第1章　贪食蛇游戏开发

一、单元概述

桌面应用程序是一种重要的应用程序形式，日常使用的应用软件多数属于桌面应用软件。因此，掌握桌面应用软件开发方法对于软件工程师来说至关重要。本章介绍如何利用 Java 图形化技术和多线程技术开发一个简单的贪食蛇游戏，以此来阐述桌面应用软件开发的基本过程、设计思路和方法。

二、知识要点及掌握程度

(1)Java 图形化技术：运用。

(2)Java 多线程技术：理解。

1.1 项目概述

利用 Java 图形化技术和多线程技术开发一个贪食蛇游戏。

游戏中玩家可以通过键盘控制游戏区中贪食蛇的运动，当蛇头碰到软件产生的食物时加长蛇身。当蛇碰到墙壁或者自己身体时，游戏结束。游戏设定 3 个级别，级别越高蛇运动的速度越快、难度越大，玩家可以根据自己的需要自由设定初始游戏级别，当玩家取得的分数到达一定水平时游戏自动进入下一个级别，达到最高级别时不再增加。游戏以最终玩家获得的分数来判断玩家水平高低。

1.2 需求分析

1.2.1 开发目标

本游戏软件的总体目标是按照特定游戏规则为玩家提供一个方便友好的游戏界面，同时满足不同级别的玩家的不同难度需求。游戏规则如下：

- 玩家通过键盘控制蛇在游戏区中运动。
- 当蛇头碰到软件提供的食物或者宝物时，得分并增加蛇身长度一个单位。
- 当蛇头碰到蛇身或墙壁时，如果没有相应的宝物则游戏结束。
- 游戏设定两种宝物，一种为穿身宝物，另一种为穿墙宝物。
- 记分规则：吃掉一个食物或者宝物加 100 分。
- 晋级规则：累计得 2000 分自动进入下一个级别。

1.2.2 运行环境

1.硬件环境

- 处理器：Inter P4 或更高。
- 内存：128MB 以上。
- 硬盘空间：1GB 以上。

2.软件环境

- 操作系统：Windows 2000/XP/Vista。
- Java 运行环境：JRE 1.5 以上。

1.2.3 功能需求描述

本游戏软件需要实现以下一些基本功能：

(1)游戏区：玩家可以在游戏区内控制蛇吃食物。

(2)状态提示区：使玩家能够在游戏过程中随时了解得分情况和获得宝物情况。

(3)游戏控制:玩家可以通过游戏控制功能来选择开始新一局游戏、暂停或者退出游戏。

(4)级别设置:玩家可以根据自己的需要自行设定游戏的开始级别。

(5)帮助:提示玩家游戏规则和游戏控制方法。

1.3　项目设计

1.3.1　模块结构设计

依据需求分析结果,贪食蛇游戏程序可以设计为单机软件。

整个系统可以分为四个模块:游戏区模块、游戏控制模块、级别设置模块和帮助模块。系统模块结构如图 1-1 所示。

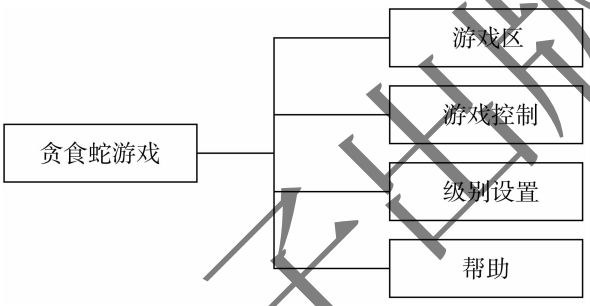

图 1-1　系统模块结构图

1.游戏区模块设计

该模块为玩家提供主体游戏功能,能够处理玩家的各种游戏操作,显示得分情况和获得宝物数目,并最终显示游戏结果。该模块应包括三个子功能模块:创建游戏区、处理玩家游戏操作和游戏状态提示。模块功能树如图 1-2 所示。

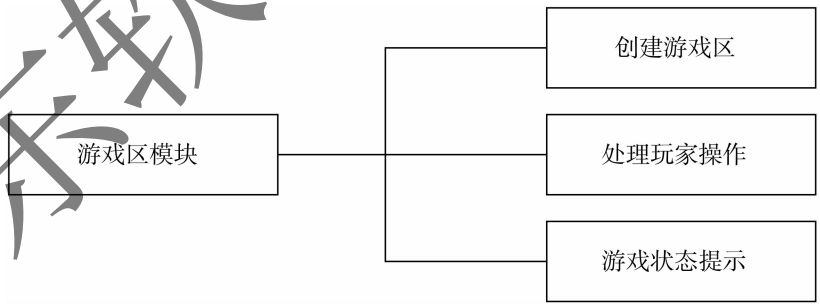

图 1-2　游戏区模块功能树图

2.游戏控制模块设计

该模块为玩家提供游戏控制功能,应包括开始游戏、暂停游戏和退出游戏三部分功能。游戏控制模块功能树如图 1-3 所示。

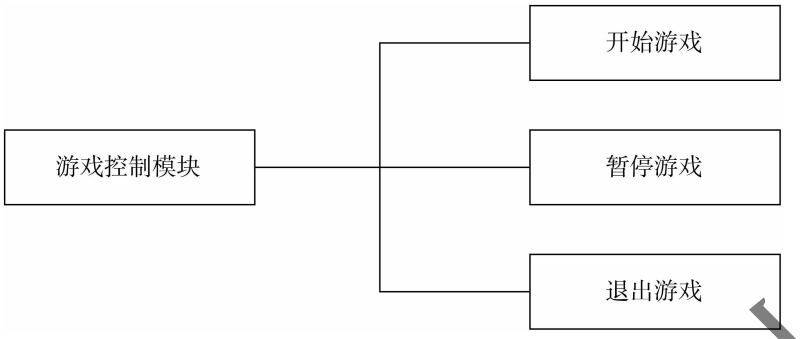

图 1-3 游戏控制模块功能树图

3.级别设置模块设计

该模块为玩家提供自行设定游戏的初始级别功能，主要提供初级、中级和高级三个游戏初始级别的设定，级别越高游戏速度越快，难度越大。级别设置模块功能树如图 1-4 所示。

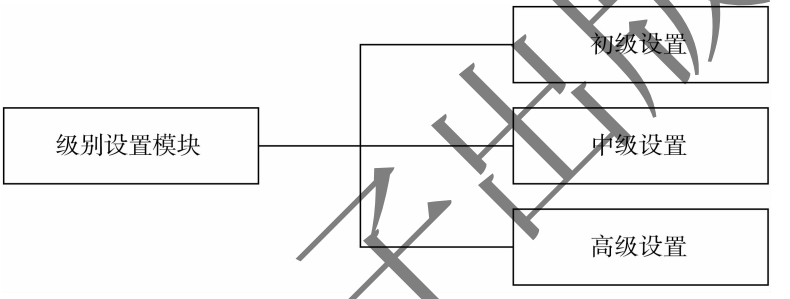

图 1-4 级别设置模块功能树图

1.3.2 系统处理流程

系统的主体处理流程如图 1-5 所示。

1.3.3 界面设计

贪食蛇游戏作为大众娱乐软件，其用户界面往往是决定软件质量的首要条件，所以界面设计是开发与设计的重点。

界面设计应遵循简洁美观、方便易用的基本原则。

1.游戏主体界面设计

游戏主体界面使用简单和直观的布局设计，游戏区放置在界面的主体位置，在游戏区上部放置状态显示区，各项游戏控制功能通过工具栏提供。具体设计效果图如图 1-6 所示。

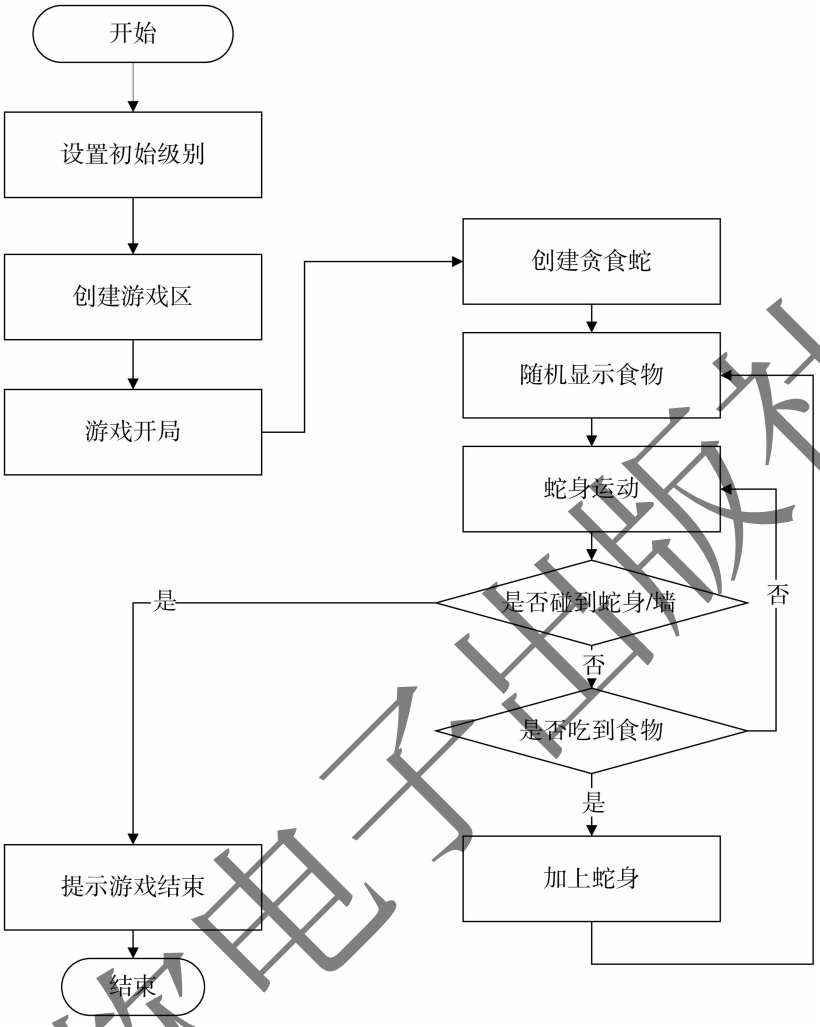

图 1-5　系统流程图

图 1-6　游戏主体界面设计效果图

2.帮助界面设计

贪食蛇游戏规则和控制方法比较简单，所以帮助界面通过完全文本的方式提供。具体设计效果图如图 1-7 所示。

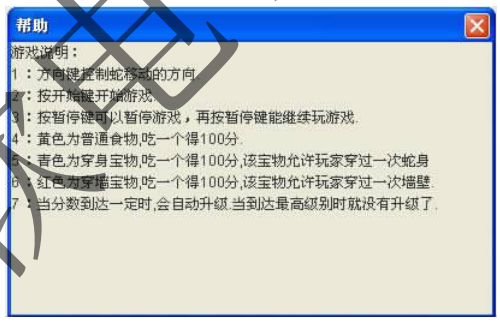

图 1-7　帮助界面设计效果图

1.3.4　类设计

1.主窗体类 MainFrame

该类完成游戏主体界面构造，根据界面设计结果，主体界面由“难度”菜单、游戏控制工具栏和游戏区构成。类的详细设计如下。

(1)属性设计：如表 1-1 所示。

表 1-1　MainFrame 类属性定义表

属性名	属性类型	描述
menuBar	JMenuBar	系统菜单栏
mLevel	JMenu	难度菜单
miBegin	JRadioButtonMenuItem	初级难度菜单项
miMiddle	JRadioButtonMenuItem	中级难度菜单项
miHard	JRadioButtonMenuItem	高级难度菜单项
toolBar	JToolBar	系统工具栏
jbtStart	JButton	系统工具栏上的游戏开始按钮
jbtPause	JButton	系统工具栏上的游戏暂停/继续按钮
jbtStop	JButton	系统工具栏上的游戏退出按钮
jbtHelp	JButton	系统工具栏上的帮助按钮
playPane	PlayPanel	游戏区面板
BEGINNER	int	游戏级别常量,标志初级,值为 1
MIDDLE	int	游戏级别常量,标志初级,值为 2
EXPERT	int	游戏级别常量,标志初级,值为 3
isEnd	boolean	游戏结束标志(true:结束;false:运行)
isPause	boolean	游戏暂停标志(true:暂停;false:运行)
level	int	当前游戏级别(1～10)

(2)方法设计:如表 1-2 所示。

表 1-2　MainFrame 类方法定义表

方法名	返回值类型	参数	方法说明
MainFrame	无	无	MainFrame 类的构造方法,完成主界面的初始创建和相关事件监听的注册

2.游戏区类 PlayPanel

该类完成游戏区构建,以及处理玩家游戏操作和游戏状态提示等功能。类的详细设计如下。

(1)属性设计:如表 1-3 所示。

表 1-3 PlayPanel 类属性定义表

属性名	属性类型	描述
length	int	当前蛇身长度
rows	int [ ]	蛇身各个组成方块的行号
columes	int [ ]	蛇身各个组成方块的列号
direction	int	蛇身当前运动方向
lastdirection	int	蛇身在改变运动方向前的运动方向
speed	int	蛇身运动速度
UP	int	蛇身运动方向常量,标志向上运动,值为 1
LEFT	int	蛇身运动方向常量,标志向左运动,值为 2
DOWN	int	蛇身运动方向常量,标志向下运动,值为 3
RIGHT	int	蛇身运动方向常量,标志向右运动,值为 4
ROWS	int	游戏区大小常量,标志行数,值为 30
COLS	int	游戏区大小常量,标志列数,值为 50
playBlocks	JButton [ ][ ]	所有游戏区方块
lost	boolean	游戏状态标志(true:结束;false:运行)
throughbody	int	当前获得穿身宝物个数
throughwall	int	当前获得穿墙宝物个数
score	int	当前得分
jlScore	JLabel	显示当前得分情况
jlThroughBody	JLabel	显示当前获得穿身宝物个数
jlThroughWall	JLabel	显示当前获得穿墙宝物个数

(2)方法设计:如表 1-4 所示。

表 1-4 PlayPanel 类方法定义表

方法名	返回值类型	参数	方法说明
createSnake()	void	无	在游戏区上创建和显示蛇
createFood()	void	无	在游戏区上创建和显示食物
moveSnake()	void	无	移动蛇身运动,并判断当前游戏状态,包括吃食物、碰蛇身和碰墙
isLost()	boolean	无	获取游戏进行状态,返回值 true 表明游戏结束,返回值 false
setSnakeDirection()	void	int direction	通过参数修改蛇运动方向

3.主框架动作事件监听器类 MainFrameActionListener

该类完成主框架动作事件的处理,具体完成难度设置菜单和工具栏动作事件的响应和处理。类的详细设计如下。

(1)属性设计:无。

(2)方法设计:如表 1-5 所示。

表 1-5　　MainFrameActionListener 类方法定义表

方法名	返回值类型	参数	方法说明
actionPerformed ()	void	ActionEvent e	动作事件的处理方法,通过判断事件源的不同区分不同用户操作,完成响应用户难度菜单和工具栏操作的请求

4.主框架键盘事件监听器类 MainFrameKeyListener

该类完成主框架键盘事件的处理,实现蛇身运动的键盘控制功能。类的详细设计如下。

(1)属性设计:无。

(2)方法设计:如表 1-6 所示。

表 1-6　　MainFrameKeyListener 类方法定义表

方法名	返回值类型	参数	方法说明
keyPressed ()	void	KeyEvente	键盘事件处理方法,通过判断用户按键的不同来设置游戏区蛇身运动的方向,实现键盘控制蛇身运动功能

5.蛇身运动线程类 SnakeThread

该类实现以线程方式移动蛇身的运动功能。类的详细设计如下。

(1)属性设计:无。

(2)方法设计:如表 1-7 所示。

表 1-7　　SnakeThread 类方法定义表

方法名	返回值类型	参数	方法说明
run ()	void	无	线程任务方法,该方法每隔一定时间移动一下蛇身,用户通过菜单修改移动蛇身的间隔来实现难度设置功能

6.帮助对话框 HelpDialog

该类实现帮助对话框界面。类的详细设计如下。

(1)属性设计:如表 1-8 所示。

表 1-8　　HelpDialog 类属性定义表

属性名	属性类型	描述
content	JTextArea	显示帮助内容

(2)方法设计:如表 1-9 所示。

表 1-9 HelpDialog 类方法定义表

方法名	返回值类型	参数	方法说明
HelpDialog()	无	无	帮助对话框的构造方法，完成帮助对话框界面的初始构造

1.4 项目实现

1.4.1 技术准备

1.4.1.1 Eclipse

1.Eclipse 简介

Eclipse 是一种可扩展的开放源代码 IDE。2001 年 11 月，IBM 公司捐出价值 4,000 万美元的源代码组建了 Eclipse 联盟，并由该联盟负责这种工具的后续开发。

Eclipse 平台的用户界面是由透视图、视图和编辑器组成的。每个“工作台”窗口都包含一个或多个透视图；透视图则包含视图和编辑器（透视图是根据用户的不同选择来布局视图和编辑器的）；可同时打开任意数目的编辑器，但在任一时刻只能有一个编辑器是活动的。星号（*）指示编辑器具有未保存的更改。

Eclipse 的主体界面如图 1-8 所示。

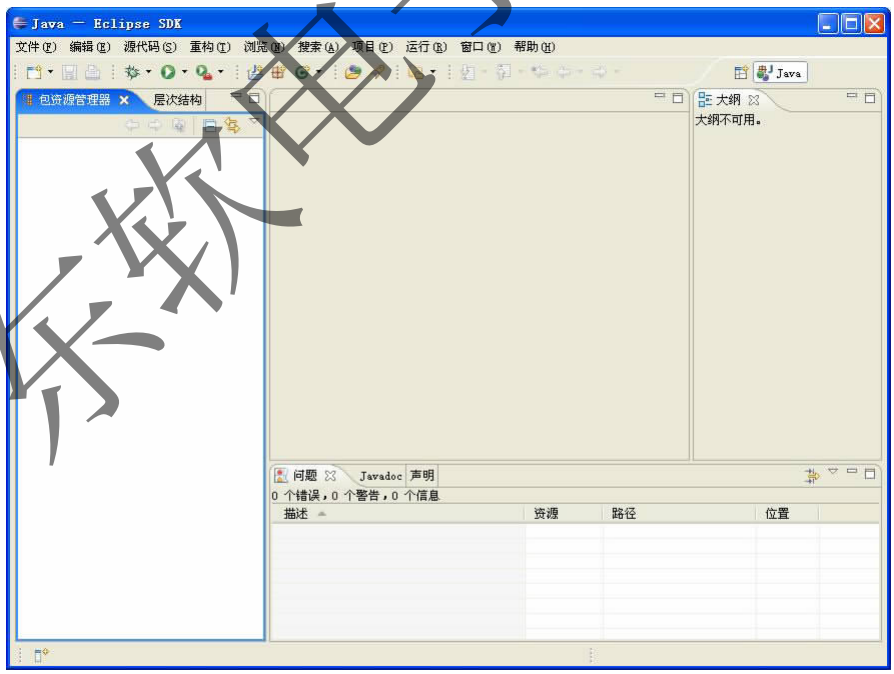

图 1-8 Eclipse 主体界面截图

2.项目构建过程

(1)在菜单中选择“文件”→“新建”→“项目”。

(2)项目列表选“Java 项目”,具体如图 1-9 所示。

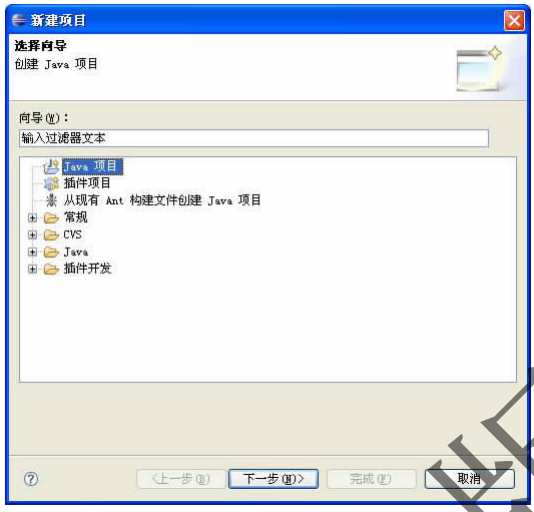

图 1-9　Eclipse 新建项目界面截图

(3)点击“下一步”按钮,输入项目名称,例如:“HelloProject”,点击“完成”按钮,具体如图 1-10 所示。

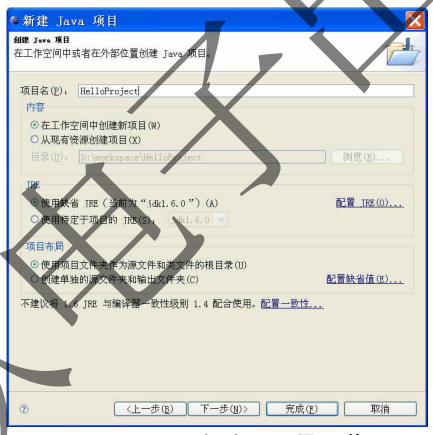

图 1-10　Eclipse 新建项目界面截图 2

(4)在菜单中选择“文件”→“新建”→“类”。

(5)在名称域输入“HelloWorld”,并点击 public static void main(String[] args) 的复选框,让 Eclipse 创建 main 方法。

(6)点击“完成”按钮,具体如图 1-11 所示。

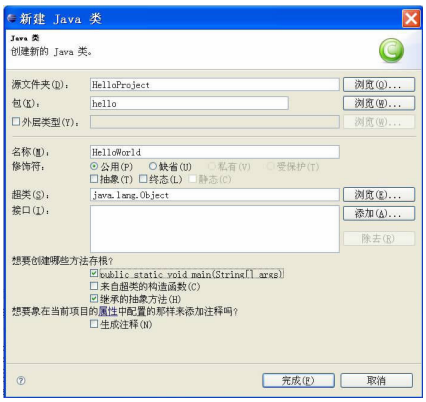

图 1-11 Eclipse 新建类界面截图

(7)打开 HelloWorld.java 文件,在 main 方法中输入:System.out.println("Hello World")。

(8)使用【Ctrl+S】保存,这时将自动编译 HelloWorld.java。

(9)在菜单中选择:"运行"→"运行方式"→"Java 应用程序",如图 1-12 所示。

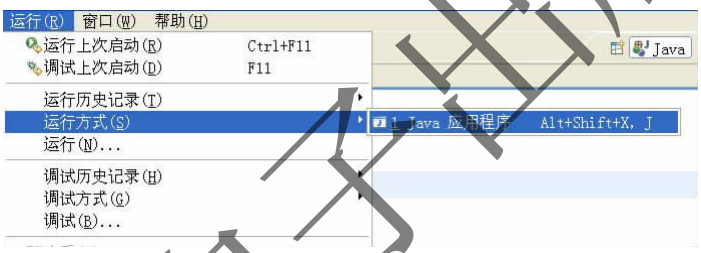

图 1-12 Eclipse 运行 Java 应用程序界面截图

(10)运行成功后,控制台窗口会显示一行"Hello World",如图 1-13 所示。

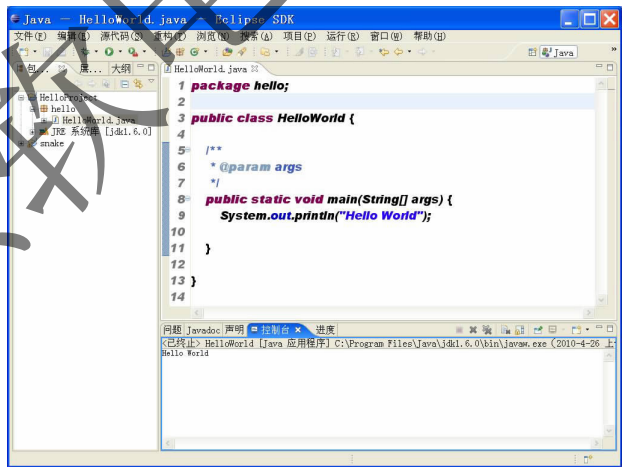

图 1-13 运行 java 程序结果截图

1.4.1.2 Java GUI 技术

在 Java 中,GUI 由 GUI 组件构成,目前常用的 GUI 组件类多数定义在 javax.swing 包中,称为 Swing 组件。这和早期的 Java 有些不同,早期的 GUI 组件定义在称为抽象窗口工具包(Abstract Window Toolkit,AWT)的 java.awt 包中。由于 AWT 组件直接绑定在本地图形用

户界面功能上，对底层平台的依赖性较强，因此可移植性较差。到了 Java 2 时，出现了 Swing 组件，Swing 组件直接使用 Java 语言编写，对本地底层平台的依赖性较少，更灵活也更稳定。Swing 组件称为轻型组件，而 AWT 组件称为重型组件。因此，在编程中建议使用 Swing 组件，以提高程序的可移植性。

为了方便实现各种图形用户界面程序，Java 提供了大量的图形用户界面类和接口，比如组件类有：JFrame、JPanel、JDialog、JButton、JLabel、JTextField、JMenu 和 JMenuBar 等；事件类有：ActionEvent、MouseEvent 和 KeyEvent 等；监听接口有：ActionListener、MouseListener 和 MouseMotionListener 等；布局管理器类有：BorderLayout、FlowLayout 和 GridLayout 等；绘图相关的类有：Griphics、Color、Font 等。

从列举的这些类和接口中可以感觉到，图形化程序设计比之前的命令行方式要复杂。特别是对于刚刚接触 Java 的人来讲，同时接触这么多的类和接口可能感觉无从下手。为了便于理解，可以简单地将图形用户界面程序设计的主要工作分成两部分。

(1)设计并创建界面外观：主要是创建组成图形界面的各个组件，并按照设计组合排列，构成完整的图形用户界面。

(2)实现界面的交互功能：主要是为界面外观添加事件处理，实现能够处理各种界面事件的处理方法，以完成程序与用户之间进行交互的任务。

1.框架类 JFrame

在 Java 中，框架是最常用的容器之一，它是编写图形化应用程序所使用的最外层容器。也就是说，编写一个图形化应用程序，先要创建一个框架，然后通过这个框架来组织其他的 GUI 组件。

(1) JFrame 的构造方法。

public JFrame()：创建不带标题的框架。

public JFrame(String title)：创建指定标题的框架。标题通过参数 title 指定。

(2) JFrame 的常用方法。

public setSize(int width, int height)：设置框架的大小。width 为框架的宽度，height 为框架的高度，它们以像素为单位。

public setVisible(boolean b)：设置框架的可视状态。参数为 true 框架显示，参数为 false 框架隐藏。

public setDefaultOperation(into peration)：设置框架缺省的关闭操作。operation 的值如果设置为 JFrame.EXIT_ON_CLOSE，则在关闭框架时退出应用程序。

2.在框架上添加组件

(1)获取框架内容窗格的方法。

public Container getContentPane()：其中 Container 类是所有容器类的父类。

(2)所有容器常用的方法。

add(Component comp)：该方法可以将组件添加到容器中，其中 Component 类是所有组件类的父类。

setLayout(LayoutManager mgr)：该方法可以为容器设置一种布局管理器，其中 LayoutManager 是所有布局管理器类必须实现的接口，相当于所有布局管理器类的父类。

(3)构造图形用户界面的基本思路。

根据界面设计确定需要的容器,然后通过 setLayout 方法为容器设置合适的布局管理器,最后通过 add 方法将组件添加到容器中。这里需要注意的是容器中的组件包括控制组件,也包括一部分容器。

具体过程可参照以下例程:

```
//TestFrame.java
import java.awt.*;
import javax.swing.*;
public class TestFrame
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("图形界面程序");
        //将框架的内容窗格的布局管理器设置为 FlowLayout 布局
        frame.getContentPane().setLayout(new FlowLayout());
        frame.getContentPane().add(new JButton("确定"));
        frame.getContentPane().add(new JButton("取消"));
        frame.setSize(200,100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

程序运行后显示如图 1-14 所示的图形界面。

图 1-14 框架运行效果图

3.事件处理

在 Java 中使用事件处理的方式来实现图形用户界面的交互,采用的事件处理机制称为委托事件处理模型。该模型规定的事件处理流程是这样的:当用户操作图形界面组件时,该组件会自动产生某种代表这种操作发生的信号,这个信号称为事件(event),产生事件的组件称为事件源。然后一个称为监听器的对象会接收到这个事件,并对其进行处理。这样,当用户操作某个 GUI 组件时,只要有监听器监听这个组件产生的事件,那么程序就可以实现对用户行为的响应,也就是所谓的交互了。

当用户行为作用于 GUI 组件时,这些组件会产生某种事件(event)。产生事件的 GUI 组件称为事件源对象,所有的 GUI 组件都可以是事件源。事件是事件类的实例,它由事件源在用户行为的作用下自动产生,Java 中所有的事件类都是 Java.lang.EventObject 的子类,EventObject 类的子类描述特定类型的事件,比如:ActionEvent 描述的是动作事件,WindowEvent 描述的是窗口事件,MouseEvent 描述的是鼠标事件,等等。AWT 事件类,定义在 java.awt.event 包中,AWT 组件和 Swing 组件都可以产生这些类型的事件。表 1-10 列举了

一些基本的用户行为、事件源和事件类型之间的关系。

表 1-10　　用户行为、事件源和事件类之间的关系

用户行为	事件源	事件类型
点击按钮	JButton(按钮)对象	ActionEvent
在文本域按下回车	JTextField(文本域)对象	ActionEvent
改变文本	JTextField(文本域)对象	TextEvent
改变文本	JTextArea(文本区)对象	TextEvent
窗口打开、关闭、最小化、还原或正在关闭	Window 及其子类对象	WindowEvent
在容器中添加或者删除组件	Container 的所有子类对象	ContainerEvent
组件移动、改变大小、隐藏或显示	Component 的所有子类对象	ComponentEvent
组件获取或者失去焦点	Component 的所有子类对象	FocuseEvent
按下或者释放键	Component 的所有子类对象	KeyEvent
鼠标按下、释放、点击、进入或离开组件、移动或者拖动	Component 的所有子类对象	MouseEvent

Java 使用委托事件处理模型来进行事件处理，就是事件源产生的事件委托给监听器进行处理。具体来说就是先要创建一个能够处理这种事件的监听器(监听器中实现了事件处理方法)，然后将这个监听器注册给产生这种事件的事件源(事件源中包含对应的注册方法)。这样，当用户行为作用于事件源(GUI 组件)时，事件源就会产生特定的事件，那么事先注册给它的特定监听器就能够接收到这个特定的事件，然后监听器调用其中实现的事件处理方法进行处理。

Java 中每一种事件类都有其对应的监听接口，要创建一个能够监听特定事件的监听器，就必须先定义一个实现监听接口的类，那么这个类的对象就是一个能够监听对应事件的监听器了。比如：ActionEvent 对应的监听接口是 ActionListener，要创建一个能够处理 ActionEvent 事件的监听器就要实现 ActionListener 接口。在每一个监听接口中都定义了若干个事件处理方法，实现监听器实际上就是实现这些事件处理方法。当事件发生并且监听器接收到这个事件时，监听器就会调用事件处理方法进行处理。比如：ActionListener 接口中定义了 actionPerformed 方法，当 ActionListener 接收到 ActionEvent 发生时，就会调用这个方法，写在这个方法体中的事件处理语句就会被执行。

(1)动作事件类。

类名：ActionEvent；

常用方法：

public Object getSource()：得到事件源引用，通常用于区分事件源。

public String getActionCommand()：得到动作命令，每个能够产生动作事件的事件源在产生动作事件时都会为动作事件赋予一个字符串类型的动作命令。对按钮来说，缺省就是按钮上显示的标签，比如：本例中的 Ok 按钮，它的动作命令就是“Ok”，不过这个动作命令可以通过事

件源的 setActionCommand 方法进行修改。通常这个方法用于区分动作事件的事件源。

(2)动作事件监听接口。

监听接口名:ActionListener;

事件处理方法:

```
public void actionPerformed(ActionEvent e);
```

(3)事件源的注册方法。

public void addActionListener(ActionListener listener):将动作事件监听器注册给事件源。

动作事件的处理过程可以参照以下例程:

```
//TestActionEvent.java
import javax.swing.*;
import java.awt.*;
//引入事件处理需要的事件类和监听接口
import java.awt.event.*;
//定义一个框架类,同时实现了动作事件的监听接口
public class TestActionEvent extends JFrame implements ActionListener
{
    // 创建两个按钮
    private JButton jbtOk = new JButton("确定");
    private JButton jbtCancel = new JButton("取消");
    // 构造方法
    public TestActionEvent(String title)
    {
      // 初始化框架标题
      super(title);
      // 设置内容窗格的布局为 FlowLayout
      getContentPane().setLayout(new FlowLayout());
      // 在内容窗格上添加两个按钮
      getContentPane().add(jbtOk);
      getContentPane().add(jbtCancel);
      // 为两个按钮注册监听器
      jbtOk.addActionListener(this);
      jbtCancel.addActionListener(this);
  }
  // main 方法
  public static void main(String[] args)
  {
    TestActionEvent frame = new TestActionEvent("动作事件");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(100, 80);
    frame.setVisible(true);
```

```
    }
    // 事件处理方法,在事件发生时被调用
    public void actionPerformed(ActionEvent e)
    {
      //判断监听到的事件是否是按钮 jbtOk 产生的
      if (e.getSource() == jbtOk)
      {
        System.out.println("确定按钮被点击");
      }
      else if (e.getSource() == jbtCancel)
      {
        System.out.println("取消按钮被点击");
      }
    }
}
```

程序运行效果如图 1-15 所示。

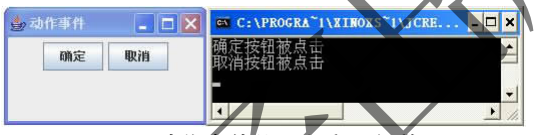

图 1-15　动作事件处理程序运行效果图

4.随堂练习

(1)使用 JFrame、JLabel、JTextField、JButton 和 JPanel 等组件实现如图 1-16 所示的登录界面。

图 1-16　登录界面运行效果

参考实现代码如下：

```
import java.awt.*;
import javax.swing.*;
public class Login extends JFrame
{
    public Login(String title)
    {
        super(title);
        Container  cp = this.getContentPane();
        JPanel p1 = new JPanel();
        p1.add(new JLabel("用户名"));
        p1.add(new JTextField(10));
        JPanel p2 = new JPanel();
```

```
        p2.add(new JLabel("口    令"));
        p2.add(new JTextField(10));

        JPanel p3 = new JPanel();
        p3.add(new JButton("登录"));
        p3.add(new JButton("退出"));


        cp.add(p1,BorderLayout.NORTH);
        cp.add(p2,BorderLayout.CENTER);
        cp.add(p3,BorderLayout.SOUTH);
    }
    public static void main(String[] args)
    {
        Login frame = new Login("Login");
        frame.setSize(200,150);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

(2)实现如图 1-17 所示程序,当按下键盘某个字母健时在窗体上显示该字母,并且按方向键时能够在窗体上移动该字母位置。

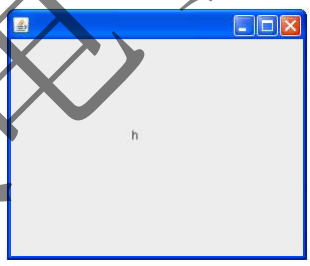

图 1-17　键盘事件程序运行效果

参考实现代码如下:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class KeyboardEventDemo extends JFrame implements KeyListener
{
    KeyMessagePanel mp=new KeyMessagePanel();
    KeyboardEventDemo()
    {
        mp.setFocusable(true);
        this.getContentPane().add(mp);
        mp.addKeyListener(this);
```

```
    }
    public void keyPressed(KeyEvent e){
        switch(e.getKeyCode()){
            case KeyEvent.VK_DOWN:mp.y+=10;break;
            case KeyEvent.VK_UP:mp.y-=10;break;
            case KeyEvent.VK_LEFT:mp.x-=10;break;
            case KeyEvent.VK_RIGHT:mp.x+=10;break;
            default:mp.s=e.getKeyChar();
        }
        mp.repaint();
    }
    public void keyReleased(KeyEvent e){

    }
    public void keyTyped(KeyEvent e){

    }
    public static void main(String[] args){
        KeyboardEventDemo f=new KeyboardEventDemo();
        f.setSize(300,300);
        f.setDefaultCloseOperation(3);
        f.setVisible(true);
    }
}
class KeyMessagePanel extends JPanel{
    int x=100;
    int y=100;
    char s='a';

    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        g.drawString(String.valueOf(s),x,y);
    }
}
```

(3)编写如图 1-18 所示的菜单界面程序。

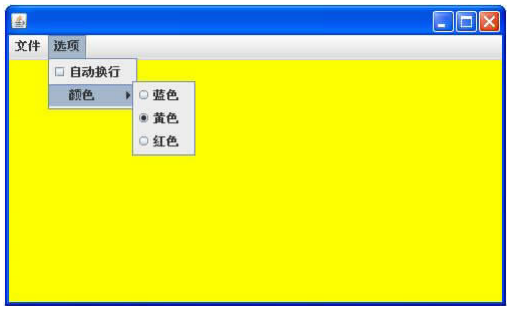

图 1-18　菜单界面程序运行效果

参考实现代码如下：

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

public class TestMenu extends JFrame implements ActionListener,ItemListener
{
    private JMenuItem jmiNew,jmiOpen,jmiExit;
    private JRadioButtonMenuItem jrbmiBlue,jrbmiYellow,jrbmiRed;

    public TestMenu(){
        JMenuBar jmb = new JMenuBar();
        setJMenuBar(jmb);
        JMenu fileMenu = new JMenu("文件", true);
        JMenu OptionMenu = new JMenu("选项", true);
        jmb.add(fileMenu);
        jmb.add(OptionMenu);

        fileMenu.add(jmiNew = new JMenuItem("新建"));
        fileMenu.add(jmiOpen = new JMenuItem("打开"));
        fileMenu.addSeparator();//在菜单中添加一条分隔线
        fileMenu.add(jmiExit = new JMenuItem("退出"));
        OptionMenu.add(new JCheckBoxMenuItem("自动换行"));
        JMenu colorSubMenu=new JMenu("颜色");
        OptionMenu.add(colorSubMenu);
        colorSubMenu.add(jrbmiBlue=new JRadioButtonMenuItem("蓝色"));
        colorSubMenu.add(jrbmiYellow=new JRadioButtonMenuItem("黄色"));
        colorSubMenu.add(jrbmiRed=new JRadioButtonMenuItem("红色"));
        ButtonGroup btg=new ButtonGroup();
        btg.add(jrbmiBlue);
        btg.add(jrbmiYellow);
        btg.add(jrbmiRed);

        jmiNew.addActionListener(this);
        jmiOpen.addActionListener(this);
        jmiExit.addActionListener(this);

        jrbmiBlue.addItemListener(this);
        jrbmiYellow.addItemListener(this);
        jrbmiRed.addItemListener(this);

}
```

```
public void actionPerformed(ActionEvent e){
    if(e.getSource()==jmiNew)
        JOptionPane.showMessageDialog(this,"新建文件");
    else if(e.getSource()==jmiOpen)
        JOptionPane.showMessageDialog(this,"打开文件");
    else if(e.getSource()==jmiExit)
        System.exit(0);
}
public void itemStateChanged(ItemEvent e){
    if(e.getSource() instanceof JRadioButtonMenuItem){
        if(jrbmiBlue.isSelected())
            getContentPane().setBackground(Color.BLUE);
        if(jrbmiYellow.isSelected())
            getContentPane().setBackground(Color.YELLOW);
        if(jrbmiRed.isSelected())
            getContentPane().setBackground(Color.RED);
    }
}
public static void main(String[] args)
{
        TestMenu frame = new TestMenu();
        frame.setSize(500,300);
        frame.setDefaultCloseOperation(3);
        frame.setVisible(true);
    }
}
```

1.4.1.3　Java 多线程技术

多线程是指在整个程序的一次运行过程中，多个线程在并发地执行。在单处理器的系统中，这些并发执行的线程可以分享 CPU 的时间，操作系统负责对它们进行调度和资源分配，从宏观上看，这些线程好像在并行执行一样，但是实际上，在任意时刻，只能有一个线程在使用 CPU。只有在多处理器的系统中，多个线程才能达到真正意义上的并行执行。

多线程有许多用途，它在图形用户界面程序设计和网络程序设计中非常常用，多线程程序不仅可以有效地利用 CPU，还可以有效地优化程序的吞吐量。在单处理器的系统中，多个线程的调度会降低一些效率；但是从程序设计、资源平衡和用户使用方便等方面来看，所牺牲的效率是完全值得的。

1.利用 Thread 类编写线程

在 Java 中编写一个线程非常容易，最简单的做法就是继承类 java.lang.Thread，这个类已经具有了创建和运行一个线程所必要的基本内容。Thread 类中最重要的方法就是 run()，线程用于完成一个实际功能的代码都放在这个方法内，因此当从 Thread 类继承后，应该覆盖这个方法，把希望并行处理的代码都写在 run()方法中，这样这些代码就能够与程序中的其他线程“同

时”执行。

2.线程的执行

线程类编写好后，要想启动线程的运行，首先需要创建线程类的对象，并通过对象引用去调用 start()方法开启线程的执行，然后由线程执行机制通过 start()方法调用 run()方法。如果不调用 start()方法，线程将永远不会启动。Thread 类中的 run()方法从来不会被显式调用，start()方法也不会被覆盖。

多线程程序的实现过程可参照以下例程：

```
//Threads.java
public class Threads{
    public static void main(String[] args)    {
    System.out.print("main begins  ");
      //创建 Threads1 类的对象
      Threads1 nt1=new Threads1();
      //创建 Threads2 类的对象
      Threads2 nt2=new Threads2();
      //通过 Threads1 类的对象引用调用 start()方法，启动线程执行
      nt1.start();
      //通过 Threads2 类的对象引用调用 start()方法，启动线程执行
      nt2.start();
      System.out.print("main ends  ");
    }
}
//通过继承 Thread 类，编写线程类 Threads1
class Threads1 extends Thread
{   //线程运行时执行的代码
    public void run(){
      for(int i=1;i<=50;i++)
        System.out.print(i+"  ");
    }
}
//通过继承 Thread 类，编写线程类 Threads2
class Threads2 extends Thread{
    public void run()    {
        for(char c='A';c<='Z';c++)
            System.out.print(c+"  ");
        for(char c='a';c<='z';c++)
            System.out.print(c+"  ");
    }
}
```

3.随堂练习

利用多线程技术实现如图 1-19 界面程序，要求当点击“时间显示开始”按钮时，在状态栏显

示系统时间，当点击“时间显示停止”按钮时，停止时间显示。

图 1-19　显示时间程序运行效果图

参考实现代码如下：

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.Date;
import java.text.*;
public classShowTime{
    public static void main(String arg[])    {
        MyFrame frame = new MyFrame();
            frame.setSize(400,300);
            frame.setLocation(200,200);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
        }
    }
class MyFrame extends JFrame implements ActionListener{
  private JTextArea jta;
  private JScrollPane jsp;
  private JLabel jl;
  private JToolBar jtb;
  private JButton jb1=new JButton("时间显示开始");
  private JButton jb2=new JButton("时间显示停止");
  private GetTimeThread timer=null;
  public MyFrame()  {
    super("多线程");
    jta=new JTextArea("在此输入文本");
    jsp = new JScrollPane(jta);
    jl=new JLabel();
    jtb=new JToolBar();
    jtb.add(jb1);
    jtb.add(jb2);
jb1.addActionListener(this);
```

```
        jb2.addActionListener(this);
        getContentPane().add(jtb,BorderLayout.NORTH);
        getContentPane().add(jsp,BorderLayout.CENTER);
        getContentPane().add(jl, BorderLayout.SOUTH);
    }
        public void actionPerformed(ActionEvent e)    {
          if(e.getSource()==jb1)  {
            if(timer==null)  {
            timer=new GetTimeThread();
            timer.start();
        }
    }
        if(e.getSource()==jb2)  {
          if(timer! =null) {
            timer.state=false;
            jl.setText("");
            timer=null;
          }
        }
    }
    class GetTimeThread extends Thread {
        boolean state=true;
        public void run() {
            while(state)
            {
                Date date=new Date();
                DateFormat dateFormatter = DateFormat.getTimeInstance();
                jl.setText(dateFormatter.format(date));
                try{
                  sleep(1000);
                }
                catch(InterruptedException ie)
                {System.out.println(ie.toString());}
            }
        }
    }
}
```

1.4.2 课堂实训

按照项目设计的结论，本项目可分为游戏区模块、游戏控制模块、级别设置模块和帮助模块，本节将按照这个思路逐个介绍该游戏的实现过程。

1.4.2.1　主体框架搭建

1.创建工程

首先,在 Eclipse 中创建本项目的工程,创建过程参见 1.4.1.1 节介绍的内容,本项目的工程名为 snake。由于本项目的模块结构简单,涉及的 Java 类较少,因此在工程中只创建一个 Java 包即可。具体过程可以在创建好的工程名上点击右键,然后选择“新建”→“包”,输入包名为“snake”。效果如图 1-20 所示。

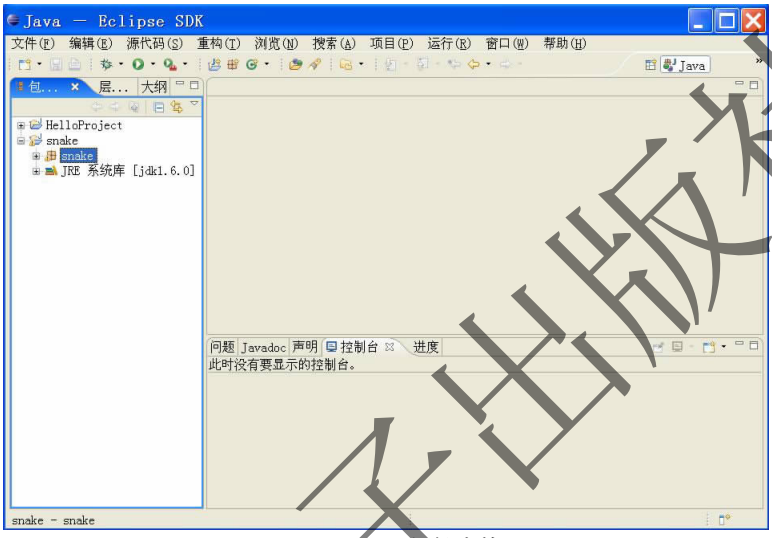

图 1-20　snake 工程创建效果图

2.建立主框架类 MainFrame

在 snake 包中新建主框架类 MainFrame,该类用于显示游戏的主体界面,创建过程为:在 snake 包上点击右键,然后选择“新建”→“类”,输入类名为:“MainFrame”;并输入该类的超类为:“javax.swing.JFrame”;并点击“完成”按钮,具体效果如图 1-21 所示。

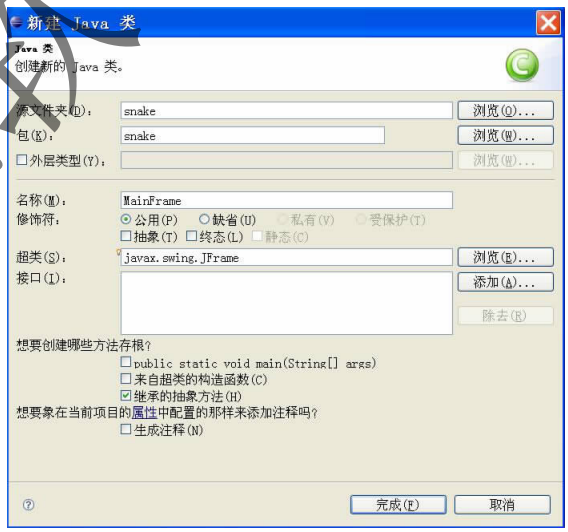

图 1-21　MainFrame 类创建效果图

MainFrame 类创建后,可以在代码编辑区中编写代码。具体效果如图 1-22 所示。

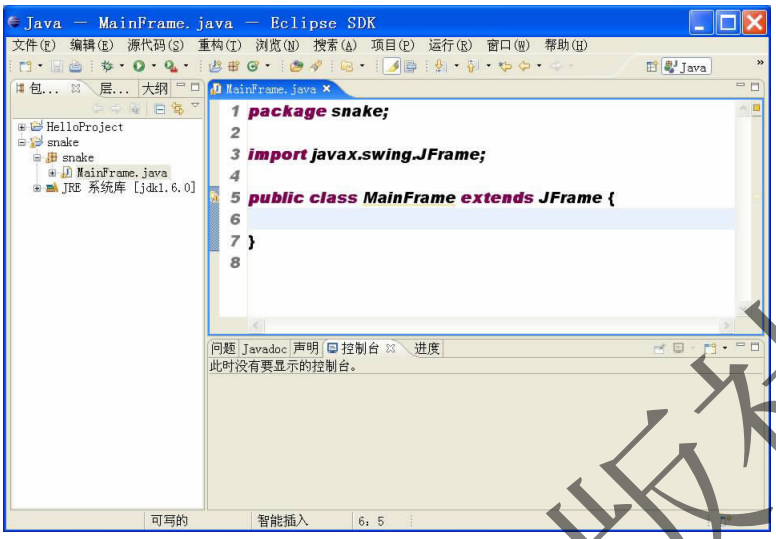

图 1-22 MainFrame 编辑效果图

3.主窗体的实现

根据界面设计,主窗体上应显示等级设置菜单、游戏控制工具栏,以及游戏状态和游戏区等内容。首先,利用 Java Swing 组件中的 JMenuBar、JMenu 和 JRadioButtonMenuItem 类构造难度菜单,然后利用 JTooBar 和 JButton 类构造控制工具栏。游戏状态栏和游戏区可以放到游戏区模块部分来实现。

具体实现代码如下:

```
package snake;

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

public class MainFrame extends JFrame {
    //定义菜单相关属性
    private JMenuBar menuBar = new JMenuBar();
    private JMenu mLevel = new JMenu("难度");
    private JRadioButtonMenuItem miBegin = new JRadioButtonMenuItem("初级");
    private JRadioButtonMenuItem miMiddle = new JRadioButtonMenuItem("中级");
    private JRadioButtonMenuItem miHard = new JRadioButtonMenuItem("高级");

    //定义控制面板相关属性
    private JToolBar toolBar = new JToolBar();
    private JButton jbtStart = new JButton("开始");//游戏开始按钮
    private JButton jbtPause= new JButton("暂停");//游戏暂停按钮
    private JButton jbtStop = new JButton("结束");//游戏退出按钮
    private JButton jbtHelp = new JButton("帮助");//帮助按钮
```

```
        //构造方法,完成主窗体的初始构造
        public MainFrame(){
            //构造等级设定菜单
            this.setJMenuBar(menuBar);
            menuBar.add(mLevel);
            ButtonGroup group = new ButtonGroup();
            group.add(miBegin);
            group.add(miMiddle);
            group.add(miHard);
            mLevel.add(miBegin);
            mLevel.add(miMiddle);
            mLevel.add(miHard);
            miBegin.setSelected(true);

            //构造游戏控制工具栏
            Container contentPane = this.getContentPane();
            contentPane.add(toolBar,BorderLayout.NORTH);
            toolBar.add(jbtStart);
            toolBar.add(jbtPause);
            toolBar.add(jbtStop);
            toolBar.add(jbtHelp);

            //设置按钮初始状态
            jbtPause.setEnabled(false);
            jbtStop.setEnabled(false);
        }
    }
```

4.主类 Snake 的实现

Snake 类为整个贪食蛇游戏程序的主类,负责启动游戏程序和创建并显示主窗体对象。Eclipse 创建类的过程参见 MainFrame 类的创建过程。

具体代码如下:

```
package snake;

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

public class Snake {
    //构造方法
    public Snake() {
        //构造并显示主窗体
        MainFrame frame = new MainFrame();
        frame.setTitle("贪食蛇游戏");
```

```
            frame.setSize(760,584);
            frame.setResizable(false);

        //将主窗体居中显示
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
          frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
          frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2,
                (screenSize.height - frameSize.height) / 2);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    //Main 方法
    public static void main(String[] args) {
        new Snake();
    }
}
```

5.运行贪食蛇游戏

Eclipse 中可以在主类 Snake 上点击右键，然后选择“运行方式”→“Java 应用程序”；主窗体运行后效果如图 1-23 所示。

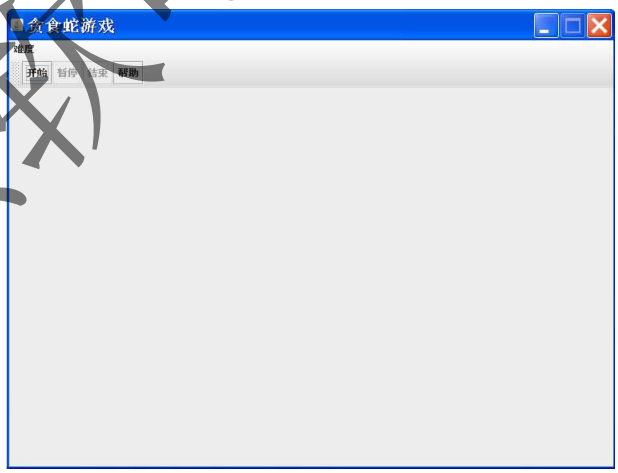

图 1-23 主窗体运行效果图

1.4.2.2 游戏区模块的实现

1.游戏区界面的实现

根据界面设计，游戏区主要用于显示蛇身运动以及游戏状态信息，它作为主窗体的一部分，

可以把游戏区制作成一个面板(JPanel)。游戏状态信息可以用一些标签(JLabel)实现;蛇身可以按照这样的思路来实现:将整个游戏区划分成 30 行、50 列的表格,每个格中放置一个按钮,每个按钮代表一个方块,把代表蛇身的方块显示出来,其他的方块隐藏起来,通过控制按钮的显示和隐藏来模拟蛇身的运动。

创建 PlayPanel 类,实现游戏区模块,具体实现代码如下:

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

public class PlayPanel extends JPanel{
    //状态栏相关属性
    private JLabel jlScore = new JLabel("0");
    private JLabel jlThroughBody = new JLabel("0");
    private JLabel jlThroughWall = new JLabel("0");
    private int score = 0;//当前得分
    private int throughBody = 0;
    private int throughWall = 0;

    //定义游戏区相关属性
    private static final int ROWS = 30;//游戏区行数
    private static final int COLS = 50;//游戏区列数
    private JButton[][] playBlocks;//游戏区的所有方块

    //构造方法
    public PlayPanel(){
        //构造状态栏
        JPanel statusPane = new JPanel();//状态栏面板
        statusPane.add(new JLabel("得分:"));
        statusPane.add(jlScore);
        statusPane.add(new JLabel("穿身宝物:"));
        statusPane.add(jlThroughBody);
        statusPane.add(new JLabel("穿墙宝物:"));
        statusPane.add(jlThroughWall);

        //构造游戏区面板
        JPanel showPane = new JPanel();//显示蛇身运动的游戏区面板
        showPane.setLayout(new GridLayout(ROWS,COLS,0,0));
        //设置边框
        showPane.setBorder(BorderFactory.createEtchedBorder());
        //创建并初始化游戏区方块
        playBlocks = new JButton[ROWS][COLS];
```

```
        for (int i = 0; i < ROWS; i++) {
            for (int j = 0; j < COLS; j++) {
                playBlocks[i][j] = new JButton();
                playBlocks[i][j].setBackground(Color.LIGHT_GRAY);
                playBlocks[i][j].setVisible(true); //将来程序完成时改成 false
                playBlocks[i][j].setEnabled(false);
                showPane.add(playBlocks[i][j]);
            }
        }
            this.setLayout(new BorderLayout());
            this.add(statusPane,BorderLayout.NORTH);
            this.add(showPane,BorderLayout.CENTER);
        }
    }
```

2.将游戏区添加到主窗体 MainFrame 上

创建 PlayPanel 类后,需要在 MainFrame 类中创建游戏区对象,并添加到主窗体的内容窗格中,具体实现代码如下:

```
public class MainFrame extends JFrame {
    ……//省略号部分为已实现部分
    //定义游戏区
    private PlayPanel playPane = new PlayPanel();
    public MainFrame(){
        ……//省略号部分为已实现部分
        //添加到主窗体的内容窗格中
        contentPane.add(playPane,BorderLayout.CENTER);
    }
}
```

运行 Snake 主类显示游戏区界面效果如图 1-24 所示。

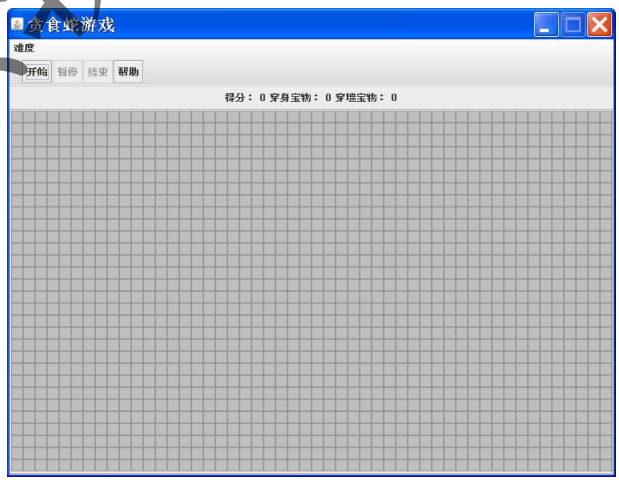

图 1-24 带游戏区的主窗体运行效果图

3.创建蛇身方法的实现

在创建蛇身和移动蛇身时应考虑保存蛇身的长度、蛇身的运动方向、蛇头的运动方向、蛇身每个方块的位置等信息，因此在 PlayPanel 类中分别定义相应的属性；同时，在创建蛇身时只要规定好相应属性的初始值，并将作为蛇身的方块显示出来即可，具体实现代码如下：

```
public class PlayPanel extends JPanel{
    ……
    //蛇身相关属性
    private int length = 3;//蛇身的初始长度
    private int[] rows = new int[ROWS * COLS];//记录蛇身每个方块的行号
    private int[] columes = new int[ROWS * COLS];//记录蛇身每个方块的列号
    public static final int UP = 1, LEFT = 2, DOWN = 3, RIGHT = 4;//贪食蛇运动方向
    private int direction = RIGHT;
    private int lastdirection = RIGHT;
    private boolean lost = false;

    //创建蛇身
    public void createSnake(){
    length=3;//蛇身初始长度
    score = 0;//当前得分
    throughBody = 0;//穿身宝物数
        throughWall = 0;//穿墙宝物数
        lost=false;//游戏结束标志
        direction = RIGHT;//蛇身运动方向
        lastdirection = RIGHT;    //蛇身在改变运动方向前的运动方向
        //初始化蛇身位置
        for (int i = 0; i <= length; i++) {
            rows[i] = 1;
          columes[i] = length - i;
        }
        //显示蛇身
        for (int i = 0; i <= length; i++) {
            playBlocks[rows[i]][columes[i]].setBackground(Color.green);
            playBlocks[rows[i]][columes[i]].setVisible(true);
        }
    }
}
```

4.放置食物方法的实现

放置食物可以先通过生成随机数的方式确定食物的位置，如果该位置与蛇身的位置重叠，就重新生成，直到不重叠为止；然后再通过生成随机数的方式确定食物的类型，根据该类型改变该位置的方块的颜色，并显示出来。

具体实现代码如下：

```
public class PlayPanel extends JPanel{
    ……
    //构造方法
    public PlayPanel(){
    ……
    //将原有 playBlocks[i][j].setVisible(true);参数改为 false
        playBlocks[i][j].setVisible(false);
        ……
}
//在游戏区内随机放置食物
public void createFood(){
    int x = 0;//食物行号
    int y = 0;//食物列号
    //随机生成食物位置,如果新位置是蛇身时,重新生成食物位置
    do{
    //随机生成食物的行
      x = (int) (Math.random() * ROWS);
      //随机生成食物的列
      y = (int) (Math.random() * COLS);
    }while (playBlocks[x][y].isVisible()) ;
    //生成随机数,用于决定食物的类型
        int random = (int) (Math.random() * 10);
        //当随机数小于 7 时,生成普通食物
        if (random < 7) {
         playBlocks[x][y].setBackground(Color.yellow);
        }//当随机数介于 7～9 之间时,生成穿身宝物
        else  if (random < 9) {
          playBlocks[x][y].setBackground(Color.blue);
        }//当随机数大于等于 9 时,生成红色
        else {
          playBlocks[x][y].setBackground(Color.red);
        }
        playBlocks[x][y].setVisible(true);
    }
}
```

5.移动蛇的方法的实现

移动蛇身可以通过去掉蛇尾的方块,在原来的蛇头前添加方块的方式来实现,同时还要考虑以下问题:

(1)在移动蛇身时是否改变方向,改变的方向是否与原来的运动方向相反;

(2)当蛇头吃食物时,应将蛇身加长;

(3)当蛇头碰到墙壁时应判断拥有穿墙宝物的状态;

(4)当蛇头碰到蛇身时应判断拥有穿身宝物的状态;

(5)吃完食物后还应在随机放置下一个食物；

(6)更新状态栏的信息。

具体实现代码如下：

```
public class PlayPanel extends JPanel{
    ……
    //移动蛇
    public void moveSnake(){
        //去掉蛇尾
        playBlocks[rows[length]][columes[length]].setVisible(false);
        playBlocks[rows[length]][columes[length]].setBackground(Color.lightGray);
        //显示除蛇头外蛇身
        for (int i = 0; i < length; i++) {
            playBlocks[rows[i]][columes[i]].setBackground(Color.green);
            playBlocks[rows[i]][columes[i]].setVisible(true);
        }
        //移动除蛇头外蛇身
        for (int i = length; i > 0; i--) {
            rows[i] = rows[i - 1];
            columes[i] = columes[i - 1];
        }
        //根据蛇身运动方向,决定蛇头位置
        switch (direction) {
            case UP:{
                if (lastdirection == DOWN)
                  rows[0] += 1;
                else {
                  rows[0] -= 1;
                  lastdirection = UP;
                }
                break;
            case LEFT: {
              if (lastdirection == RIGHT)
                columes[0] += 1;
              else {
                columes[0] -= 1;
                lastdirection = LEFT;
              }
              break;
            }
            case DOWN: {
              if (lastdirection == UP)
                rows[0] -= 1;
```

```
        else {
          rows[0] += 1;
          lastdirection = DOWN;
        }
        break;
      }
      case RIGHT: {
        if (lastdirection == LEFT)
          columes[0] -= 1;
        else {
          columes[0] += 1;
          lastdirection = RIGHT;
        }
        break;
      }
      }
        //处理蛇头碰到墙壁时操作
      if (rows[0] >= ROWS || rows[0] < 0 || columes[0] >= COLS ||  columes[0] < 0 )
{
     //处理有穿墙宝物时的穿墙操作
        if (throughWall ! = 0) {
        //更改穿墙宝物数,并更新状态栏
      throughWall--;
      jlThroughWall.setText(Integer.toString(throughWall));
      //当蛇头碰到底部墙壁时,蛇头从顶部墙壁重新进入
      if (rows[0] >= ROWS) {
      rows[0] = 0;
      }
      //当蛇头碰到顶部墙壁时,蛇头从底部墙壁重新进入
      else if (rows[0] < 0) {
        rows[0] =  ROWS - 1;
      }
      //当蛇头碰到右侧墙壁时,蛇头从左侧墙壁重新进入
      else if (columes[0] >= COLS) {
      columes[0] = 0;
      }
      //当蛇头碰到左侧墙壁时,蛇头从右侧墙壁重新进入
      else if (columes[0] < 0) {
         columes[0] = COLS - 1;
      }
            } //当没有穿墙宝物时,游戏结束
        else {
```

```
            lost = true;
            return;
        }
    }
        //蛇头碰到蛇身时的处理操作
          if (playBlocks[rows[0]][columes[0]].getBackground().equals(Color.green))
     {
            if (throughBody ! = 0) {
    throughBody--;
    jlThroughBody.setText(Integer.toString(throughBody));
            }
            else {
              lost = true;
              return;
            }
          }
          //蛇头吃完食物后,蛇身加长,并随机显示下一个食物或宝物
          if (playBlocks[rows[0]][columes[0]].getBackground().equals(Color.yellow)
              ||playBlocks[rows[0]][columes[0]].getBackground().equals(Color.blue)
              ||playBlocks[rows[0]][columes[0]].getBackground().equals(Color.red)) {
      length++;
      createFood();
      //更新状态栏
      score += 100;
      jlScore.setText(Integer.toString(score));
        //获得穿墙宝物时的操作
        if (playBlocks[rows[0]][columes[0]].getBackground().equals(Color.blue))
     {
            throughBody++;
            jlThroughBody.setText(Integer.toString(throughBody));
        }
        //获得穿身宝物时的操作
        if (playBlocks[rows[0]][columes[0]].getBackground().equals(Color.red))
     {
            throughWall++;
            jlThroughWall.setText(Integer.toString(throughWall));
          }
          }
    //显示蛇头
    playBlocks[rows[0]][columes[0]].setBackground(Color.green);
    playBlocks[rows[0]][columes[0]].setVisible(true);
```

```
        }
    }
```

6.清空游戏区方法的实现

当游戏结束后需要清空游戏区,该方法可以通过将所有游戏区的方块颜色和可见性全部设置为初始状态来实现。具体实现代码如下:

```
public class PlayPanel extends JPanel{
    ……
    //清空游戏区
    public void clear(){
        score = 0;
        throughBody = 0;
        throughWall = 0;
        jlScore.setText("0");
        jlThroughBody.setText("0");
        jlThroughWall.setText("0");
        for (int i = 0; i < ROWS; i++) {
            for (int j = 0; j < COLS; j++) {
                playBlocks[i][j].setBackground(Color.LIGHT_GRAY);
                playBlocks[i][j].setVisible(false);
            }
        }
    }
}
```

7.获取游戏状态方法的实现

在控制游戏时需要获取游戏的状态,可以通过返回 lost 属性来实现。具体实现代码如下:

```
public class PlayPanel extends JPanel{
    ……
    //获取游戏状态
    public boolean isLost(){
        return lost;
    }
}
```

8.设置蛇身运动方向方法的实现

在控制游戏时需要设定蛇身的运动方向,可以通过设定代表运动方向的 direction 属性来实现。具体实现代码如下:

```
public class PlayPanel extends JPanel{
    //设置蛇的运行方向
    public void setSnakeDirection(int direction){
        this.direction = direction;
    }
}
```

1.4.2.3　游戏控制模块的实现

由于游戏的控制需要在主窗体上完成，因此该模块的功能需要在 MainFrame 类中实现。游戏控制主要体现在两个方面：一是通过键盘控制蛇身的运动方向；二是通过工具栏来控制游戏的开始、暂停和结束。下面从这两个方面分别阐述。

1.通过键盘控制蛇运动方向功能的实现

键盘的控制可以通过在主窗体类 MainFrame 中定义一个键盘事件的监听器类 MainFrameKeyListener 来实现；而且在游戏时，通过键盘只能改变蛇的运动方向，因此在 MainFrameKeyListener 类中只需根据用户的按键重新设定游戏区上蛇身的运动方向即可。在 MainFrameKeyListener 类定义完成后，还需要在主窗体的构造方法中创建 MainFrameKeyListener 类的对象，并注册给键盘事件的事件源，也就是主窗体对象。

具体实现代码如下：

```
public class MainFrame extends JFrame {
    ……//省略号部分为已实现部分
    private boolean isPause = false;//游戏暂停标志
    private boolean isEnd = true;//游戏结束标志
    public MainFrame(){
        ……//省略号部分为已实现部分
        //设置按钮不可获取焦点
        jbtStart.setFocusable(false);
        jbtPause.setFocusable(false);
        jbtStop.setFocusable(false);
        jbtHelp.setFocusable(false);
        //创建键盘事件的监听器对象
        MainFrameKeyListener keyListener = new MainFrameKeyListener();
        //将监听器注册给主窗体
        this.addKeyListener(keyListener);
    }
    //键盘事件的监听器类
    class MainFrameKeyListener extends KeyAdapter{
        public void keyPressed(KeyEvent e) {
            //判断游戏状态
            if (! isEnd && ! isPause) {
                //根据用户按键，设置蛇运动方向
                if (e.getKeyCode() == KeyEvent.VK_UP) {
                playPane.setSnakeDirection(PlayPanel.UP);
                }
                if (e.getKeyCode() == KeyEvent.VK_DOWN) {
                playPane.setSnakeDirection(PlayPanel.DOWN);
                }
                if (e.getKeyCode() == KeyEvent.VK_LEFT) {
                playPane.setSnakeDirection(PlayPanel.LEFT);
```

```
            }
            if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
          playPane.setSnakeDirection(PlayPanel.RIGHT);
            }
          }
        }
      }
}
```

2.游戏开始、暂停和结束功能的实现

游戏的开始、暂停和结束功能的主要任务是控制蛇身的移动，而蛇身的移动与键盘的控制可以同时进行，因此，蛇身的不断移动需要定义一个单独的线程类 SnakeThread 来完成。SnakeThread 类的主要任务是在一定的间隔下不断地移动蛇身，同时能够响应游戏的开始、暂停和结束请求。

SnakeThread 类的实现代码如下：

```
public class MainFrame extends JFrame {
……//省略号部分为已实现部分
class SnakeThread   extends Thread {
    public void run() {
      while (true) {
        try {
          //停顿
          Thread.sleep(speed);
          //当游戏处于正常运行状态,则移动蛇身
          if (! isEnd && ! isPause) {
           playPane.moveSnake();
           if(playPane.isLost()){
           isEnd = true;
           JOptionPane.showMessageDialog(null, "游戏结束!");
           isPause = false;
           playPane.clear();
           jbtStart.setEnabled(true);
           jbtPause.setText("暂停");
           jbtPause.setEnabled(false);
           jbtStop.setEnabled(false);
           }

        }
         }
         catch (Exception ex){}
      }
    }
  }
```

```
}
```

要响应用户对工具栏按钮的操作，还要为工具栏按钮注册动作事件的监听器。该监听器可以通过 MainFrameActionListener 类来实现。

具体实现代码如下：

```
public class MainFrame extends JFrame {
    ……//省略号部分为已实现部分

    SnakeThread thread = new SnakeThread();//游戏主线程
    //构造方法

    public MainFrame(){
        ……//省略号部分为已实现部分
        //创建动作事件的监听器对象
        MainFrameActionListener actionListener = new MainFrameActionListener();
        //将监听器注册给各个控制按钮
        jbtStart.addActionListener(actionListener);
        jbtPause.addActionListener(actionListener);
        jbtStop.addActionListener(actionListener);
    }
    //动作事件的监听器类
    class MainFrameActionListener implements ActionListener{
      public void actionPerformed(ActionEvent e){
          //当用户点击开始时执行
          if(e.getSource()==jbtStart){
          //初始化状态
          isEnd = false;
          isPause = false;
          //创建贪食蛇
          playPane.createSnake();
          //随机摆放食物
          playPane.createFood();
              try {
                //启动游戏
                thread.start();
              }
              catch (Exception ex) {

              }
              jbtStart.setEnabled(false);
           jbtPause.setEnabled(true);
          jbtStop.setEnabled(true);
```

```
            }else if(e.getSource()==jbtPause){
                if (isPause == true ) {
                jbtPause.setText("暂停");
                }else   if (isPause == false) {
                jbtPause.setText("继续");
                }
                isPause = ! isPause;
            }else if(e.getSource()==jbtStop){
            isEnd = true;
                isPause = false;
                playPane.clear();
                jbtStart.setEnabled(true);
                jbtPause.setText("暂停");
                jbtPause.setEnabled(false);
                jbtStop.setEnabled(false);
                }
            }
        }
}
```

1.4.2.4 级别设置模块的实现

由于游戏的级别设置需要在主窗体上完成，因此该模块的功能需要在 MainFrame 类中实现。级别的设定可以通过改变蛇身每次移动的间隔，也就是游戏的速度来实现。具体的实现代码如下：

```
public class MainFrame extends JFrame {
    ……//省略号部分为已实现部分
    public static final int BEGINNER = 1, MIDDLE = 2, EXPERT = 3;//游戏级别常量
    private int level = BEGINNER;//当前游戏级别
    public MainFrame(){
        ……//省略号部分为已实现部分
        //为级别设定按钮注册监听器
        miBegin.addActionListener(actionListener);
        miMiddle.addActionListener(actionListener);
        miHard.addActionListener(actionListener);
        }
        class MainFrameActionListener implements ActionListener{
            public void actionPerformed(ActionEvent e){
                ……//省略号部分为已实现部分
                if(e.getSource()==miBegin){
                speed=300;
                }else if(e.getSource()==miMiddle){
                speed=200;
                }else if(e.getSource()==miHard){
```

```
                    speed=100;
                }
            }
        }
    }
```

1.4.2.5　帮助模块的实现

根据界面设计，帮助界面可以通过定义一个对话框 JDialog 来实现，在对话框上显示帮助信息，然后在 MainFrame 类中弹出该对话框即可。具体实现代码在此不作详细介绍。

1.5　项目改进

上述的贪食蛇游戏实现了基本的游戏功能，在此基础上还可以考虑从以下几个方面进行改进：

(1)为游戏增加障碍功能：在游戏区中设定各种形状的障碍，以增加游戏的难度和乐趣。

(2)为游戏增加排行榜功能：在游戏结束时记录游戏得分，并将得分计入排行榜，当玩家的得分超过排行榜中得分时更新排行榜名次。

(3)改进实现方式：本项目采用在游戏区放置按钮方块的方式来实现蛇身，可以考虑使用在组件上绘图的方式来实现蛇身。