = new JTextField();    //输入对方端口
    public static final short REDPLAYER = 1;
    public static final short BLACKPLAYER = 0;
}

```



```

public Frmchess() {
    JPanel panel1 = new JPanel(new BorderLayout());
    JPanel panel3 = new JPanel(new BorderLayout());
    String urlString = "C:
    JLabel label = new JLabel(new ImageIcon(urlString));
    panel1.add(label, BorderLayout.CENTER);
    panel2.setLayout(new BorderLayout());
    panel3.setLayout(new FlowLayout());
    JLabel jLabel1 = new JLabel("输入 IP");
    JLabel jLabel2 = new JLabel("输入对方端口");
    panel3.add(jLabel1);
    panel3.add(jLabel2);
    jTextField1.setText("127.0.0.1");
    jTextField2.setText("3004");
    panel3.add(jLabel1);
    panel3.add(jTextField1);
    panel3.add(jLabel2);
    panel3.add(jTextField2);
    panel3.add(button1);
    panel3.add(button2);
    this.getContentPane().setLayout(new BorderLayout());
    this.getContentPane().add(panel1, BorderLayout.NORTH);
    this.getContentPane().add(panel2, BorderLayout.CENTER);
    this.getContentPane().add(panel3, BorderLayout.SOUTH);
    this.setSize(380, 600);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setTitle("网络中国象棋游戏");
    this.setVisible(true);
    button1.setEnabled(false);
    button2.setEnabled(true);
    setVisible(true);
    this.addWindowListener(new WindowAdapter() { //窗口关闭事件
        public void windowClosing(WindowEvent e) {
            try {
                panel2.send("quit|"); //向对方发送离开信息
                System.exit(0);
            } catch (Exception ex) {
            }
        }
    });
    /**
     * 鼠标监听事件
     */
    button1.addMouseListener(new MouseAdapter() { //“认输”按钮
        @Override
        public void mouseClicked(MouseEvent e) {
            try {
                panel2.send("lose|"); //向对方发送认输信息
            } catch (Exception ex) {
            }
        }
    });
}

```

IsMyChess(int idx)方法根据棋子数组下标判断是否是自己棋子。棋子数组下标 idx 为 0~15

是黑方, 16~31 是红方。代码如下:

```
private boolean IsMyChess(int idx) {
    boolean functionReturnValue = false;
    if (idx >= 0 && idx < 16 && LocalPlayer == BLACKPLAYER) {
        functionReturnValue = true;
    }
    if (idx >= 16 && idx < 32 && LocalPlayer == REDPLAYER) {
        functionReturnValue = true;
    }
    return functionReturnValue;
}
```

SetMyTurn()方法设置是否该自己走棋的提示信息。代码如下:

//设置是否该自己走棋的提示信息

```
private void SetMyTurn(boolean bolIsMyTurn) {
    IsMyTurn = bolIsMyTurn;
    if (bolIsMyTurn) {
        message = "请您开始走棋";
    } else {
        message = "对方正在思考...";
    }
}
```

“开始”按钮的单击事件启动线程, 通过 read()方法实现不断侦听本机设定的端口, 得到对方发送过来的信息。同时显示棋盘, 并默认自己为黑方。代码如下:

```
button2.addMouseListener(new MouseAdapter() { // “开始”按钮
    @Override
    public void mouseClicked(MouseEvent e) {
        String ip = jTextField1.getText();
        int remoteport = Integer
            .parseInt(jTextField2.getText());
        int receiveport;
        if (remoteport == 3003)
            receiveport = 3004;
        else
            receiveport = 3003;
        panel2.startJoin(ip, remoteport, receiveport); //开始联机
        button1.setEnabled(true);
        button2.setEnabled(true);
    }
});

public static void main(String[] args) {
    Frmchess f = new Frmchess();
}
```

如果你只有一台计算机, 可以运行两个实例, 其中将地址填写为: 127.0.0.1, 这样一个作为红方, 另一个作为黑方, 便可以和自己对弈了。注意对方端口一个为 3003, 另一个为 3004。如果在不同的计算机上, 就都可以用 3003。两人对战象棋运行界面如图 16-4 所示, 运行初始界面如图 16-5 所示。

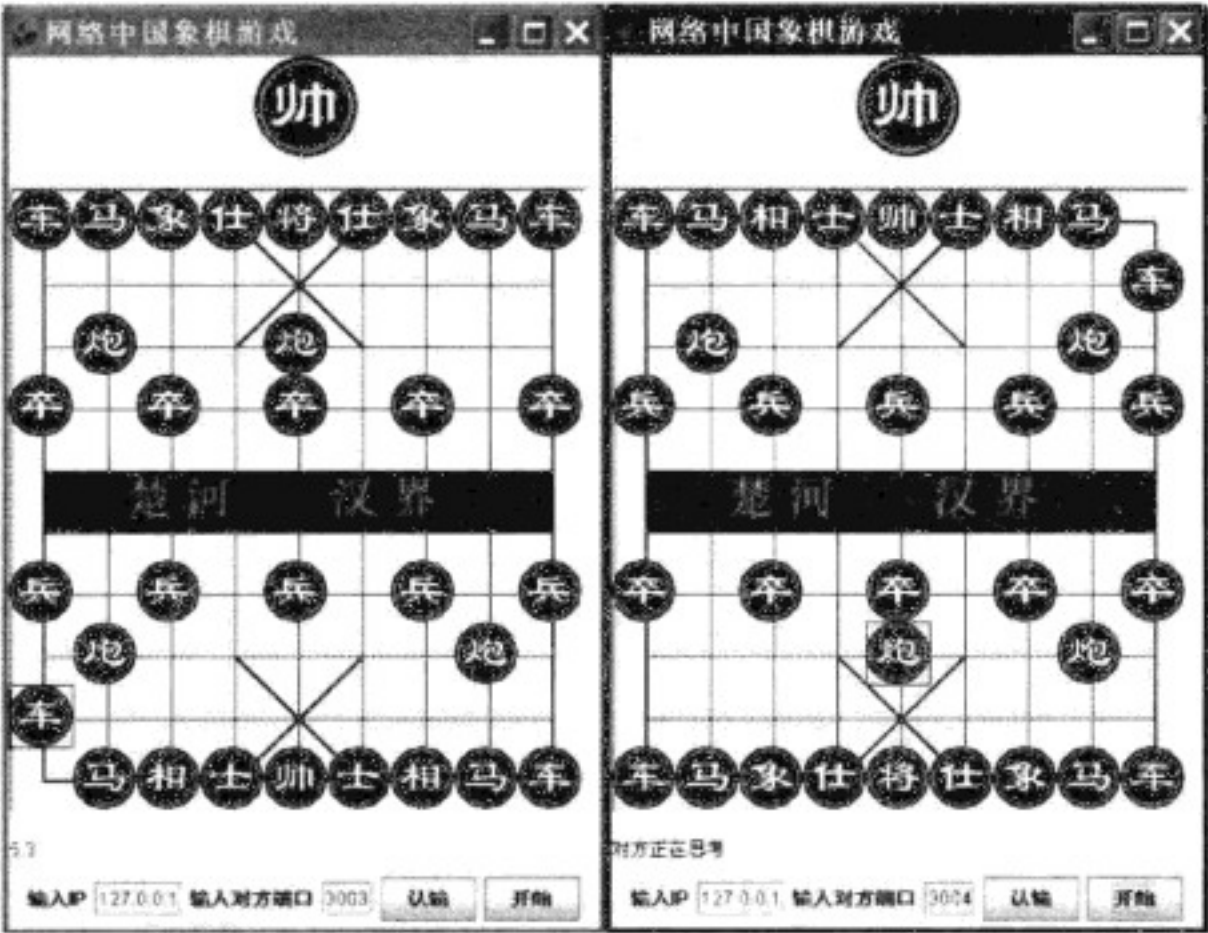


图 16-4 两人对战象棋运行界面

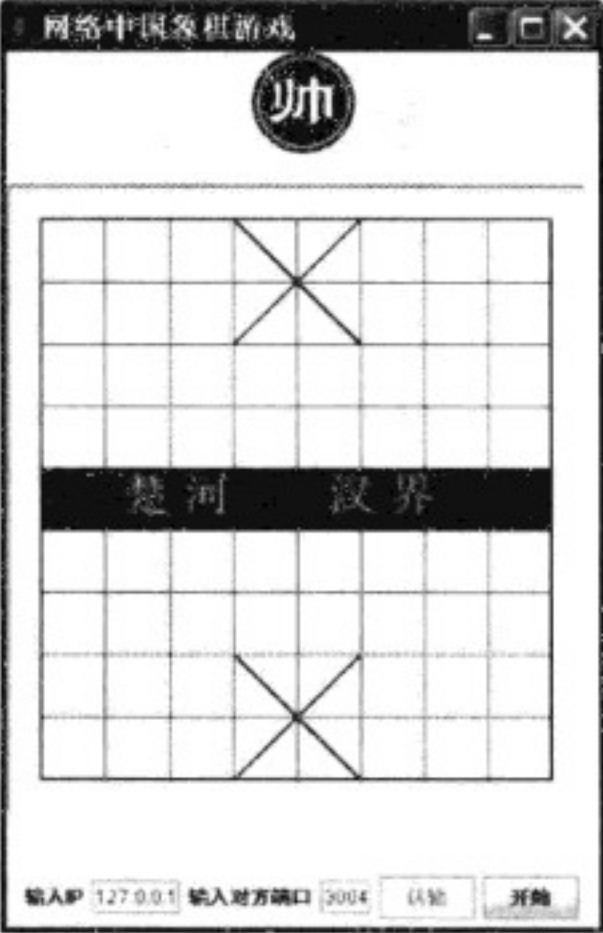


图 16-5 两人对战象棋运行初始界面

第 17 章

RGP 走迷宫游戏

17.1 RGP 走迷宫游戏介绍

RGP 走迷宫游戏就是玩家通过方向键控制游戏主角 RGP 人物从绿色通道中走到右下角的出口处。游戏可以提高玩家的观察力，训练玩家的思维力，开发玩家的想象力。RGP 走迷宫游戏的运行界面如图 17-1 所示。游戏中 RGP 人物有动画的效果。

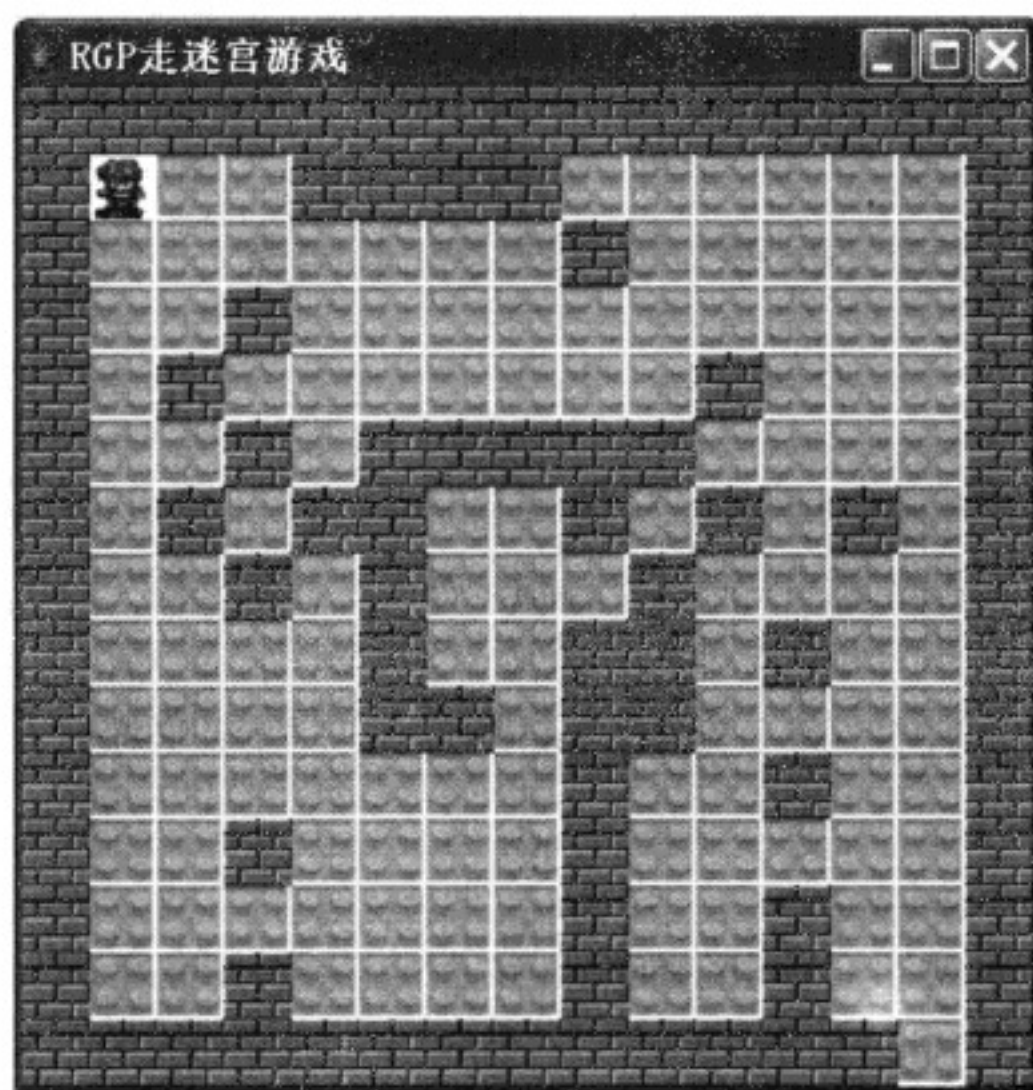


图 17-1 走迷宫游戏运行界面

17.2 程序设计的思路

17.2.1 游戏素材

游戏程序中用到地板、人物、墙，分别使用图 17-2 所示的图片来表示。

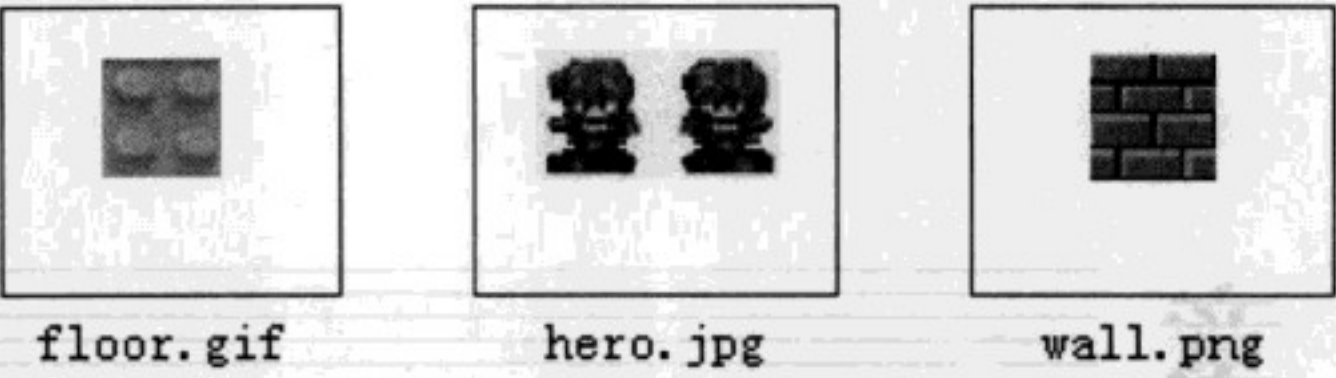


图 17-2 相关图片素材

17.2.2 设计思路

游戏在 15×15 的方格区域中实现,因此采用二维数组 map 保存游戏地图信息。根据 map[x][y] 中记录的地图信息在屏幕上绘制图案。这里我们仅定义: 0 时画出地板 floor.gif, 1 时画出城墙 wall.png, 当然可以根据游戏的需要定义椅子为 2、宝座为 3 等图案信息, 即可勾勒出一张内容丰富的背景地图。

人物的移动在 keyPressed(KeyEvent e) 按键事件中处理, 当人物角色的移动符合游戏规则才可以移动, 例如碰到墙不能移动。无论是否移动均调用 repaint() 方法重新绘制窗体。在移动后, 判断是否到达出口 (13, 13) 坐标处, 如果到达则游戏结束。

17.2.3 RGP 人物的动画

大家也许都知道, 所谓的动画, 并不是一个“会动的画”, 而是一组“连续变动的画”, 就好比 Flash 制作时需要凭借“帧”调节画面运动一样, 在 Java 游戏开发中一样要通过类似的方式来控制画面。

实现这一点, 首先我们需要一组像下面这样连续的图像。



日常生活中, 我们都知道自己应该迈左脚还是迈右脚, 但对计算机而言必须明确提示下一步应该迈哪只脚。因此, 我们还需要一个变量 count 充当“计步器”, 以明确下步状态。

```
private int count;
```

而要想实现动画, 最重要的一点, 就是画面的连续, 即多步操作的处理, 为此我们使用到了 Java 中的 Thread, 即线程。

```
private Thread threadAnime;
```

在 Java 中, 目前不支持例如 C# 式的函数直接被线程调用的方式。在 Java 中实现线程, 必须通过继承 Thread 类或实现 Runnable 接口。

这里以使用 Thread 类的继承为例, 代码如下:

```
//内部类, 用于处理计步动作
private class AnimationThread extends Thread {
    public void run() {
        while (true) {
            //count 计步器
            if (count == 0) {
                count = 1;
            } else if (count == 1) {
                count = 0;
            }
        }
    }
}
```



```

        //重绘画面
        repaint();

        //每 300ms 改变一次动作
        try {
            Thread.sleep(300);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

所谓继承，也可以简单地理解为复制所继承类的全部方法，而这里我们重写了 `run()` 方法，没有改变其他方法。也就是说，我们将以自己的方式运行 `AnimationThread` 这个类。

另外，在处理画人物的方法时，我们将其内部变更如下：

```

//以 count 作为图像的偏移数值
g.drawImage(roleImage, x*CS, y*CS, x*CS+CS, y*CS+CS,
            count*CS, 0, CS+count*CS, CS, this);

```

推导公式如图 17-3 所示。

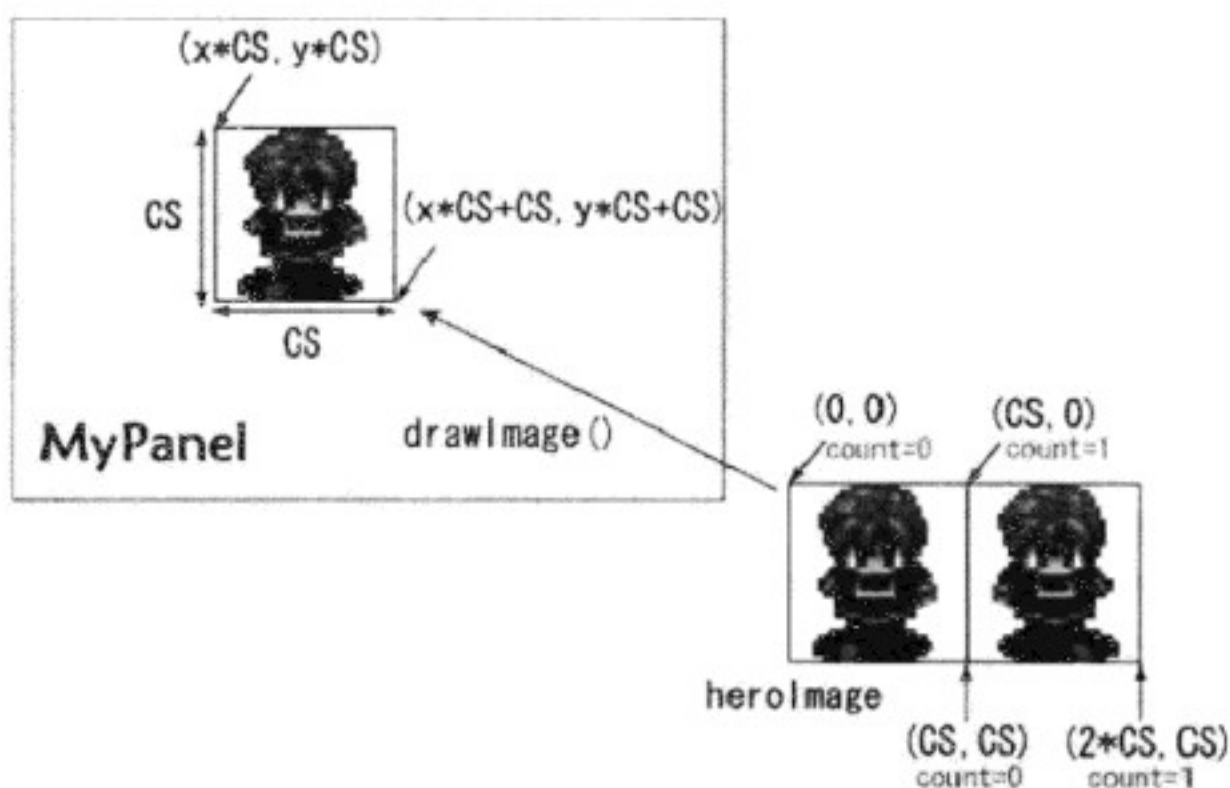


图 17-3 推导公式

最后，我们在 `MyPanel` 构建之初即启动线程，令线程的相关操作活性化。代码如下：

```

//实例化内部线程 AnimationThread
threadAnime = new Thread(new AnimationThread());
//启动线程
threadAnime.start();

```

17.3 走迷宫游戏设计的步骤

17.3.1 设计主窗体类 (miGong.java)

```

import java.awt.Container;
import javax.swing.JFrame;

```



```

public class miGong extends JFrame {
    public miGong () {
        //默认的窗体名称
        setTitle("RGP 走迷宫游戏");
        //获得我们自定义面板（地图面板）的实例
        MyPanel panel = new MyPanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
        //执行并构建窗体设定
        pack();
    }
    public static void main(String[] args) {
        Example1 el = new Example1();
        //设定允许窗体关闭操作
        el.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //显示窗体
        el.setVisible(true);
    }
}

```

17.3.2 设计游戏面板类（MyPanel.java）

游戏面板类 MyPanel 继承 JPanel 类，用于实现游戏地图及人物的行走功能，同时实现 KeyListener 接口响应用户的按键事件。

导入包及相关类：

```

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.*;
public class MyPanel extends JPanel implements KeyListener{

```

MyPanel 面板类实现鼠标侦听接口，并定义一些成员变量。代码如下：

```

    private static final long serialVersionUID = 1L;
    //窗体的宽与高
    private static final int WIDTH = 480;
    private static final int HEIGHT = 480;
    //设定背景方格默认行数
    private static final int ROW = 15;
    //设定背景方格默认列数
    private static final int COL = 15;
    //单个图像大小，默认采用 32×32 大小的图形，可根据需要调整比例。
    //当然，始终应和窗体大小比例协调；比如 32×32 大小的图片，如果
    //一行设置 15 个，那么就是 480，也就是本例默认的窗体大小，
    //当然，我们也可以根据 ROW×CS，COL×CS 在初始化时自动调整
    private static final int CS = 32;
    //设定地图
    private int[][] map = { {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1} ,
        {1,0,0,0,1,1,1,1,0,0,0,0,0,0,1} , {1,0,0,0,0,0,0,0,0,1,0,0,0,0,1} ,
        {1,0,0,1,0,0,0,0,0,0,0,0,0,0,1} , {1,0,1,0,0,0,0,0,0,0,1,0,0,0,1} ,

```



```

{1,0,0,1,0,1,1,1,1,0,0,0,0,1} , {1,0,1,0,1,1,0,0,1,0,1,0,1,0,1} ,
{1,0,0,1,0,1,0,0,0,1,0,0,0,0,1} , {1,0,0,0,0,1,0,0,1,1,0,1,0,0,1} ,
{1,0,0,0,0,1,1,0,1,1,0,0,0,0,1} , {1,0,0,0,0,0,0,0,1,0,0,1,0,0,1} ,
{1,0,0,1,0,0,0,0,1,0,0,0,0,0,1} , {1,0,0,0,0,0,0,0,1,0,0,1,0,0,1} ,
{1,0,0,1,0,0,0,0,1,0,0,1,0,0,1} , {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}} ;
//设定显示图像的对象
private Image floorImage;
private Image wallImage;
//新增了一个角色
private Image roleImage;
//角色坐标
private int x, y;
//此处我们添加一组常数,用来区别左、右、上、下按键的触发
//之所以采用数字进行区别,原因是数字运算效率高
private static final int LEFT = 0;
private static final int RIGHT = 1;
private static final int UP = 2;
private static final int DOWN = 3;
//增加计步器
private int count;
private Thread threadAnime;
private int direction; //新增变量,用来确认角色所对应的方向,对应的按键将被触发

```

MyPanel 面板类的构造方法初始化人物角色的位置,实例化内部线程 AnimationThread 控制人物自身的动画效果。代码如下:

```

public MyPanel() {
    //设定初始构造时面板的大小
    setPreferredSize(new Dimension(WIDTH, HEIGHT));
    //在初始化时载入图形
    loadImage();
    x = 1; //角色起始坐标
    y = 1;
    direction=DOWN; //默认为角色向下
    //在面板构建时赋予计步器初值
    count = 0;
    //设定焦点在本窗体并赋予监听对象
    setFocusable(true);
    addKeyListener(this);
    //实例化内部线程 AnimationThread
    threadAnime = new Thread(new AnimationThread());
    //启动线程
    threadAnime.start();
}

```

paintComponent(Graphics g)方法在 JPanel 基础上调用 drawMap(g)方法构建底层地图背景,drawRole(g)方法在指定位置角色坐标(x,y)处画出人物。代码如下:

```

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    drawMap(g); //画出地图
    drawRole(g); //画出人物
}

```


drawRole(g)画出人物。代码如下:

```
private void drawRole(Graphics g) {
    //g.drawImage(roleImage, x * CS, y * CS, this);
    //以 count 作为图像的偏移数值
    g.drawImage(roleImage, x*CS, y*CS, x*CS+CS, y*CS+CS,
        count*CS, 0, CS+count*CS, CS, this);
}
```

loadImage()方法载入程序中使用的所有图像。代码如下:

```
private void loadImage() {
    //获得当前类对应的相对位置 image 文件夹下的地板图像
    ImageIcon icon = new ImageIcon(getClass().getResource("image/floor.gif"));
    //将地板图像实例赋予 floorImage 变量
    floorImage = icon.getImage();
    //获得当前类对应的相对位置 image 文件夹下的墙体图像
    icon = new ImageIcon(getClass().getResource("image/wall.gif"));
    //将墙体图像实例赋予 wallImage 变量
    wallImage = icon.getImage();
    //导入一名“英雄”来当主角
    icon = new ImageIcon(getClass().getResource("image/hero.jpg"));
    roleImage = icon.getImage();
}
```

drawMap(Graphics g)方法根据 map[x][j]中记录的地图信息在屏幕上绘制图案。这里我们仅定义:标记为 0 时画出地板,标记为 1 时画出城墙,当然可以根据游戏的需要定义椅子为 2、宝座为 3 等图案信息,即可勾勒出一张内容丰富的背景地图。代码如下:

```
private void drawMap(Graphics g) {
    //双层 for 循环进行地图描绘
    for (int x = 0; x < ROW; x++) {
        for (int j = 0; j < COL; j++) {
            switch (map[x][j]) {
                case 0 :
                    //map 的标记为 0 时画出地板
                    //在指定位置描绘出我们所加载的图形
                    g.drawImage(floorImage, j * CS, x*CS, this);
                    break;
                case 1 :
                    //map 的标记为 1 时画出城墙
                    g.drawImage(wallImage, j * CS, x*CS, this);
                    break;
                //我们可以依次类推无数背景组合,例如定义椅子为 2、宝座为 3 等图案信息
                default: //当所有 case 值皆不匹配时,将执行此操作
                    break;
            }
        }
    }
    //g.drawImage(roleImage, 240, 240, this);
}
```

人物的移动在 keyPressed(KeyEvent e)按键事件中处理,当人物角色的移动符合游戏规则时才

可以移动，例如碰到墙不可以移动。无论是否移动均调用 `repaint()` 方法重新绘制窗体。在移动后，判断是否到达出口（13，13）坐标处，如果到达则游戏结束。代码如下：

```
public void keyPressed(KeyEvent e) {
    int keyCode = e.getKeyCode();           //获得按键编号
    switch (keyCode) {                       //通过 keyCode 识别用户的按键
        case KeyEvent.VK_LEFT :             //当触发 Left 时
            //进行 left 操作，仅符合 move() 方法中的规则时执行，以下相同
            move(LEFT);
            break;
        case KeyEvent.VK_RIGHT :            //当触发 Right 时
            move(RIGHT);
            break;
        case KeyEvent.VK_UP :               //当触发 Up 时
            move(UP);
            break;
        case KeyEvent.VK_DOWN :             //当触发 Down 时
            move(DOWN);
            break;
    }
    //重新绘制窗体图像
    repaint();
    if(x==13 &&y==13) //移动目标区（右下角）
        JOptionPane.showMessageDialog(this, "恭喜通过一关");
}
```

`isAllow(int x, int y)` 方法完成符合游戏规则的判断，参数 `(int x, int y)` 是人物移动的目的位置，如果 `(x, y)` 是墙则不能移动，返回 `False`。代码如下：

```
/**
 * 用于判定是否允许移动的发生，被 move() 函数调用
 * @param x
 * @param y
 * @return
 */
private boolean isAllow(int x, int y) {
    //以 (x, y) 为交点进行数据判定
    //在本例中仅以 0 作为地板的参数，1 作为墙的参数
    if (map[y][x] == 1) {
        //不允许移动时，返回 false
        return false;
    }
    //允许移动时，返回 true
    return true;
}
```

`move(int event)` 函数在移动人物角色时，调用 `isAllow(int x, int y)` 方法完成符合游戏规则的判断，如果符合游戏规则则人物位置改变，同时修改人物朝向。代码如下：

```
/** 判断移动事件，关联 isAllow() 方法
```



```

* @param event
*/
private void move(int event) {    //仅执行符合规则的操作
    switch (event) {
        case LEFT:
            //依次判定事件
            if (isAllow(x-1, y)) x--;    //人物位置改变
            direction = LEFT;
            break;
        case RIGHT:
            if (isAllow(x+1, y)) x++;
            direction = RIGHT;
            break;
        case UP:
            if (isAllow(x, y-1)) y--;
            direction = UP;
            break;
        case DOWN:
            if (isAllow(x, y+1)) y++;
            direction = DOWN;
            break;
        default:
            break;
    }
}

```

以下为内部类，用于处理计步动作，实现人物走动的效果。代码如下：

```

private class AnimationThread extends Thread {
    public void run() {
        while (true) {
            //count 计步器
            if (count == 0) {
                count = 1;
            } else if (count == 1) {
                count = 0;
            }
            //重绘画面
            repaint();
            //每 300ms 改变一次动作
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
//内部类结束

```

游戏结束画面如图 17-4 所示。



图 17-4 游戏结束画面

第 18 章

青蛙过河游戏

18.1 青蛙过河游戏介绍

青蛙过河游戏是一个有趣的智力游戏。6 只青蛙分成两队，左边 3 只，右边 3 只，都要过河，但是互不相让。中间只有一块空白石头。玩家需要用一种方法将左面的青蛙和右面的青蛙进行互换。

游戏说明如下：

- (1) 用鼠标单击青蛙头部，它会向前跳；
- (2) 最多只能跳过一只青蛙；
- (3) 单击“开始”按钮，游戏复原。

青蛙过河游戏运行界面如图 18-1 所示。

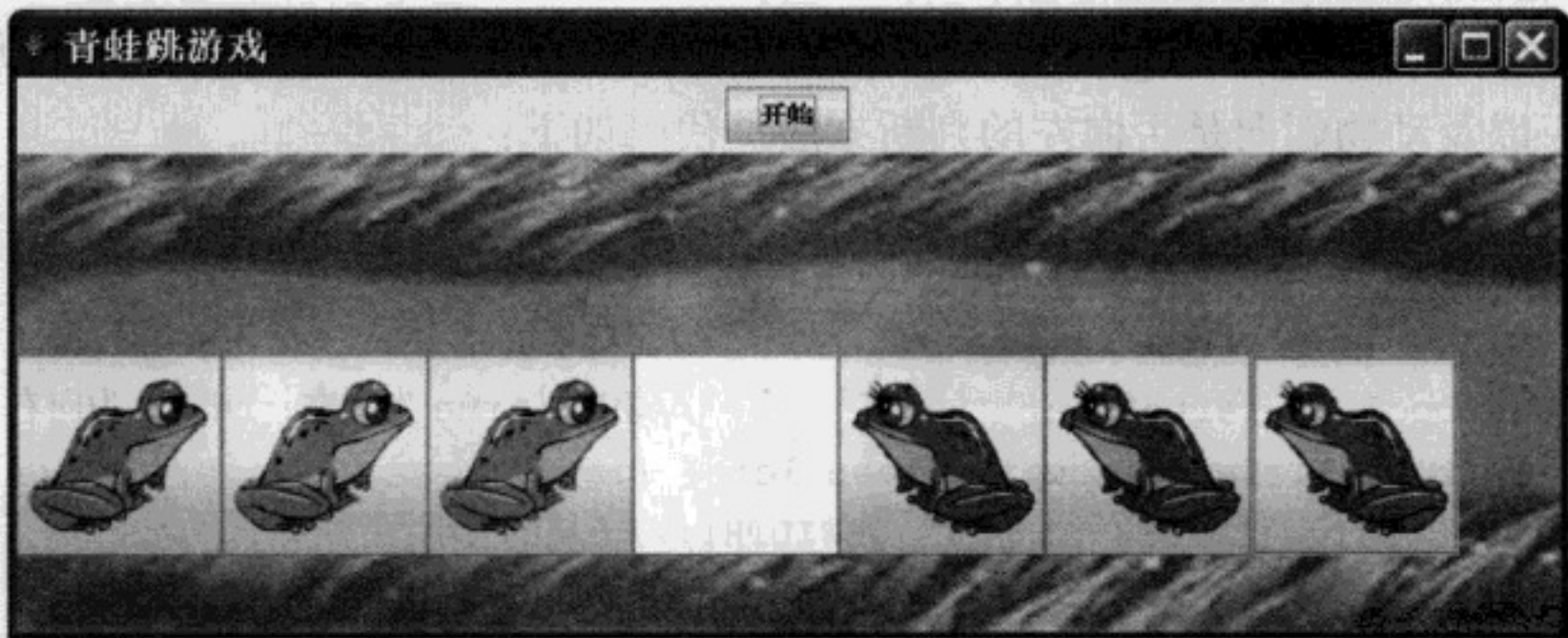


图 18-1 青蛙过河游戏运行界面

18.2 程序设计的思路

18.2.1 游戏素材

游戏程序中用到小河、左右方向的青蛙、空白石头等，分别使用图 18-2 所示的图片来表示。

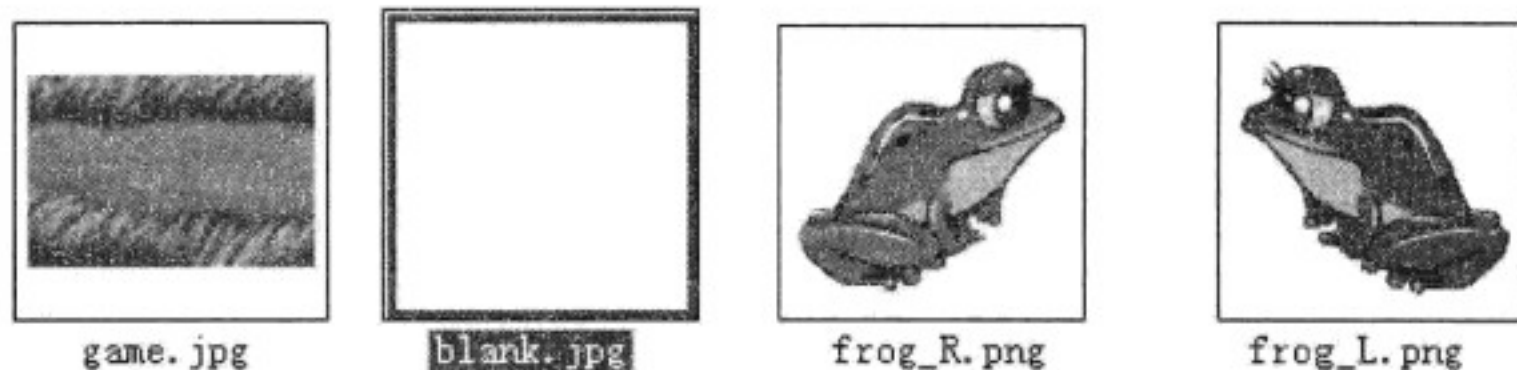


图 18-2 相关图片素材

18.2.2 设计思路

使用一个带背景（background.jpg）的面板作为游戏区域，在其上面显示青蛙方块对象。青蛙对象使用继承 JButton 类的 frogCell 按钮类实现，并且能响应鼠标事件。

创建一个继承 JFrame 类的主窗体类 MainFrame，显示带背景（background.jpg）的面板。在其中加入“开始”按钮，可以将动物方块恢复到初始位置，重新开始游戏。

18.3 青蛙过河游戏设计的步骤

18.3.1 设计青蛙方块类（frogCell.java）

在项目中创建一个继承 JButton 类的的 frogCell 按钮类，用于表示青蛙。

导入包及相关类：

```
import javax.swing.Icon;
import javax.swing.JButton;
```

frogCell 按钮类的构造方法设置青蛙方块的方向、方块的图案、方块显示时的大小，并提供是否是空块，以及动物朝向是否向左的判断方法。代码如下：

```
public class frogCell extends JButton {
    public static final int IMAGEWIDTH = 100; // 方块宽度
    private boolean blank = false;           // 空块标记
    private boolean left = true;              // 朝向，true 为向左，false 为向右
    public frogCell(Icon icon, boolean left) {
        this.setSize(IMAGEWIDTH, IMAGEWIDTH); // 方块的大小
        this.setIcon(icon); // 方块的图标（图案）
        this.left = left; // 方块中的动物朝向
    }
    public boolean isBlank() {
        if (blank)
            return true;
        else
            return false;
    }
    // 判断此块方块中的动物朝向是否向左
    public boolean isLeft() {
        if (left)
            return true;
        else
            return false;
    }
}
```



```

        return false;
    }
    //设置此块方块中的动物朝向（向左或向右）
    public void setLeft(boolean b) {
        left= b;
    }
    //设置此块有无动物图案
    public void setBlank(boolean b) {
        blank= b;
    }
}

```

18.3.2 设计游戏面板类（frogPanel.java）

游戏面板类实现在面板上显示河流图片的背景，并在其上加载 $n+1$ 个动物方块，默认为 6 个（ $n=6$ ）青蛙方块对象及一个空白的方块对象。同时实现鼠标侦听接口来响应鼠标事件。

导入包及相关类：

```

import java.awt.Color;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.*;
public class frogPanel extends JPanel implements MouseListener{
    private frogCell[] cells ;           //动物方块数组
    private frogCell cellBlank = null; //空白方块
    private Image image;                 //河流背景图片

```

frogPanel 面板类的构造方法实例化背景图片，并设置本身为透明效果，否则图片效果将无法显示。代码如下：

```

public frogPanel() { //构造方法
    super();
    setLayout(null); //设置空布局
    try {
        image=ImageIO.read(new File("pic\\game.jpg"));
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    setOpaque(false); //设置为透明
    //init(6);
}

```

init(int n)方法加载 $n+1$ 个动物方块，默认为 6 个（ $n=6$ ）青蛙方块对象及一个空白的方块对象，同时设置这 $n+1$ 个动物方块的图片、朝向、是否空白等属性。最后向面板中添加所有动物方块（含空白方块），对所有方块添加鼠标监听事件。代码如下：

```

public void init(int n) { //初始化游戏
    //删除 Panel 里面已经添加的动物方块
    if(cells!=null)
        this.removeAll();
    cells = new frogCell[n+1]; //创建动物方块数组
    int i= 0; //动物方块序号
    Icon icon = null; //动物图标

```



```

frogCell cell = null;//动物方块对象
for (i = 0; i < cells.length; i++) { //循环
    if(i<cells.length/2) //左边青蛙
    {
        icon = new ImageIcon("pic\\frog_R.png");//获取动物图标
        cell = new frogCell(icon, false);//实例化动物方块对象
        //cell.setLeft(false);
    }
    if(i==(int)(cells.length/2)) //中间空白方块
    {
        icon = new ImageIcon("pic\\blank.jpg");//获取空白方块图标
        cell = new frogCell(icon, false);//实例化动物方块对象
        cell.setBackground(Color.BLACK);
        cell.setVisible(true);
        cell.setBlank(true);//标记此动物方块为空白图标
    }
    if(i>cells.length/2) //右边青蛙
    {
        icon = new ImageIcon("pic\\frog_L.png");//获取动物图标
        cell = new frogCell(icon,true);//实例化动物方块对象
        //cell.setLeft(true);
    }
    //设置动物方块的坐标
    cell.setLocation(i * frogCell.IMAGEWIDTH, 100);
    cells[i] = cell;//将动物方块存储到动物方块数组中
}
for (i = 0; i < cells.length; i++) {
    this.add(cells[i]);//向面板中添加所有动物方块
    cells[i].addMouseListener(this);//对方块添加鼠标监听事件
}
}

```

isOver()方法判断游戏是否结束,通过判断朝向来实现。代码如下:

```

public boolean isOver()
{
    for (int i = 0; i < cells.length; i++) {
        if(i<cells.length/2)
        {
            if(!cells[i].isLeft())return false;
        }
        if(i==(int)(cells.length/2))
        {
            if(!cells[i].isBlank())
                return false;
        }
        if(i>cells.length/2)
        {
            if(cells[i].isLeft())return false;
        }
    }
    return true;
}

```


}

青蛙过河游戏实现的关键是单击事件的处理。通过 `e.getSource()` 方法获取触发事件的对象，找到相应的动物方块。如果被单击的是空白方块，则无需处理。

如果 `cells[i]` 是向左的青蛙方块，则需判断左侧相邻的方块 `cells[i - 1]` 是否为空，为空则相邻的 `cells[i]` 和 `cells[i - 1]` 这两个方块交换图案，并修改它们方块的朝向等属性。假如判断左侧相邻的方块 `cells[i - 1]` 不为空则需判断左侧的方块 `cells[i - 2]` 是否为空，如果为空将跳跃，则 `cells[i]` 和 `cells[i - 2]` 这两个方块交换图案，并修改它们方块的朝向等属性。

如果 `cells[i]` 是向右的青蛙方块，处理过程类似 `cells[i]` 是向左的青蛙方块。

每次移动成功均要判断游戏是否结束。代码如下：

```
public void mouseClicked(MouseEvent e) {
    //找到相应的动物方块
    int i;
    for (i = 0; i < cells.length; i++) {
        if (cells[i] == e.getSource()) //获取触发事件的对象
            break;
    }
    //如果被单击的是空白方块则无需处理
    if (cells[i].isBlank()) return;
    //向左的青蛙方块
    if (cells[i].isLeft() && cells[i - 1].isBlank()) { //并且前方是空白方块
        Icon icon = null; //动物图标
        icon = cells[i].getIcon();
        cells[i].setIcon(cells[i - 1].getIcon());
        cells[i - 1].setIcon(icon);

        cells[i].setBlank(true); //标记此动物方块为空白图标
        cells[i - 1].setBlank(false); //标记此动物方块为非空白图标
        cells[i - 1].setLeft(true); //标记此动物方块为向左图标
        if (isOver())
            JOptionPane.showMessageDialog(null, "成功", "提示",
                                           JOptionPane.OK_OPTION);
    } else if (cells[i].isLeft() && !cells[i - 1].isBlank()
               && cells[i - 2].isBlank()) { //并且前方第 2 块是空白方块
        Icon icon = null; //动物图标
        icon = cells[i].getIcon();
        cells[i].setIcon(cells[i - 2].getIcon());
        cells[i - 2].setIcon(icon);

        cells[i].setBlank(true); //标记此动物方块为空白图标
        cells[i - 2].setBlank(false); //标记此动物方块为非空白图标
        cells[i - 2].setLeft(true); //标记此动物方块为向左图标
        if (isOver())
            JOptionPane.showMessageDialog(null, "成功", "提示",
                                           JOptionPane.OK_OPTION);
    } else //向右的青蛙方块
        if (!cells[i].isLeft() && cells[i + 1].isBlank()) {
            Icon icon = null; //动物图标
            icon = cells[i].getIcon();
```



```

        cells[i].setIcon(cells[i + 1].getIcon());
        cells[i + 1].setIcon(icon);

        cells[i].setBlank(true); // 标记此动物方块为空白图标
        cells[i + 1].setBlank(false); // 标记此动物方块为非空白图标
        cells[i + 1].setLeft(false); // 标记此动物方块为向右图标
        if (isOver())
            JOptionPane.showMessageDialog(null, "成功", "提示",
                JOptionPane.OK_OPTION);
    } else
        if (!cells[i].isLeft() && !cells[i + 1].isBlank()
            && cells[i + 2].isBlank()) {
            Icon icon = null; // 动物图标
            icon = cells[i].getIcon();
            cells[i].setIcon(cells[i + 2].getIcon());
            cells[i + 2].setIcon(icon);

            cells[i].setBlank(true); // 标记此动物方块为空白图标
            cells[i + 2].setBlank(false); // 标记此动物方块为非空白图标
            cells[i + 2].setLeft(false); // 标记此动物方块为向右图标
            if (isOver())
                JOptionPane.showMessageDialog(null, "成功", "提示",
                    JOptionPane.OK_OPTION);
        }
    }
}

```

18.3.3 设计主窗体类 (frogFrame.java)

在项目中创建一个继承 JFrame 类的主窗体类 frogFrame，在该类中分别创建游戏面板 frogPanel 类的实例 gamePanel 和一个“开始”按钮 button。在“开始”按钮 button 的单击事件中调用 gamePanel.init(n) 方法初始化游戏界面从而开始游戏。在游戏失败后可以单击“开始”按钮将动物方块恢复到初始位置，重新开始游戏。代码如下：

```

import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class frogFrame extends JFrame {
    private int n=6; // 动物数量

    public static void main(String args[]) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    frogFrame frame = new frogFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}

```



```
public frogFrame() {
    super();
    getContentPane().setLayout(new BorderLayout());
    setTitle("青蛙跳游戏");
    setBounds(300, 300, 758, 314);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    final JPanel panel = new JPanel();          //实例化 JPanel
    getContentPane().add(panel, BorderLayout.NORTH);    //添加到上方
    final frogPanel gamePanel = new frogPanel();    //实例化游戏面板
    //添加到中央位置
    getContentPane().add(gamePanel, BorderLayout.CENTER);
    final JButton button = new JButton();        //实例化按钮
    //注册事件
    button.addActionListener(new ActionListener() {
        public void actionPerformed(final(ActionEvent) e) {
            //开始游戏
            gamePanel.init(n);
            gamePanel.repaint();
        }
    });
    button.setText("开始");
    panel.add(button);
}
```

游戏成功的运行界面如图 18-3 所示。

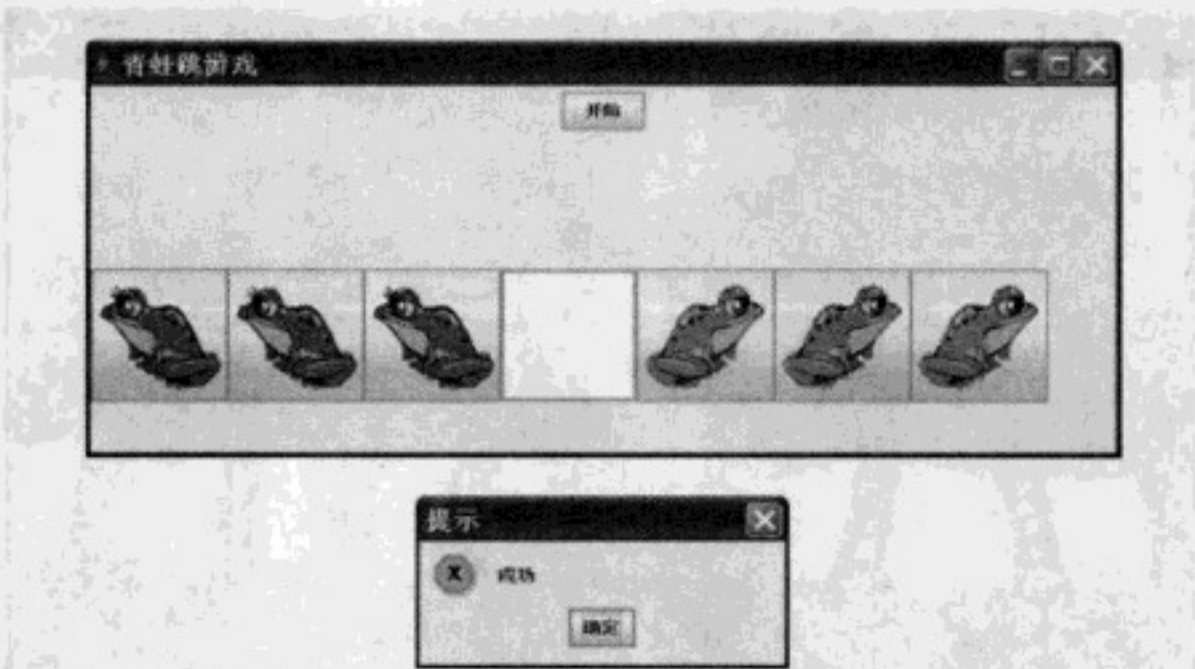


图 18-3 青蛙跳游戏成功运行的界面

当然本游戏也可以是 8 只青蛙分成两队，左边 4 只，右边 4 只，只需要做如下修改：

```
private int n=8;//动物数量
```

即可以在调用 gamePanel.init(n)方法时初始化成 8 只青蛙游戏界面，如图 18-4 所示。

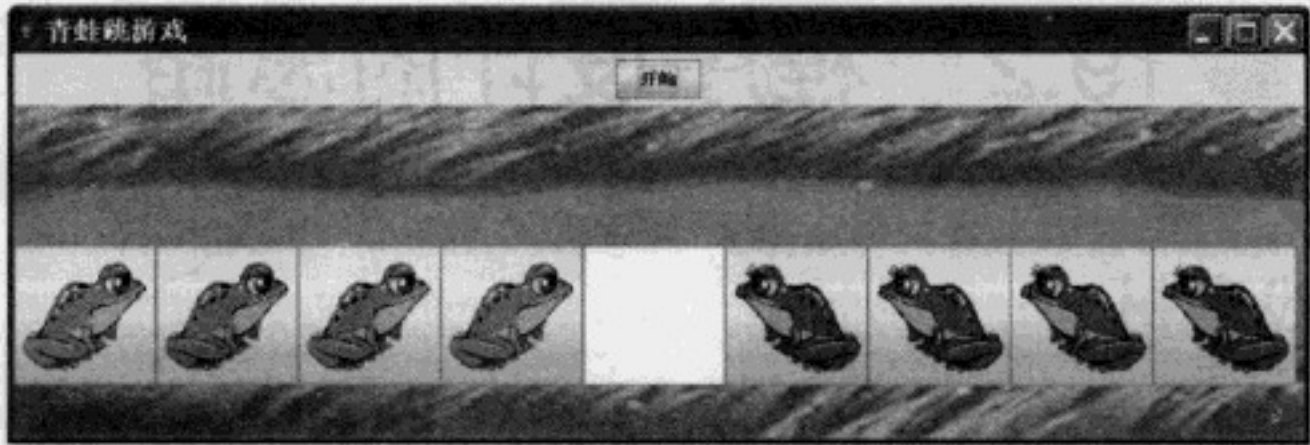


图 18-4 8 只青蛙跳游戏界面

第 19 章

打 猎 游 戏

19.1 打猎游戏介绍

打猎游戏程序界面底部会有野猪随机出现，并以不固定的速度移动；上方有小鸟以反方向飞过。当用鼠标在它们身上进行单击操作时，如果打中该动物，则动物消失，在界面左上角得到相应分数，但是如果动物跑出界面，存戏就会扣除一定的分数。

另外，界面右上角会显示当前剩余子弹的数量，如无子弹，需要等待系统装载子弹。打猎运行的运行界面，如图 19-1 所示。

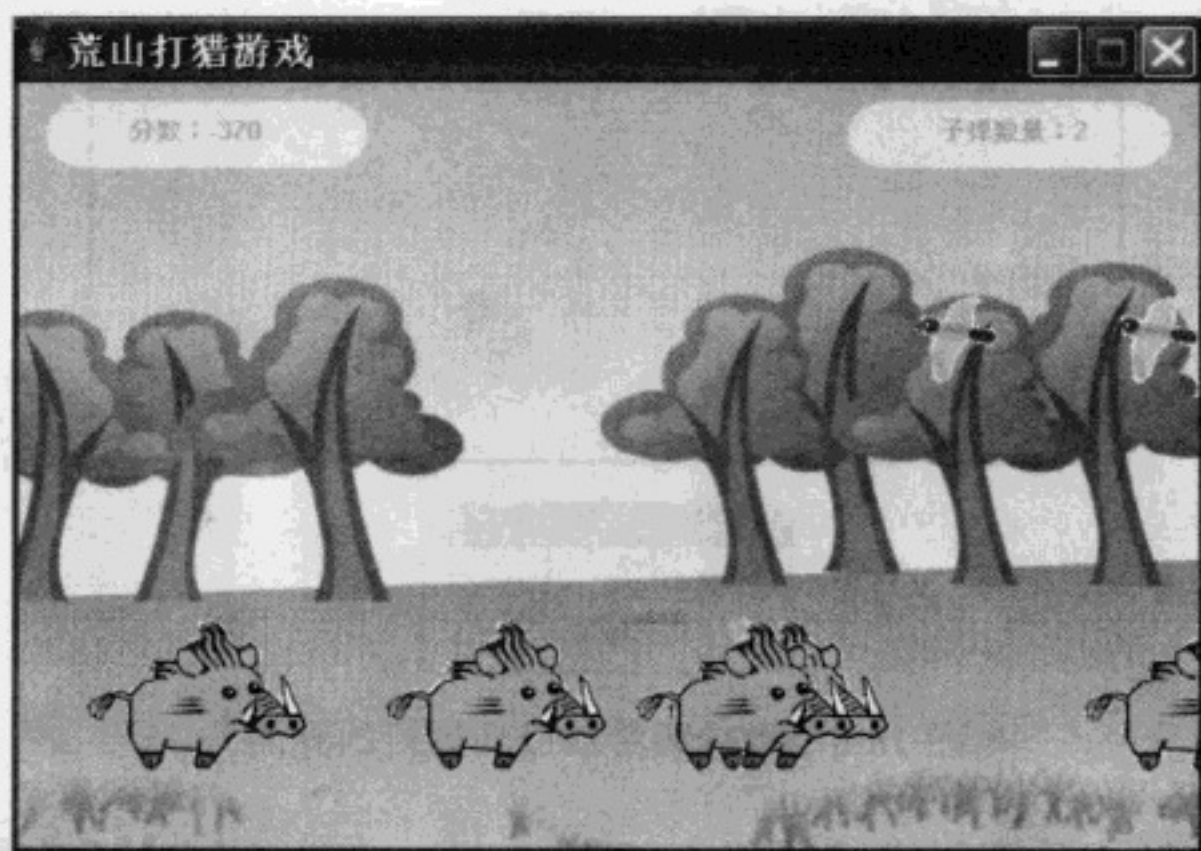


图 19-1 打猎游戏运行界面

19.2 程序设计的思路

19.2.1 游戏素材

打猎游戏程序中用到森林、小鸟、野猪等，分别使用图 19-2 所示的图片来表示。注意.GIF 格式的图片本身就具有动画效果。



图 19-2 相关图片素材

19.2.2 设计思路

使用一个带背景（background.jpg）的面板作为森林，在其上面显示小鸟、野猪对象。小鸟、野猪对象分别使用继承 JLabel 类的 BirdLabel 标签类、继承 JLabel 类的 PigLabel 标签类来实现。创建一个继承 JFrame 类的主窗体类 MainFrame，使用两个线程分别生成小鸟和野猪角色对象，并且能响应鼠标事件。

19.3 关键技术

19.3.1 控制动物组件的移动速度

本程序的难点在于控制动物组件的移动速度。如果每个动物移动的速度相同，就会使程序运行效果枯燥乏味，没有游戏难度也就失去了游戏进行下去的意义，因此需要在线程中控制每个组件的移动速度。在线程循环中，可以通过随机数来确定新创建的动物组件移动线程的休眠时间，这样就可以为每个动物组件设置不同的移动速度。

例如，小鸟组件的实现代码如下：

```
public class BirdLabel extends JLabel implements Runnable {
    //随机生成线程的休眠时间，即控制小鸟组件的移动速度
    private int sleepTime = (int) (Math.random() * 300) + 5;
    private int y = 100;
    private Thread thread;//将线程作为成员变量
    public void run() {
        parent = null;
        int width = 0;
        try {
            while (width <= 0 || parent == null) {
                if (parent == null){
                    parent = getParent();//获取父容器
                } else {
                    width = parent.getWidth();//获取父容器的宽度
                }
                Thread.sleep(10);
            }
            for (int i = width; i > 0 && parent != null; i -= 8) {
                setLocation(i, y);//从右向左移动本组件的位置
                Thread.sleep(sleepTime);//休眠片刻
            }
        }
    }
}
```



```

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

在 run()方法中实现不断地从右向左移动小鸟组件的位置，每次移动 8 个像素。但是由于每个小鸟组件的休眠时间不一致，因此产生了移动速度不同的效果。

野猪组件的移动速度控制原理与小鸟组件同理。

19.3.2 随机间歇产生动物组件

在 MainFrame 窗体类的内部线程类 PigThread 中，调用了 Math 类的 random()方法随机确定线程休眠的时间，这样可以不定时产生野猪角色。窗体类 MainFrame 的内部线程类 PigThread（生成野猪角色的线程）的代码如下：

```

/**
 * 生成野猪角色的线程
 */
class PigThread extends Thread {
    @Override
    public void run() {
        while (true) {
            //创建代表野猪角色的标签组件
            PigLabel pig = new PigLabel();
            pig.setSize(120, 80); //设置组件初始大小
            backgroundPanel.add(pig); //添加组件到背景面板
            try {
                //线程随机休眠一段时间
                sleep((long) (random() * 3000) + 500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```

窗体类 MainFrame 的内部线程类 BirdThread（生成小鸟角色的线程）的代码如下：

```

/**
 * 生成小鸟角色的线程
 */
class BirdThread extends Thread {
    @Override
    public void run() {
        while (true) {
            //创建代表小鸟的标签组件
            BirdLabel bird = new BirdLabel();
            bird.setSize(50, 50); //设置组件初始大小
            backgroundPanel.add(bird); //添加组件到背景面板
            try {
                //线程随机休眠一段时间
                sleep((long) (Math.random() * 3000) + 500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```



```

    }
    }
}

```

上段代码的 `random()` 方法是 `Math` 类的静态方法，因此在 `MainFrame` 类的包引用位置使用了 `import static java.lang.Math.random;` 导入语句。

19.3.3 玻璃面板显示

`JFrame` 由根面板、一个玻璃面板（`glassPane`）、分层面板结合而成，分层面板由一个内容面板（`contentPane`）和一个可选择的菜单条（`JMenuBar`）组成，而内容面板和可选择的菜单条放在同一分层。玻璃面板是完全透明的，默认值为不可见，为接收鼠标事件和在所有组件上绘图提供方便。

`JFrame` 提供的相关方法如下：

```

Container getContentPane(); //获得内容面板
setContentPane(Container); //设置内容面板
JMenuBar getMenuBar(); //获取菜单条
setMenuBar(JMenuBar); //设置菜单条
JLayeredPane getLayeredPane(); //获得分层面板
setLayeredPane(JLayeredPane); //设置分层面板
Component getGlassPane(); //获得玻璃面板
setGlassPane(Component); //设置玻璃面板

```

本游戏中子弹信息的提示采用了玻璃面板，玻璃面板内用一个标签显示子弹信息。代码如下：

```

infoPane = (JPanel) getGlassPane(); //获取玻璃面板
JLabel label = new JLabel("装载子弹……"); //创建提示标签组件
label.setHorizontalAlignment(SwingConstants.CENTER);
label.setFont(new Font("楷体", Font.BOLD, 32));
label.setForeground(Color.ORANGE);
infoPane.setLayout(new BorderLayout());
infoPane.add(label); //添加提示标签组件到玻璃面板

```

玻璃面板是完全透明的，默认值为不可见。因此玻璃面板 `infoPane` 的显示由 `infoPane.setVisible(true)` 来控制。

19.4 打猎游戏设计的步骤

19.4.1 设计小鸟类（`BirdLabel.java`）

在项目中创建一个继承 `JLabel` 类的 `BirdLabel` 标签类，用于表示小鸟，并且实现 `Runnable` 接口。通过线程控制小鸟的移动效果，以及实现扣分功能。

导入包及相关类：

```

import java.awt.Container;
import java.awt.event.*;
import javax.swing.*;

```


BirdLabel 标签类通过 Runnable 接口使用线程。BirdLabel 标签类的构造方法中，首先创建小鸟图标对象，并将其设置为标签组件的图标。然后为该组件添加鼠标事件监听器以及组件事件监听器。最后将此 BirdLabel 实例 this 作为参数来创建线程对象。代码如下：

```
public class BirdLabel extends JLabel implements Runnable {
    //随机生成线程的休眠时间，即控制小鸟移动的速度
    private int sleepTime = (int) (Math.random() * 300) + 5;
    private int y = 100;
    private Thread thread;//将线程作为成员变量
    private Container parent;
    private int score = 15;//该类角色对应的分数
    /**
     * 构造方法
     */
    public BirdLabel() {
        super();
        //创建小鸟图标对象
        ImageIcon icon = new ImageIcon(getClass().getResource("bird.gif"));
        setIcon(icon);//设置组件图标
        addMouseListener(new MouseAction());//添加鼠标事件监听器
        //添加组件事件监听器
        addComponentListener(new ComponentAction());
        thread = new Thread(this);//创建线程对象
    }
    /**
     * 组件的组件事件监听器
     */
    private final class ComponentAction extends ComponentAdapter {
        public void componentResized(final ComponentEvent e) {
            thread.start();//线程启动
        }
    }
    /**
     * 组件的鼠标事件监听器
     */
    private final class MouseAction extends MouseAdapter {
        public void mousePressed(final MouseEvent e) {
            if (!MainFrame.readyAmmo())//如果子弹没有准备好
                return;//什么也不做
            MainFrame.useAmmo();//消耗子弹
            appScore();//加分
            destory();//销毁本组件
        }
    }
}
```

在 run()方法中实现不断地从右向左移动小鸟组件的位置，每次移动 8 个像素。但是由于每个小鸟组件的休眠时间不一致，因此产生了移动速度不同的效果。代码如下：

```
public void run() {
    parent = null;
    int width = 0;
    try {
```



```

        while (width <= 0 || parent == null) {
            if (parent == null){
                parent = getParent();//获取父容器
            } else {
                width = parent.getWidth();//获取父容器的宽度
            }
            Thread.sleep(10);
        }
        for (int i = width; i > 0 && parent != null; i -= 8) {
            setLocation(i, y);//从右向左移动本组件的位置
            Thread.sleep(sleepTime);//休眠片刻
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    if (parent != null) {
        MainFrame.appScore(-score * 10); //自然销毁将扣分
    }
    destory();//移动完毕，销毁本组件
}

```

destory()方法实现从父容器中移除本组件 Label。代码如下：

```

public void destory() {
    if (parent == null)
        return;
    parent.remove(this);//从父容器中移除本组件
    parent.repaint();
    parent = null; //通过该语句终止线程循环
}

```

appScore()是实现游戏加分的方法。代码如下：

```

private void appScore() {
    MainFrame.appScore(15);
}

```

19.4.2 设计野猪类 (PigLabel.java)

项目中创建一个继承 JLabel 类的 PigLabel 标签类，用于表示野猪，并且实现 Runnable 接口。通过线程控制野猪的移动效果，以及实现扣分功能。代码基本与继承 JLabel 类的 BirdLabel 标签类相似。

```

import java.awt.Container;
import java.awt.event.*;
import javax.swing.*;

```

PigLabel 标签类通过 Runnable 接口使用线程。PigLabel 标签类的构造方法中，创建野猪图标对象，并将其设置为标签组件的图标。然后为该组件添加鼠标事件监听器以及组件事件监听器。最后将此 PigLabel 实例 this 作为参数来创建线程对象。代码如下：

```

public class PigLabel extends JLabel implements Runnable {
    //随机生成线程的休眠时间，即控制野猪移动的速度
    private int sleepTime = (int) (Math.random() * 300) + 30;
    private int y = 260;//组件的垂直坐标
}

```



```

private int score = 10;//该类角色对应的分数
private Thread thread;//内置线程对象
private Container parent;//组件的父容器对象
/**
 * 构造方法
 */
public PigLabel() {
    super();
    ImageIcon icon = new ImageIcon(getClass().getResource(
        "pig.gif"));//加载野猪图片
    setIcon(icon);//设置本组件的图标
    //添加鼠标事件适配器
    addMouseListener(new MouseAdapter() {
        //按下鼠标按键的处理方法
        public void mousePressed(final MouseEvent e) {
            if (!MainFrame.readyAmmo())
                return;
            MainFrame.useAmmo();//消耗子弹
            appScore();//给游戏加分
            destory();//销毁本组件
        }
    });
    //添加组件事件适配器
    addComponentListener(new ComponentAdapter() {
        //调整组件大小
        public void componentResized(final ComponentEvent e) {
            thread.start();//启动线程
        }
    });
    thread = new Thread(this);    //初始化线程对象
}
public void run() {
    parent = null;
    int width = 0;
    while (width <= 0 || parent == null) { //获取父容器宽度
        if (parent == null)
            parent = getParent();
        else
            width = parent.getWidth();
    }
    //从左向右移动本组件
    for (int i = 0; i < width && parent != null; i += 8) {
        setLocation(i, y);
        try {
            Thread.sleep(sleepTime);//休眠片刻
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    if (parent != null) {
        MainFrame.appScore(-score * 10); //自然销毁将扣分
    }
}

```



```

    }
    destory();
}
public void destory() { //从容器中移除本组件的方法
    if (parent == null)
        return;
    parent.remove(this);
    parent.repaint();
    parent = null; //通过该语句终止线程循环
}
private void appScore() { //加分的方法
    MainFrame.appScore(20);
}
}

```

19.4.3 设计背景面板类 (BackgroundPanel.java)

背景面板类实现在面板上显示森林图片的背景。代码如下:

```

import java.awt.Graphics;
import java.awt.Image;
import javax.swing.JPanel;
public class BackgroundPanel extends JPanel {
    private Image image; //背景图片
    public BackgroundPanel() {
        setOpaque(false); //设置为透明
        setLayout(null);
    }
    public void setImage(Image image) {
        this.image = image;
    }
    /**
     * 画出背景
     */
    protected void paintComponent(Graphics g) {
        if (image != null) {
            int width = getWidth(); //图片宽度
            int height = getHeight(); //图片高度
            g.drawImage(image, 0, 0, width, height, this); //画出图片
        }
        super.paintComponent(g);
    }
}

```

19.4.4 设计主窗体类 (MainFrame.java)

在项目中创建一个继承 JFrame 类的主窗体类 MainFrame, 在该类中分别创建生成小鸟和小猪角色的内部线程类。代码如下:

```

import static java.lang.Math.random;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MainFrame extends JFrame {

```



```

private static long score = 0;//分数
private static Integer ammoNum = 5;//子弹数量
private static JLabel scoreLabel;//分数
private BackgroundPanel backgroundPanel;
private static JLabel ammoLabel;
private static JPanel infoPane;

```

在 `MainFrame()` 构造方法中首先进行窗体大小调整，子弹信息的提示采用了玻璃面板，玻璃面板内用一个标签 `label` 显示子弹信息。其次创建带背景的面板 `backgroundPanel`，在其中添加鼠标事件适配器。最后添加显示子弹数量的标签组件。代码如下：

```

public MainFrame() {
    super();
    setResizable(false);//调整窗体大小
    setTitle("打猎游戏");
    infoPane = (JPanel) getGlassPane();//获取玻璃面板
    JLabel label = new JLabel("装载子弹……");//创建提示标签组件
    label.setHorizontalAlignment(SwingConstants.CENTER);
    label.setFont(new Font("楷体", Font.BOLD, 32));
    label.setForeground(Color.ORANGE);
    infoPane.setLayout(new BorderLayout());
    infoPane.add(label);//添加提示标签组件到玻璃面板

    setAlwaysOnTop(true);//是窗体则保持在最顶层
    setBounds(100, 100, 573, 411);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    backgroundPanel = new BackgroundPanel();//创建带背景的面板
    backgroundPanel.setImage(new ImageIcon(getClass()
        .getResource("background.jpg")).getImage());//设置背景图片
    getContentPane().add(backgroundPanel,
        BorderLayout.CENTER);
    //添加鼠标事件适配器
    addMouseListener(new FrameMouseListener());
    scoreLabel = new JLabel();//显示分数的标签组件
    scoreLabel.setHorizontalAlignment(SwingConstants.CENTER);
    scoreLabel.setForeground(Color.ORANGE);
    scoreLabel.setText("分数: ");
    scoreLabel.setBounds(25, 15, 120, 18);
    backgroundPanel.add(scoreLabel);
    ammoLabel = new JLabel();//显示子弹数量的标签组件
    ammoLabel.setForeground(Color.ORANGE);
    ammoLabel.setHorizontalAlignment(SwingConstants.RIGHT);
    ammoLabel.setText("子弹数量: " + ammoNum);
    ammoLabel.setBounds(422, 15, 93, 18);
    backgroundPanel.add(ammoLabel);
}

appScore(int num) 方法实现游戏加分功能。代码如下:
public synchronized static void appScore(int num) {
    score += num;
    scoreLabel.setText("分数: " + score);
}

```


useAmmo()方法实现消耗子弹的功能。子弹数量递减后判断是否小于 0，如果小于 0，则显示提示信息面板 infoPane，程序等待 1 秒钟的装载子弹时间，恢复子弹数量为 5 颗，之后隐藏提示信息面板。代码如下：

```
public synchronized static void useAmmo() { //消耗子弹
    synchronized (ammoNum) {
        ammoNum--; //子弹数量递减
        ammoLabel.setText("子弹数量: " + ammoNum);
        if (ammoNum <= 0) { //判断子弹数量是否小于 0
            new Thread(new Runnable() {
                public void run() {
                    //显示提示信息面板
                    infoPane.setVisible(true);
                    try {
                        // 1 秒钟装载子弹的时间
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    ammoNum = 5; //恢复子弹数量
                    //修改子弹数量标签的文本
                    ammoLabel.setText("子弹数量: " + ammoNum);
                    infoPane.setVisible(false); //隐藏提示信息面板
                }
            }).start();
        }
    }
}
```

readyAmmo()方法判断子弹是否够用。代码如下：

```
public synchronized static boolean readyAmmo() {
    synchronized (ammoNum) {
        return ammoNum > 0;
    }
}
```

内部类 class FrameMouseListener 实现窗体的鼠标事件监听器。代码如下：

```
private final class FrameMouseListener extends MouseAdapter {
    public void mousePressed(final MouseEvent e) {
        Component at = backgroundPanel.getComponentAt(e.getPoint());
        if (at instanceof BackgroundPanel) { //如果鼠标击到面板也扣除子弹
            MainFrame.useAmmo(); //消耗子弹
        }
    }
}
```

main(String args[])是游戏程序启动的主方法。它调用 start()方法启动生成野猪角色的线程和生成小鸟角色的线程。代码如下：

```
public static void main(String args[]) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                MainFrame frame = new MainFrame();
```



```

        frame.setVisible(true);
        frame.start();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
});
}

```

start()方法启动生成野猪角色的线程和生成小鸟角色的线程。代码如下:

```

public void start() {
    new PigThread().start();
    new BirdThread().start();
}

```

生成野猪角色的线程类代码如下所示:

```

class PigThread extends Thread {
    @Override
    public void run() {
        while (true) {
            //创建代表野猪的标签组件
            PigLabel pig = new PigLabel();
            pig.setSize(120, 80); //设置组件的初始大小
            backgroundPanel.add(pig); //添加组件到背景面板
            try {
                //线程随机休眠一段时间
                sleep((long) (random() * 3000) + 500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```

生成小鸟角色的线程类代码如下所示:

```

class BirdThread extends Thread {
    @Override
    public void run() {
        while (true) {
            //创建代表小鸟的标签组件
            BirdLabel bird = new BirdLabel();
            bird.setSize(50, 50); //设置组件初始大小
            backgroundPanel.add(bird); //添加组件到背景面板
            try {
                //线程随机休眠一段时间
                sleep((long) (Math.random() * 3000) + 500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```


第 20 章

2.5D 游戏

20.1 2.5D 游戏介绍

通常的概念中，2D 也就是所谓的二维，即平面图形——由 x 与 y 坐标构成的图形，其内容由水平的 x 轴向与垂直的 y 轴向描绘而确定，也就是由长和高形成的所谓二维平面。

而所谓 3D，也称之为三维。其图形内容除了有水平的 x 轴向与垂直的 y 轴向外还有进深的 z 轴向，故称三维（XYZ），也就是由长、宽、高三项要素形成所谓的三维立体。

2D 与 3D 的主要区别在于：3D 可以包含 360° 的信息，能从各个角度去表现，构成近似于现实空间的有质感视角；而 2D 通常只能表现如表格、棋盘版的平面数据。3D 的立体感、光景效果要比二维平面图形要好得多，因为它的立体、光线、阴影都是相对真实存在的，而 2D 显然不具备这些优势。高拟真度、高自由度使得 3D 图形大受欢迎。

因此渐渐地 3D 图形开始成为主流，充斥于电影、电视乃至游戏的各个角落。但是，由于 3D 技术实现的复杂性及对用户环境的高要求，在所有领域都完全使用 3D 构图还并不现实，由此引发了另外一种图形表现形式的出现——2.5D 图形。

所谓的 2.5D，介乎于模真的 3D 与完全平面的 2D 之间。既模拟了 3D 的空间感，也兼具 2D 的灵动简单，是一种“优势”的综合体。诚然 2.5D 最早的出现动机只是为了 2D 到 3D 间的过渡。但就其应用而讲，好的 2.5D 图形既有 3D 的自由度与质感，又能利用 2D 图形将漫画式人物塑造得惟妙惟肖，使其拥有纯 3D 还无法做到的优势。因此 2.5D 在现在乃至未来的一段较长时间里还会和 3D 并存，直到 3D 图形的开发效率及表现形式能彻底取代 2.5D 为止。利用斜 45° 的 2D 图片可以得到 3D 效果，图 20-1 是一张 2.5D 的实际效果图。

本章采用 2.5D 技术开发立体效果的推箱子游戏，游戏开始后的界面如图 20-2 所示，玩家可以通过方向键控制人物移动箱子到目的地（蓝色方块）。

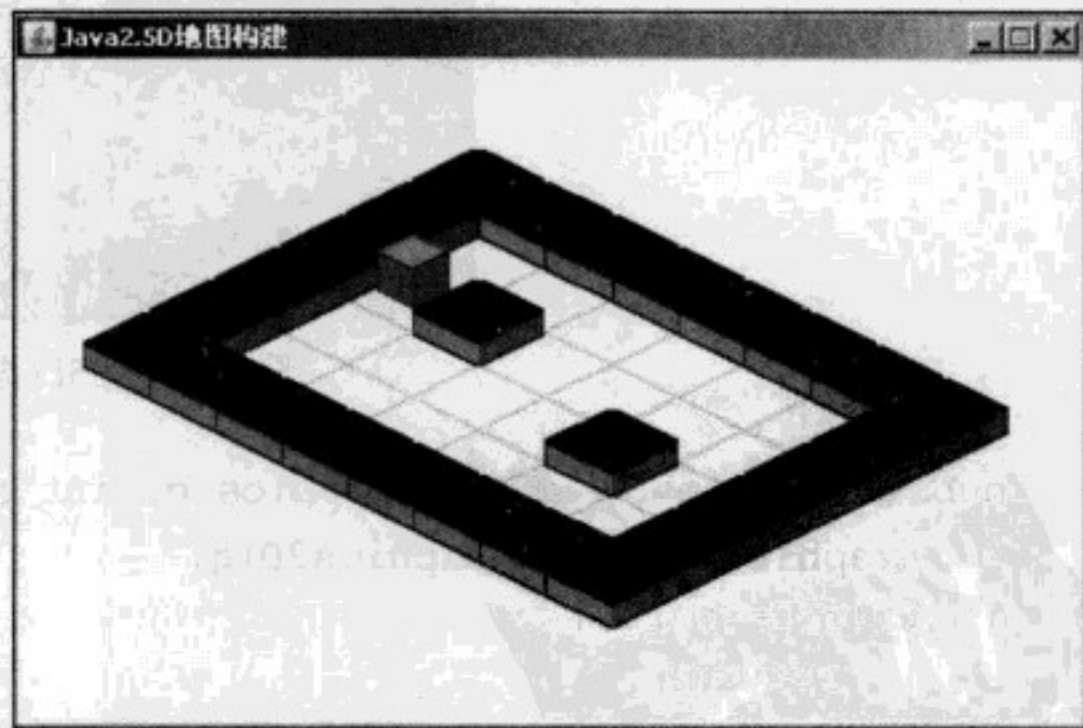


图 20-1 2.5D 的效果图

目的地

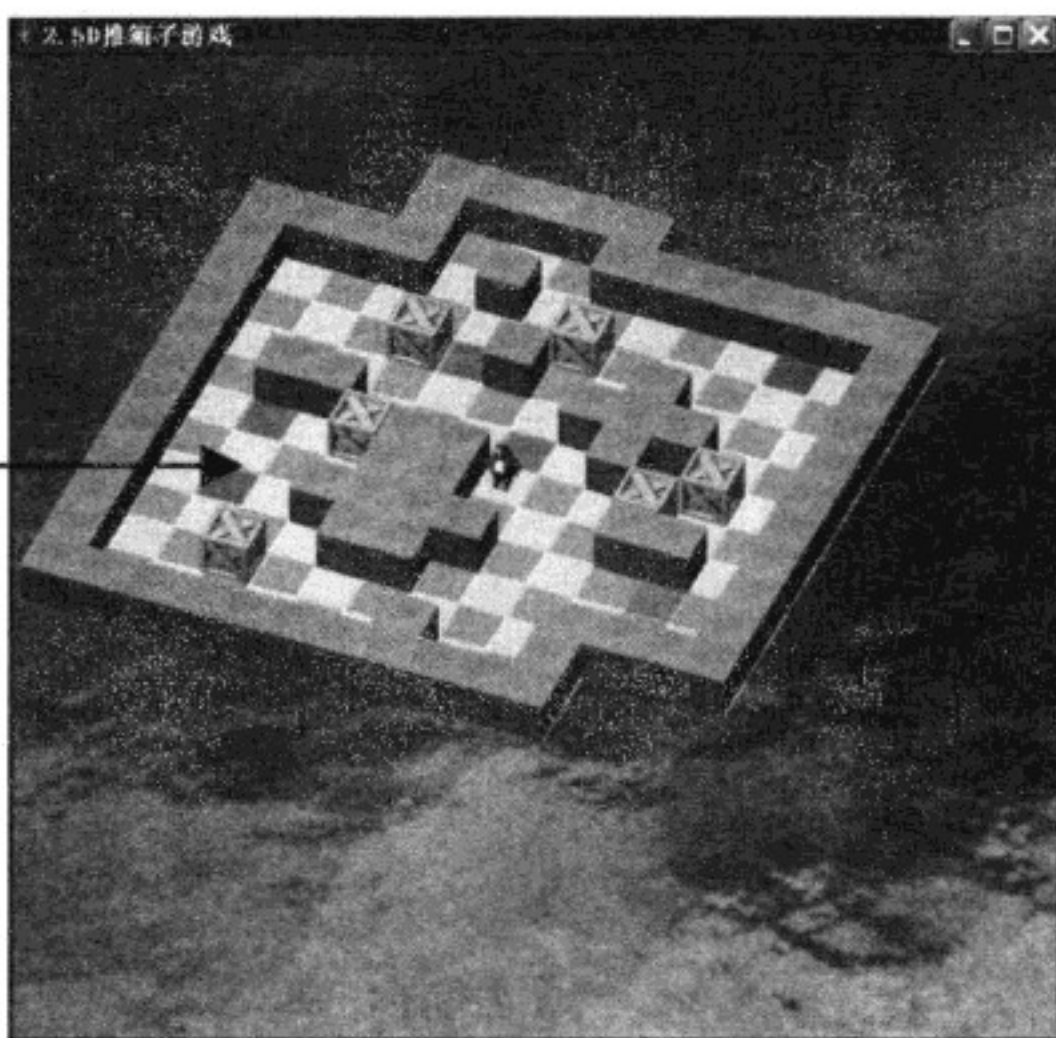


图 20-2 游戏开始后的界面

20.2 2.5D 游戏设计思路

在 2.5D 推箱子游戏中利用斜 45°视角，如果要产生图 20-3 所示 14×14 的网格，则需要绘制地图中的多边形，而不是矩形，其代码如下：

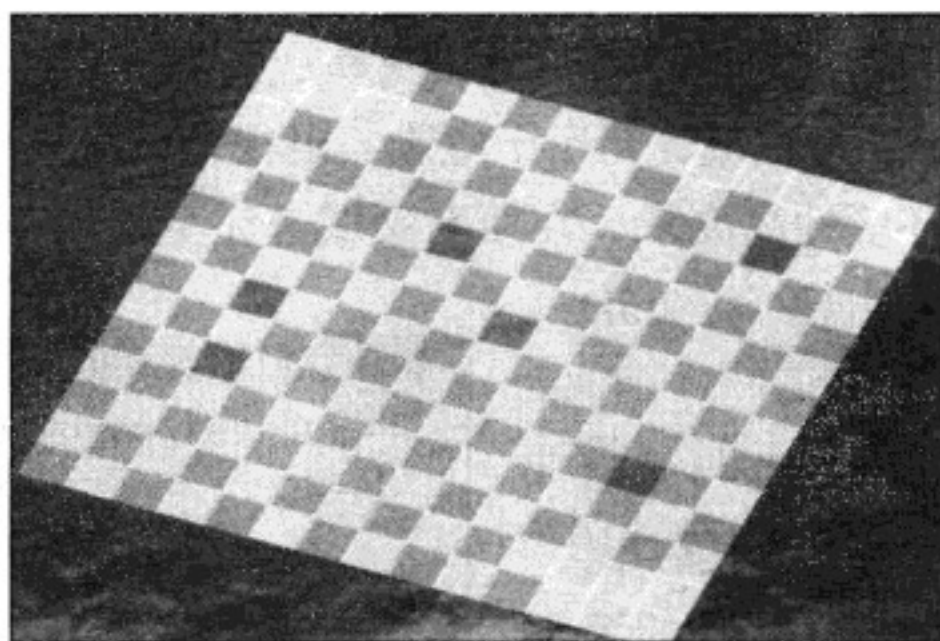


图 20-3 2.5D 游戏的网格界面

```
public void myDrawRect(Graphics g, int x, int y) { // 绘制多边形
    Graphics2D g2D = (Graphics2D) g;
    if (g2D == null) {
        return;
    }
    GeneralPath path = new GeneralPath();
    path.moveTo(x + 14, y);
    path.lineTo(x + 53, y + 10);
    path.lineTo(x + 37, y + 37);
    path.lineTo(x - 2, y + 26);
    path.lineTo(x + 14, y);
    g2D.fill(path); // g.draw(myPath);
}
```


对于图 20-3 所示的网格界面采用 map1 数组存储,列序号向右逐次增加,行序号向左逐次增加,如图 20-4 所示。

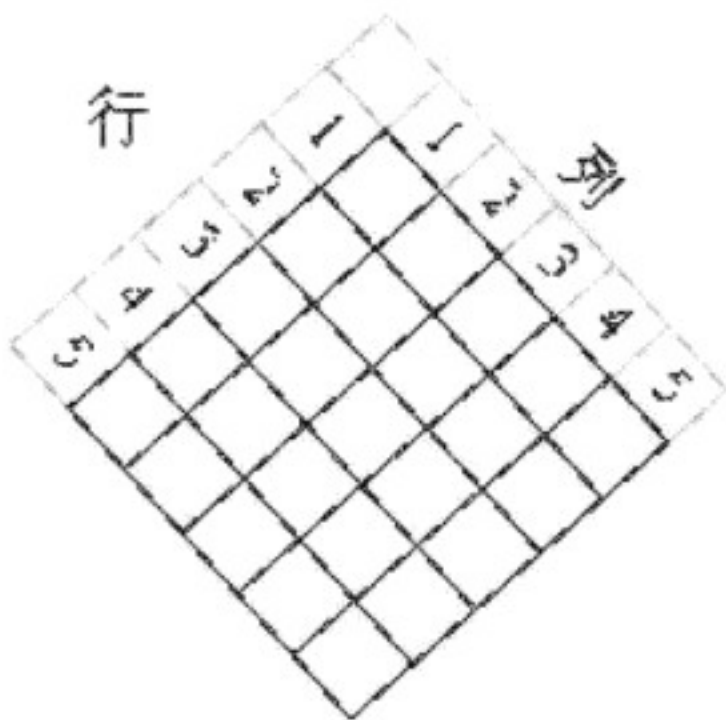


图 20-4 数组存储示意图

由于 2.5D 呈现技术与 2D 的绘制方式基本相同,在绘制时只需注意绘制的顺序即可。本游戏中地图元素采用的是等大图元(每个物件大小相同),因此可以先绘制底层地图,再绘制上层(建筑物)地图。这里采用两个数组 map1、map2 分别表示地图的两层,第一层数组 map1 中 0 表示白色空地,1 表示灰色空地,2 表示目的地。第二层数组 map2 中 1 表示箱子,2 表示墙,3 表示绿色的箱子。第一层和第二层地图中的-1 代表该处没有任何地图元素。代码如下:

//map1 为第一层地图, map2 为第二层地图

```
private int [][]map1={//第一层地图，即地板层
    {-1,-1,-1,1, 0, 1, 0, 1,-1,-1,-1,-1,-1},
    {-1,-1,-1,0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
    {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 2, 0, 1},
    {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
    {0, 1, 0, 1, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1},
    {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
    {0, 1, 0, 1, 0, 1, 0, 2, 0, 1, 0, 1, 0, 1},
    {1, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
    {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1},
    {1, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1, 2, 1, 0},
    {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1},
    {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
    {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,-1,-1,-1},
    {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,-1,-1,-1}
};

private int [][]map2={//第二层地图，即建筑物
    {-1,-1,-1,2, 2, 2, 2, 2,-1,-1,-1,-1,-1,-1},
    {-1,-1,-1,2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2},
    {2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2},
    {2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2},
    {2, 0, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0, 0, 2},
    {2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 2},
    {2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 1, 0, 2},
    {2, 0, 0, 0, 1, 2, 2, 0, 0, 0, 1, 0, 0, 2},
    {2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2},
    {2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 2},
    {2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2}
};
```



```
        {2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2},
        {2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2,-1,-1,-1},
        {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,-1,-1,-1}
    };
```

2.5D 游戏中的图片为了配合倾斜后的 x, y 坐标拼接，大部分平面图必须转换为 45° 图。图 20-5 是游戏中人物和箱子的 45° 图。这样游戏中不需要对图片再进行变换。虽然也可以在平面图上自动换算出所需的斜视图形，但细节处通常不够理想，而且耗费不必要的运算资源，还是交给美术人员直接制作出成品图最好。



图 20-5 人物和箱子的 45° 图

注意在 2D 游戏中箱子是平面图，2.5D 游戏中的图片是 45° 图。图 20-6 是箱子平面图和 45° 图的对比。



图 20-6 箱子平面图和 45° 图的对比

20.3 程序设计的步骤

20.3.1 创建游戏界面类（PushBox.java）

在项目中创建一个继承 JPanel 的 PushBox 类，用于显示游戏界面和实现游戏逻辑。引入的包和相关的类：

```
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.geom.GeneralPath;
import javax.swing.JPanel;
```

PushBox 类实现 KeyListener 接口，从而监听键盘事件，同时定义一些成员变量。代码如下：

```
public class PushBox extends JPanel implements KeyListener{
    private Image pic[] = null; //图片
    int initX=200,initY=70;
    //map1 为第一层地图，map2 为第二层地图
    private int [][]map1={//第一层地图，即地板层
        {-1,-1,-1,1, 0, 1, 0, 1,-1,-1,-1,-1,-1,-1},
        {-1,-1,-1,0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 2, 0, 1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
    };
```



```

        {0, 1, 0, 1, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 2, 0, 1, 0, 1, 0, 1},
        {1, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1},
        {1, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1, 2, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, -1, -1, -1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, -1, -1, -1}
    };

    private int [][]map2={//第二层地图,即建筑物
        {-1,-1,-1,2, 2, 2, 2, 2,-1,-1,-1,-1,-1,-1},
        {-1,-1,-1,2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2},
        {2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2},
        {2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2},
        {2, 0, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0, 0, 2},
        {2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 2},
        {2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 1, 0, 2},
        {2, 0, 0, 0, 1, 2, 2, 0, 0, 0, 1, 0, 0, 2},
        {2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2},
        {2, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 2},
        {2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2},
        {2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2},
        {2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, -1, -1, -1},
        {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, -1, -1, -1}
    };

    //定义一些常量,对应地图的元素
    final byte WALL = 2, BOX = 1, BOXONEND = 3, END = 2,
        WhiteGRASS = 0, BlackGRASS = 1;
    private int row = 7, column = 7;
    //加载图片
    Image box=Toolkit.getDefaultToolkit().getImage("images\\box.png");
    Image wall = Toolkit.getDefaultToolkit().getImage("images\\wall.png");
    Image greenBox= Toolkit.getDefaultToolkit().getImage("images\\greenbox.png");
    Image man = Toolkit.getDefaultToolkit().getImage("images\\a1.png");//人物
    Image background= Toolkit.getDefaultToolkit().getImage("images\\background.jpg");

```

PushBox 类的构造方法设定焦点在本面板,并将本面板作为监听对象。代码如下:

```

public PushBox() {
    setFocusable(true);           // 设置焦点
    this.addKeyListener(this);
}

```

myDrawRect(Graphics g, int x, int y)方法绘制游戏网格中的多边形。代码如下:

```

public void myDrawRect(Graphics g, int x, int y){//绘制多边形
    Graphics2D g2D=(Graphics2D)g;
    if(g2D==null){
        return;
    }
    GeneralPath path = new GeneralPath();
    path.moveTo(x+14, y);
    path.lineTo(x+53, y+10);
    path.lineTo(x+37, y+37);
    path.lineTo(x-2, y+26);
}

```



```

        path.lineTo(x+14, y);
        g2D.fill(path); //g.draw(myPath);
    }

```

paint(Graphics g)方法画游戏界面。2.5D 游戏需注意绘制的顺序。本游戏中地图元素采用的是等大图元（每个物件大小相同），因此可以先绘制底层地图，再绘制上层（建筑物）地图。代码如下：

```

public void paint(Graphics g) {
    g.clearRect(0,0,this.getWidth(),getHeight());
    g.setColor(Color.BLACK);
    g.drawImage(background, 0, 0,800,800,this);//画游戏背景
    //绘制第一层地图，即地板层
    //WhiteGRASS = 0, END = 2,BlackGRASS = 1;
    for(int i=0; i<map1.length; i++){
        for(int j=0; j<map1[i].length; j++){
            //根据索引值进行坐标转换
            int X = initX+36*j-15*i;
            int Y = initY+10*j+25*i;
            if(map1[i][j] == WhiteGRASS){//白色空地
                /*设置 paint 的颜色*/
                g.setColor(new Color(255, 220, 220, 220));
                this.myDrawRect(g, X, Y);
            }
            else if(map1[i][j] == BlackGRASS){//灰色空地
                g.setColor(new Color(255, 170, 170, 170));
                this.myDrawRect(g, X, Y);
            }
            else if(map1[i][j] == END){//目的地
                g.setColor(new Color(255, 60, 255, 120));
                this.myDrawRect(g, X, Y);
            }
        }
    }
    //开始绘制第二层地图，即建筑物所在层
    for(int i=0; i<map2.length; i++){
        for(int j=0; j<map2[i].length; j++){
            //根据索引值进行坐标转换
            int X = initX+36*j-15*i;
            int Y = initY+10*j+25*i;
            if(map2[i][j] == BOX){//第二层上有箱子处
                g.drawImage(box, X-1, Y-27,this);
            }
            else if(map2[i][j] == WALL){//墙
                g.drawImage(wall, X, Y-25,this);
            }
            else if(map2[i][j] == BOXONEND){//目的地的绿色箱子
                g.drawImage(greenBox, X-1, Y-27,this);
            }
        }
        //绘制人
        if(i == row && j == column){
            g.drawImage(man, X-1, Y-27,this);
        }
    }
}

```



```
}
```

以下是键盘事件，根据用户按键处理人物的移动。当人物角色移动时，此处暂不考虑是否符合游戏规则，例如人物碰到墙不能移动。最后调用 `repaint()` 方法重新绘制窗体。代码如下：

```
public void keyPressed(KeyEvent e) {
    // TODO Auto-generated method stub
    if (e.getKeyCode() == KeyEvent.VK_UP) {    //向上
        moveUp();
    }
    if (e.getKeyCode() == KeyEvent.VK_DOWN) {    //向下
        moveDown();
    }
    if (e.getKeyCode() == KeyEvent.VK_LEFT) {    //向左
        moveLeft();
    }
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) {    //向右
        moveRight();
    }
    repaint();
    if (isWin()) {
        JOptionPane.showMessageDialog(this, "恭喜您通过此关!!!");
    }
}

private void moveLeft() {
    column--;
}

private void moveDown() {
    row++;
}

private void moveRight() {
    column++;
}

private void moveUp() {
    row--;
}
```

`isWin()` 方法判断当前是否已经胜利，只需检查当前界面是否仍存在没有变绿的箱子（在目的地的箱子）即可。代码如下：

```
public boolean isWin(){
    for(int i=0; i<map2.length; i++){
        for(int j=0; j<map2[i].length; j++){
            if(map2[i][j] == BOX){//存在不是绿色的箱子
                return false;
            }
        }
    }
    return true;
}
```

20.3.2 设计游戏窗口类（BoxFrame2.java）

在项目中创建一个继承 `JFrame` 的 `BoxFrame2` 类，用于显示游戏面板 `PushBox`。代码如下：

```
import java.awt.Container;
import javax.swing.JFrame;
```



```

public class BoxFrame2 extends JFrame {
    public BoxFrame2() {
        //默认的窗体名称
        setTitle("2.5D 推箱子游戏");
        //获得自定义面板“PushBox 面板”的实例
        PushBox panel = new PushBox();
        Container contentPane = getContentPane();
        contentPane.add(panel);
        setSize(800, 800);
    }
    public static void main(String[] args) {
        BoxFrame2 e1 = new BoxFrame2();
        //设定允许窗体关闭的操作
        e1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //显示窗体
        e1.setVisible(true);
    }
}

```

由于人物角色移动时不考虑是否符合游戏规则，因此会出现图 20-7 所示的效果。

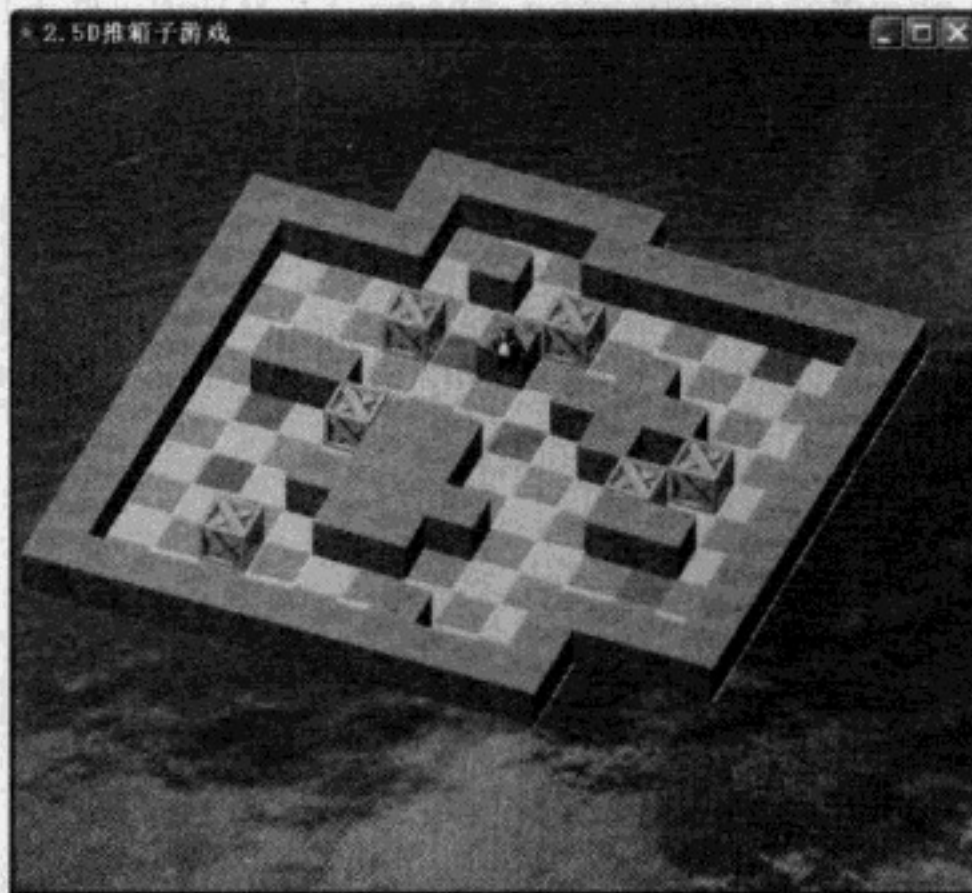


图 20-7 人在墙上的效果

基于上述原因，对人物的移动仍需要判断是否符合游戏规则。游戏规则的实现基本与 2D 推箱子游戏一致，不过需要注意由于采用两层技术，因此地图信息放在两个数组 map1 和 map2 中。而 2D 推箱子游戏则放在一个数组 map 中。具体代码如下：

```

private void moveLeft() {
    // TODO Auto-generated method stub
    //左一位 p1 为 WALL
    if (map2[row][column - 1] == WALL)
        return;
    //左一位 p1 为 BOX
    if (map2[row][column - 1] == BOX || map2[row][column - 1] == BOXONEND) {
        if (map2[row][column - 2] == WALL)
            return;
        if (map2[row][column - 2] == BOX)
            return;
        if (map2[row][column - 2] == BOXONEND)

```



```

        return;
    //左两位 p2 为 END, GRASS 则向上一步
    if (map1[row][column - 2] == END
        || map1[row][column - 2] == WhiteGRASS
        || map1[row][column - 2] == BlackGRASS) {
        //左左一位 p2 为 END
        if (map1[row][column - 2] == END) //上上一位 p2 为 END
            map2[row][column - 2] = BOXONEND;
        if (map1[row][column - 2] == WhiteGRASS //上上一位 p2 为 GRASS
            || map1[row][column - 2] == BlackGRASS)
            map2[row][column - 2] = BOX;
        map2[row][column - 1] = -1; //原来的箱子被移掉
        //人离开后修改人的坐标
        man = Toolkit.getDefaultToolkit().getImage("images\\b1.png"); //向左移动人物
        column--;
    }
} else {
    //左一位为 GRASS 或 END, 其他情况不用处理
    if (map1[row][column - 1] == WhiteGRASS
        || map1[row][column - 1] == BlackGRASS
        || map1[row][column - 1] == END) {
        //人离开后修改人的坐标
        man = Toolkit.getDefaultToolkit().getImage("images\\b1.png"); //人物离开
        column--;
    }
}
}

private void moveUp() {
    // TODO Auto-generated method stub
    //上一位 p1 为 WALL
    if (map2[row - 1][column] == WALL)
        return;
    //上一位 p1 为 BOX, 需考虑 P2
    if (map2[row - 1][column] == BOX || map2[row - 1][column] == BOXONEND) {
        if (map2[row - 2][column] == WALL)
            return;
        if (map2[row - 2][column] == BOX)
            return;
        if (map2[row - 2][column] == BOXONEND)
            return;
    }
    //上两位 p2 为 END, GRASS 则向上一步
    if (map1[row - 2][column] == END
        || map1[row - 2][column] == WhiteGRASS
        || map1[row - 2][column] == BlackGRASS) {
        //上两位 p2 为 END
        if (map1[row - 2][column] == END) //上上一位 p2 为 END
            map2[row - 2][column] = BOXONEND;
        if (map1[row - 2][column] == WhiteGRASS //上上一位 p2 为 GRASS
            || map1[row - 2][column] == BlackGRASS)
            map2[row - 2][column] = BOX;
        map2[row - 1][column] = -1; //原来的箱子被移掉
    }
}

```



```

        //人离开后修改人的坐标
        man=Toolkit.getDefaultToolkit().getImage("images\\c1.png");//人物向上
        row--;
    }
} else {
    //上一位为 GRASS 或 END, 无须考虑 P2
    if (map1[row - 1][column] == WhiteGRASS
        || map1[row - 1][column] == BlackGRASS
        || map1[row - 1][column] == END) {
        //人离开后修改人的坐标
        man = Toolkit.getDefaultToolkit().getImage("images\\c1.png");//人物离开
        row--;
    }
}
}
}

```

其他两种方向的代码与此相似, 这里不再列出。从代码中可见, 2.5D 推箱游戏与 2D 推箱子游戏的规则判断基本一致。不过采用两层技术的好处在于第一层(地板层)一直保持不变, 因此在判断目的地时不用考虑人物角色、箱子是否在目的地上从而影响判断。加上此逻辑判断后, 人物的移动就不可以随意移动了。游戏最终效果如图 20-8 所示。

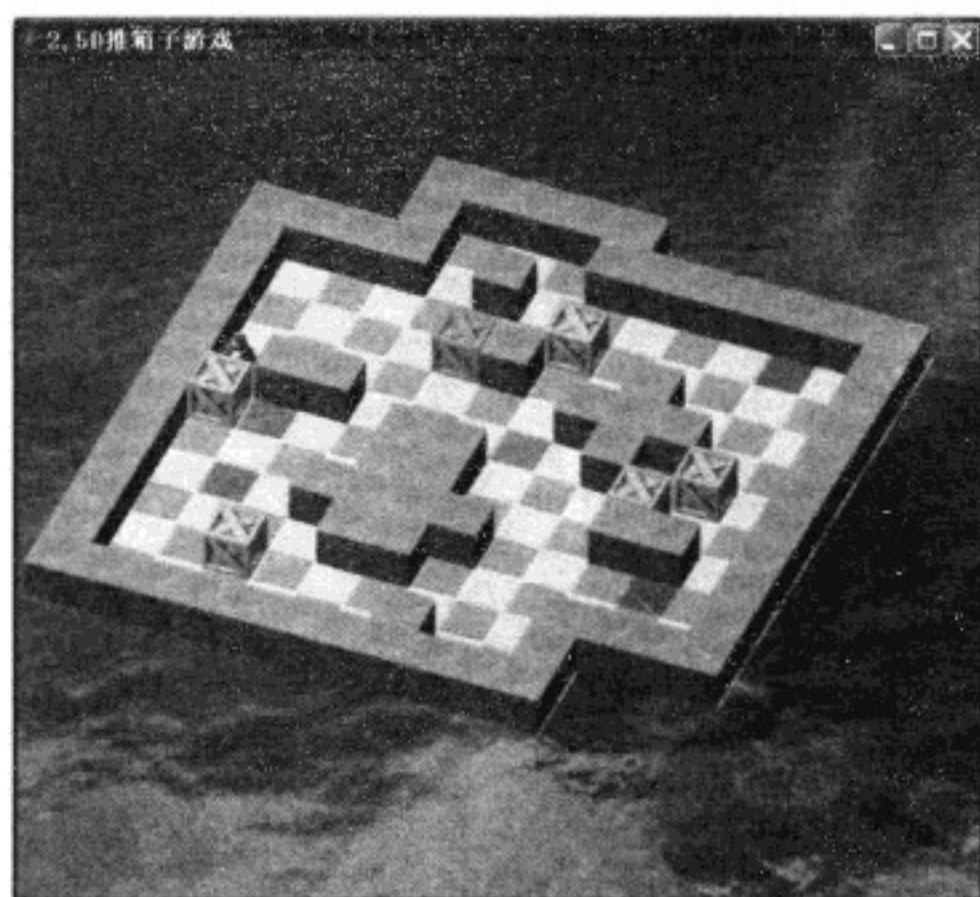


图 20-8 游戏最终效果