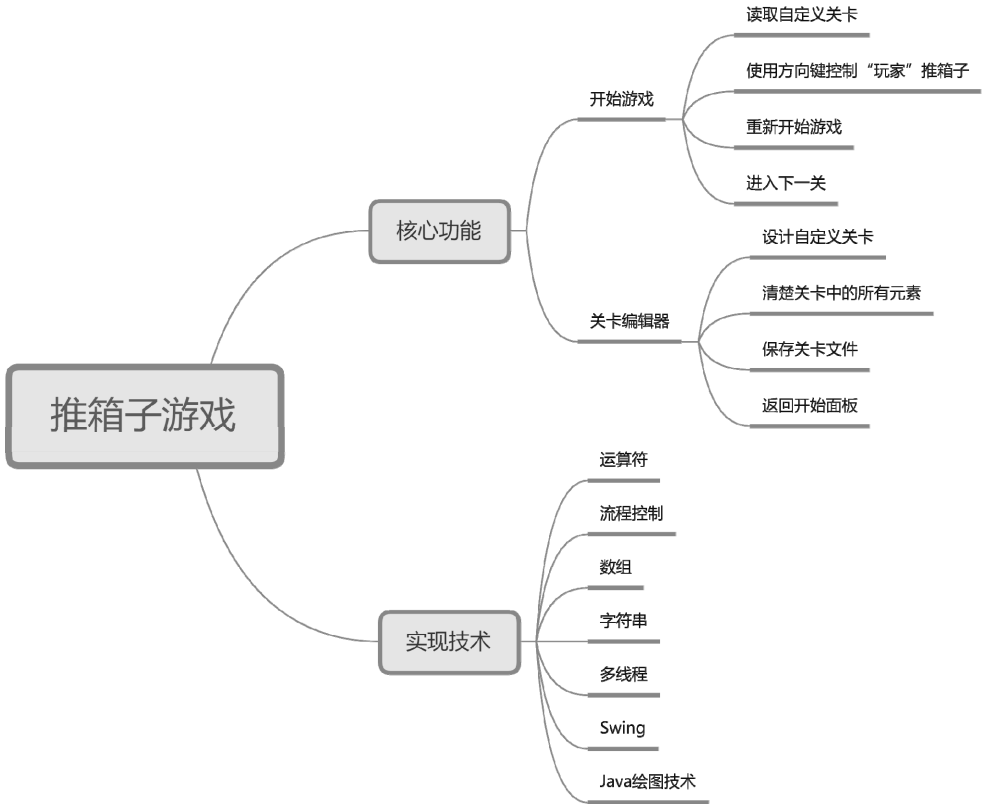
第 1 章 推箱子游戏

——运算符+流程控制+数组+字符串+多线程+Swing

自从休闲益智类游戏问世以来，不论男女老幼，都多了一种休闲娱乐的选择，并对这类游戏抱有极大的热情。推箱子游戏是一款经典的休闲益智类游戏，用户只有找到推箱子的正确路线，才可以把所有的箱子都推至目的地。本章将使用 Java 语言中的 Swing、绘图技术等技术，并结合运算符、流程控制、数组、字符串、多线程等关键技术开发一款推箱子游戏。在这款推箱子游戏中，用户不仅可以挑战预设的关卡，还可以设计并挑战自定义的关卡。

本项目的核心功能及实现技术如下：



1.1 开 发 背 景

经典的推箱子游戏是一款休闲益智类游戏，旨在训练用户的逻辑思考能力。开始游戏后，用户需要在一个空间狭小的仓库里把每一个箱子都推至指定位置。操作稍有不慎，就会出现箱子无法被移动或者通道被堵住的情况。只有巧妙地利用有限的空间和通道，合理地安排箱子移动的次序和位置，才能顺利地完成

任务。

由于 Java 语言提供了一套完整的 GUI 开发框架，因此程序开发人员可以使用 Java 语言开发一些简单的游戏。本章将使用 Java 语言开发一款推箱子游戏，这款游戏包含“开始游戏”和“关卡编辑器”两个功能。其中：“开始游戏”功能通过设计开始面板和游戏面板来实现；“关卡编辑器”功能则通过设计关卡编辑器面板来实现。

本游戏的目标如下：

☑ 本游戏规则简单、操作灵活、难度适中、趣味性强。
☑ 通过连续既定关卡，提升本游戏的可玩性和挑战性。
☑ 通过重新开始游戏，提高本游戏的重玩性。
☑ 通过设计自定义关卡，增强本游戏的耐玩性。

1.2 系 统 设 计

1.2.1 开发环境

本游戏所需的开发环境如下：

☑ 操作系统：推荐 Windows 10、Windows 11 或更高版本，同时兼容 Windows 7（SP1）。
☑ 开发工具：Eclipse。
☑ 开发语言：Java。
☑ 开发环境：JDK 21。

1.2.2 业务流程

用户开始游戏后，即可进入开始面板。

用户在选择“开始游戏”功能后，即可使用方向键控制“玩家”推箱子。用户如果可以把所有的箱子都推至目的地，并且本游戏具有下一个关卡，就能够进入下一个关卡。用户如果未能把所有的箱子都推至目的地，并且本游戏出现箱子无法被移动或者通道被堵住的情况，就需要重新开始游戏。

用户在选择“关卡编辑器”功能后，即可设计自定义关卡。用户先选择墙块、玩家、箱子或者目的地等元素，再使用鼠标把已经选择的元素绘制在指定位置上。用户设计完成自定义关卡后，既可以清除关卡中的所有元素，又可以保存关卡文件，还可以返回开始游戏面板。

推箱子游戏的业务流程如图 1.1 所示。

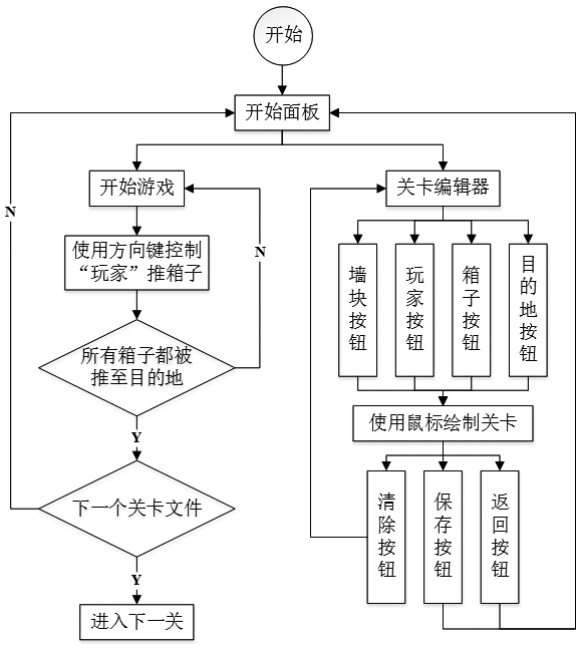


图 1.1 推箱子游戏的业务流程图

1.2.3 功能结构

本游戏的功能结构已经在章首页中给出。作为

一款经典的益智类游戏，它实现的具体功能如下：

- ☑ 开始游戏：用户通过方向键控制“玩家”推箱子，当所有箱子都被推至目的地后，本游戏会进入下一个关卡。如果用户绘制了自定义关卡，则本游戏会在开始游戏后优先进入自定义关卡；在用户通过自定义关卡后，本游戏会进入既定的第一个关卡。
- ☑ 关卡编辑器：用户打开关卡编辑器后，可以使用鼠标左键绘制关卡中的元素。用户可以通过窗体下方的按钮选择要绘制的元素，例如墙块、箱子、玩家、目的地等。在选择要绘制的元素后，如果用户在“关卡编辑器”面板上按住鼠标左键并拖曳，则本游戏会在鼠标划过的区域画满相应的元素。鼠标右键具有擦除关卡中元素的功能，并支持拖曳操作。如果用户单击“清除”按钮，则本游戏会清除关卡中的所有元素。用户完成自定义关卡的绘制后，可以单击“保存”按钮保存关卡文件，并跳转至开始面板。如果用户单击“返回”按钮，则本游戏也会跳转至开始面板。

1.3 技 术 准 备

本游戏主要使用 Java 语言中的运算符、流程控制、数组、字符串、多线程、Swing、绘图技术等关键技术。接下来，我们将简述这些关键技术，并举例说明这些关键技术在本游戏中的具体作用。

- ☑ 运算符：运算符是一些特殊的符号。Java 语言提供了丰富的运算符，如赋值运算符、算术运算符、比较运算符等。例如，在设计“关卡编辑器”面板时，需要使用比较运算符判断鼠标是否在“关卡编辑器”面板范围内，代码如下：

```
boolean inMap = e.getX() > offsetX && e.getX() < 400 + offsetX
        && e.getY() > offsetY + 20 && e.getY() < 400 + offsetY + 20;
```

- ☑ 流程控制：流程控制是一种经过逻辑判断后，决定要执行哪一段代码或者执行哪一个方法的过程。例如，在读取关卡文件前，需要使用流程控制中的 if 语句判断关卡文件是否存在。代码如下：

```
if (!f.exists()) {                                          // 如果文件不存在
    System.err.println("关卡不存在:" + mapName);
    return null;
}
```

- ☑ 数组：数组是具有相同数据类型的一组数据的集合。在程序设计中，引入数组可以更有效地管理和处理数据。根据数组的维数可以将数组分为一维数组、二维数组等。一维数组实质上是一组相同类型数据的线性集合，当在程序中需要处理一组数据，或者传递一组数据时，可以使用这种类型的数组。例如，在读取关卡文件时，可以把读出的字符串拆分成一个 char 型的一维数组。代码如下：

```
char codes[] = tmp.toCharArray();                           // 读出的字符串拆分成字符数组
```

如果一个一维数组中的每个元素本身又是一个数组，那么这个数组就是一个二维数组。二维数组常用于表示表，表中的信息以行和列的形式组织，其中第一个下标代表元素所在的行，第二个下标代表元素所在的列。例如，在编写关卡类时，需要把关卡中的所有元素都记录在一个 RigidBody[][] 二维数组中。代码如下：

```
private RigidBody matrix[][];                               // 用于保存关卡中所有元素的数组
```

- ☑ 字符串：字符串是 Java 程序中经常被处理的对象。在 Java 语言中，字符串是作为 String 类的实例来处理的。以对象的方式处理字符串，将使字符串更加灵活、方便。字符串操作包括连接字符串、获取字符串信息、去除空格、字符串替换、判断字符串的开始与结尾、判断字符串是否相等、按

字典顺序比较两个字符串、字母大小写转换、字符串分割、格式化字符串等。例如，在创建关卡文件时，可以创建一个 StringBuilder 类的对象 data，用于存储将要写入关卡文件中的内容，包括空白区域的占位符、墙的占位符、玩家的占位符、箱子的占位符、目的地的占位符等。代码如下：

```
StringBuilder data = new StringBuilder();                          // 创建一个 StringBuilder 对象，用于关卡文件的内容
for (int i = 0, ilength = arr.length; i < ilength; i++) {
    for (int j = 0, jlength = arr[i].length; j < jlength; j++) {
        RigidBody rb = arr[i][j];                                  // 获取关卡中的模型抽象类对象
        if (rb == null) {                                          // 如果是空对象
            data.append(NULL_CODE);                                // 拼接空白区域的占位符
        } else if (rb instanceof Wall) {                           // 如果是墙
            data.append(WALL_CODE);                                // 拼接墙的占位符
        } else if (rb instanceof Player) {                         // 如果是玩家
            data.append(PLAYER_CODE);                              // 拼接玩家的占位符
        } else if (rb instanceof Box) {                            // 如果是箱子
            data.append(BOX_CODE);                                 // 拼接箱子的占位符
        } else if (rb instanceof Destination) {                    // 如果是目的地
            data.append(DESTINATION_CODE);                         // 拼接目的地的占位符
        }
    }
    data.append("\n");                                             // 拼接换行
}
```

☑ 多线程：Java 语言提供了并发机制，这种机制允许程序开发人员可以在程序中执行多个线程，其中每个线程都能够完成一个功能，并且能够与其他线程并发执行，这种机制被称为多线程。例如，gotoAnotherLevel()方法用于进入其他关卡，该方法包含一个 Thread 线程对象，该对象用于在跳转关卡前延时 0.5 秒，以避免关卡切换太快。代码如下：

```
private void gotoAnotherLevel(int level) {
    frame.removeKeyListener(this);                                 // 主窗体删除本类实现的键盘事件
    // 创建线程，创建 Runnable 接口的匿名类
    Thread t = new Thread(() -> {
        try {
            Thread.sleep(500);                                     // 0.5 秒之后
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (level > GameMapUtil.getLevelCount()) {                 // 如果传入的关卡数大于最大关卡数
            frame.setPanel(new StarPanel(frame));                  // 进入开始面板
            JOptionPane.showMessageDialog(frame, "通关啦！");       // 弹出通关对话框
        } else {
            frame.setPanel(new GamePanel(frame, level));           // 进入对应关卡
        }
    });
    t.start();                                                     // 启动线程
}
```

☑ Swing：在开发 Swing 程序时，窗体是 Swing 组件的承载体。开发 Swing 程序的流程可以被简单地概括为首先通过继承 javax.swing.JFrame 类创建一个窗体，然后向这个窗体中添加组件，最后为添加的组件设置监听事件。JFrame 类的常用构造方法包括以下两种形式：

public JFrame()：创建一个初始不可见、没有标题的窗体。

public JFrame(String title)：创建一个不可见、具有标题的窗体。

例如，在设计本游戏的主窗体时，需要创建一个 Java 类命名为 MainFrame，并继承 JFrame 类，代码如下：

```
public class MainFrame extends JFrame {
}
```

☑ 绘图技术：在 Java 语言中，Graphics 类是所有图形上下文的抽象基类，它允许应用程序在组件以及闭屏图像上进行绘制。Graphics 类封装了基本绘图操作所需的状态信息，主要包括颜色、字体、画笔、文本、图像等。

虽然 Graphics 类实现的功能非常有限，但是继承 Graphics 类的 Graphics2D 类实现的功能非常强大，因此 Graphics2D 是推荐使用的绘图类。例如，在设计开始面板时，先声明一个 Graphics2D 类的对象，再通过 paintImage()方法在背景图片上绘制内容，代码如下：

```
Graphics2D g2;                                                    // 图片绘图对象
…// 省略部分代码
private void paintImage() {
    g2.drawImage(GameImageUtil.backgroundImage, 0, 0, this);
    g2.setColor(Color.BLACK);                                     // 使用黑色
    g2.setFont(new Font("黑体", Font.BOLD, 40));                  // 字体
    g2.drawString("开始游戏", 230, y1 + 30);                      // 绘制第一个选项的文字
    g2.drawString("关卡编辑器", 230, y2 + 30);                    // 绘制第二个选项的文字
    g2.drawImage(GameImageUtil.playerImage, x, y, this) ;         // 将玩家图片作为选择图标
}
```

《Java 从入门到精通（第 7 版）》详细地讲解了运算符、流程控制、数组、字符串、多线程、Swing、绘图技术等关键技术。对这些知识不太熟悉的读者，可以参考该书的相关章节进行深入学习。

1.4 工具类设计

开发项目时，编写公共类可以减少重复代码的编写，进而提高代码的重用性和维护性。本游戏创建了两个公共类文件：GameImageUtil.java（图片工具类）和 GameMapUtil.java（关卡工具类）。接下来，我们将分别对这两个公共类中的方法进行详细介绍。

1.4.1 图片工具类

GameImageUtil 图片工具类用于统一管理本游戏所使用的图片。本游戏的所有图片都被放置在 com.mr.image 包中，并使用静态常量 IMAGE_PATH 来记录该包的路径。GameImageUtil 类提供了玩家图片、箱子未到达目的地图片、箱子已到达目的地图片、墙图片、目的地图片和开始面板的背景图片，代码如下：

```
private static final String IMAGE_PATH = "src/com/mr/image";      // 图片存放的路径
public static BufferedImage playerImage;                          // 玩家图片
public static BufferedImage boxImage1;                            // 箱子未到达目的地图片
public static BufferedImage boxImage2;                            // 箱子已到达目的地图片
public static BufferedImage wallImage;                            // 墙图片
public static BufferedImage destinationImage;                     // 目的地图片
public static BufferedImage backgroundImage;                      // 背景图片
```

由于所有图片对象均为静态属性，它们不能在构造方法中被直接赋值。因此，本类使用静态代码块为这些图片对象进行初始化赋值。静态代码块代码如下：

```
static {
    try {
        playerImage = ImageIO.read(new File(IMAGE_PATH, "player.png"));
        boxImage1 = ImageIO.read(new File(IMAGE_PATH, "box1.png"));
        boxImage2 = ImageIO.read(new File(IMAGE_PATH, "box2.png"));
        wallImage = ImageIO.read(new File(IMAGE_PATH, "wall.png"));
        destinationImage = ImageIO
                .read(new File(IMAGE_PATH, "destination.png"));
```

```
            backgroundImage = ImageIO
                    .read(new File(IMAGE_PATH, "background.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

1.4.2 关卡工具类

GameMapUtil 是一个关卡工具类，它作为本游戏读写关卡文件的接口，封装了所有与关卡文件相关的操作。

1. 静态常量

GameMapUtil 类定义了三种类型的静态常量：关卡文件中各元素的占位符、关卡文件的存放路径以及自定义关卡的名称。其中：占位符常量和关卡文件的存放路径常量是私有的，因为它们仅在类内部使用；而自定义关卡名称常量是公有的，因为它需要在类的外部（如地区编辑器中）使用。所有静态常量如下：

```
private static final char NULL_CODE = '0';                          // 空白区域使用的占位符
private static final char WALL_CODE = '1';                          // 墙使用的占位符
private static final char BOX_CODE = '2';                           // 箱子使用的占位符
private static final char PLAYER_CODE = '3';                        // 玩家使用的占位符
private static final char DESTINATION_CODE = '4';                   // 目的地使用的占位符
private static final String MAP_PATH = "src/com/mr/map";            // 关卡存放的路径
public static final String CUSTOM_MAP_NAME = "custom";              // 自定义关卡名称
```

2. 创建关卡文件

createMap()方法用于创建关卡文件。该方法在关卡文件中使用占位符来记录各个游戏元素所在的位置。该方法接收两个参数：第一个参数 arr 是一个数组，它表示关卡中的模型矩阵；第二个参数 mapName 表示要创建的关卡文件的名称。该方法会解析 arr 数组中的所有模型对象，并根据模型对象所属的类型，在字符串 data 中填充相应的占位符。最后，该方法使用文件输出流将字符串 data 写入指定的关卡文件中。

该方法的具体代码如下：

```
static public void createMap(RigidBody[][] arr, String mapName) {
    StringBuilder data = new StringBuilder();                       // 关卡文件将要写入的内容
    for (int i = 0, ilength = arr.length; i < ilength; i++) {
        for (int j = 0, jlength = arr[i].length; j < jlength; j++) {
            RigidBody rb = arr[i][j];                               // 获取关卡中的模型对象
            if (rb == null) {                                       // 如果是空对象
                data.append(NULL_CODE);                             // 拼接空白区域的占位符
            } else if (rb instanceof Wall) {                        // 如果是墙
                data.append(WALL_CODE);                             // 拼接墙的占位符
            } else if (rb instanceof Player) {                      // 如果是玩家
                data.append(PLAYER_CODE);                           // 拼接玩家的占位符
            } else if (rb instanceof Box) {                         // 如果是箱子
                data.append(BOX_CODE);                              // 拼接箱子的占位符
            } else if (rb instanceof Destination) {                 // 如果是目的地
                data.append(DESTINATION_CODE);                      // 拼接目的地的占位符
            }
        }
        data.append("\n");                                          // 拼接换行
    }
    File mapFile = new File(MAP_PATH, mapName);                     // 创建关卡文件对象
    // 开始文件输出流
    try (FileOutputStream fos = new FileOutputStream(mapFile);) {
        fos.write(data.toString().getBytes());                      // 将字符串写入文件中
```

```
            fos.flush();                                    // 刷新输出流
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

3．读取关卡文件数据

本游戏需要通过 readMap()方法将关卡文件数据转为 Map 关卡类对象。该方法中的 mapName 参数指定了要读取的关卡文件的名称。本游戏的默认关卡以数字命名，而用户自己绘制的关卡则使用其原有的 CUSTOM_MAP_NAME 属性值命名，因此该方法参数的类型是字符串。

readMap()方法通过文件输入流逐行读取关卡文件中的字符，然后创建 20×20 的模型类数组 data。接着，该方法判断读出的字符属于哪种模型类型，并在数组 data 对应的位置上创建模型对象。通过这种方式，该方法将文件数据转换为模型矩阵数组。最后，该方法根据填充好的数组 data 创建 Map 关卡对象，从而完成读取关卡文件的功能。

方法的具体代码如下：

```
public static Map readMap(String mapName) {
    File f = new File(MAP_PATH, mapName);                   // 获取关卡文件对象
    if (!f.exists()) {                                      // 如果文件不存在
        System.err.println("关卡不存在:" + mapName);
        return null;
    }
    RigidBody[][] data = new RigidBody[20][20];             // 关卡数组
    // 开启缓冲输入流
    try (FileInputStream fis = new FileInputStream(f);
            InputStreamReader isr = new InputStreamReader(fis);
            BufferedReader br = new BufferedReader(isr)) {
        String tmp = null;                                  // 读取一行数据的临时字符串
        int row = 0;                                        // 当前读取的行数
        while ((tmp = br.readLine()) != null) {             // 循环读取一行包含内容的字符串
            char codes[] = tmp.toCharArray();               // 将读取的字符串转换为字符数组
            // 循环字符数组，并保证读取的行数不超过 20
            for (int i = 0; i < codes.length && row < 20; i++) {
                RigidBody rb = null;                        // 准备保存到关卡数组中的模型对象
                switch (codes[i]) {                         // 判断读出的字符
                    case WALL_CODE :                        // 如果是墙的占位符
                        rb = new Wall();                    // 模型以墙的形式实例化
                        break;
                    case BOX_CODE :                         // 如果是箱子的占位符
                        rb = new Box();                     // 模型以箱子的形式实例化
                        break;
                    case PLAYER_CODE :                      // 如果是玩家的占位符
                        rb = new Player();                  // 模型以玩家的形式实例化
                        break;
                    case DESTINATION_CODE :                 // 如果是目的地的占位符
                        rb = new Destination();             // 模型以目的地的形式实例化
                        break;
                }
                data[row][i] = rb;                          // 将模型对象保存到关卡数组中
            }
            row++;                                          // 读取的行数递增
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return new Map(data);                                   // 返回包含此关卡对象
}
```

除了之前的方法，还有一个重载的读取关卡文件数据的方法，它可以直接接收一个 int 类型变量作为参数。使用 int 参数有利于本游戏计算关卡数。重载方法会自动将 int 类型转换为 String 类型，代码如下：

```
public static Map readMap(int mapNum) {
    // 将数字转换为字符串，调用重载方法
    return readMap(String.valueOf(mapNum));
}
```

4．读取自定义关卡文件

readCustomMap()方法专门用于读取用户绘制的自定义关卡文件。由于自定义文件的文件名是固定的，因此该方法不需要传入任何参数。

readCustomMap()方法首先会检查用户是否创建了自定义关卡。如果没有创建，则返回 null；否则调用 readMap()方法读取自定义文件。

readCustomMap()方法具体代码如下所示：

```
public static Map readCustomMap() {
    File f = new File(MAP_PATH, CUSTOM_MAP_NAME);          // 创建自定义关卡文件对象
    if (!f.exists()) {                                     // 如果文件不存在
        return null;
    }
    Map map = readMap(CUSTOM_MAP_NAME);                    // 读取自定义文件中的数据
    return map;                                            // 返回自定义关卡对象
}
```

5．获取总关卡数

因为本游戏通关一个关卡之后会自动进入下一个关卡，所以需要判断本游戏共有多少关卡，当通过最后一个关卡之后，本游戏要回到开始界面。getLevelCount()方法就是用来返回总关卡数的。

getLevelCount()方法会读取 MAP_PATH 路径下的所有文件。如果这些文件中包含用户绘制的自定义关卡，则返回文件总数−1 作为总关卡数；否则返回总文件数。如果 MAP_PATH 路径表示的不是某个文件夹，则该方法直接返回 0。

方法的具体代码如下：

```
public static int getLevelCount() {
    File dir = new File(MAP_PATH);                         // 读取关卡存放路径
    if (dir.exists()) {                                    // 如果该路径是文件夹
        File maps[] = dir.listFiles();                     // 获取该文件夹下的所有文件
        for (File f : maps) {                              // 遍历这些文件
            if (CUSTOM_MAP_NAME.equals(f.getName())) {     // 如果存在自定义关卡
                return maps.length - 1;                    // 总关卡数 = 文件数 -1
            }
        }
        return maps.length;                                // 总关卡数 = 文件数
    }
    return 0;                                              // 没有关卡
}
```

6．读取自定义关卡文件

当用户重启本游戏之后，会自动将清除以前创建的自定义关卡。clearCustomMap()方法用于实现此功能，该方法会在主窗体创建时被调用，以清除过去的自定义关卡文件。

方法的具体代码如下：

```
public static void clearCustomMap() {
    File f = new File(MAP_PATH, CUSTOM_MAP_NAME);          // 创建自定义关卡文件对象
    if (f.exists()) {                                      // 如果文件存在
```

```
            f.delete();                                    // 删除
        }
    }
```

1.5 模型类设计

模型类是指对本游戏中出现的实物进行封装操作后而成的类。本游戏中有两种模型类：通用模型类和关卡类。接下来，我们分别介绍这两种类。

1.5.1 模型抽象类

本游戏会包含多种类型的模型，这些模型都具有一些共同特征，例如有坐标和图片，将这些共同特征封装成一个模型抽象类作为公共父类，可以减少其他模型的代码量。RigidBody 类就是这样一个模型抽象类，下面从四个方面介绍该类。

1．属性

RigidBody 类有三个属性，分别是模型在游戏面板中的横坐标 x 和纵坐标 y，以及模型使用的图片对象 image，代码如下：

```
public abstract class RigidBody {
    public int x;                                          // 横坐标索引
    public int y;                                          // 纵坐标索引
    private Image image;                                   // 图片
}
```

2．构造方法

RigidBody 类提供了两种构造方法。其中，第一种构造方法仅初始化图片，代码如下：

```
public RigidBody(Image image) {
    this.image = image;
}
```

第二种构造方法仅初始化横纵坐标，并且该构造方法会被箱子类所使用，代码如下：

```
public RigidBody(int x, int y) {
    this.x = x;
    this.y = y;
}
```

3．获取模型图片，重设模型图片

虽然 RigidBody 类已经提供了初始化图片的构造方法，但在游戏过程中模型的图片可能会发生变化。因此除了提供图片的 getter 方法，还需要提供 setter 方法，以确保模型的图片可以灵活地被读取和替换。代码如下：

```
public Image getImage() {
    return image;
}
public void setImage(Image image) {
    this.image = image;
}
```

4．重写 equals()方法

在本游戏中，需要判断某个模型对象是否已经被包含在关卡中。由于本游戏使用 ArrayList 来保存关卡中对应的模型对象，因此要使用 ArrayList 的 contains()方法来判断是否包含某个模型的话，就必须重写 RigidBody 类的 equals()方法来指定比较规则。

本游戏判断两个模型对象是否是同一个元素时，除了要看两个模型对象所属模型类是否一致，还要看它们的坐标是否重合。因此，在 equals()方法中，类型和横纵坐标被作为判断标准。只要类型相同且坐标重合的模型对象，本游戏就会将其视为同一个元素，例如两个处在同一个位置的墙块会被认为是同一个元素。重写后的 equals()方法代码如下：

```
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    RigidBody other = (RigidBody) obj;
    if (x != other.x)
        return false;
    if (y != other.y)
        return false;
    return true;
}
```

1.5.2 墙块类

墙块在本游戏中会阻挡玩家和箱子移动。Wall 类是本游戏的墙块类。墙块使用的图片是由图片工具类 GameImageUtil.wallImage 提供的灰色方块，其效果如图 1.2 所示。墙块类直接使用父类构造方法来为图片赋值。


（灰色）

图 1.2 本游戏中的墙块效果图

墙块类代码如下：

```
public class Wall extends RigidBody {
    public Wall() {
        super(GameImageUtil.wallImage);
    }
}
```

1.5.3 目的地类

目的地在本游戏中标示箱子需要摆放的目标位置，当所有箱子都被摆放在目的地之后，本游戏会进入下一个关卡。Destination 类是本游戏的目的地类。目的地使用的图片是由图片工具类 GameImageUtil.destinationImage 提供的绿色圆圈，其效果如图 1.3 所示。目的地类直接使用父类构造方法来为图片赋值。


（绿色）

图 1.3 本游戏中的目的地效果图

目的地类代码如下：

```
public class Destination extends RigidBody {
    public Destination() {
        super(GameImageUtil.destinationImage);
    }
}
```

1.5.4 玩家类

玩家类是由用户控制的“玩家”角色，该角色可以在上、下、左、右四个方向上自由移动，并且具有推动箱子的能力。Player 类是本游戏的玩家类。玩家类使用的图片是由图片工具类 GameImageUtil.playerImage 提供的蓝色方块，其效果如图 1.4 所示。玩家类直接使用父类构造方法来为图片赋值。

图 1.4 本游戏中的玩家效果图

玩家类代码如下：

```
public class Player extends RigidBody{
    public Player() {
        super(GameImageUtil.playerImage);
    }
}
```

1.5.5 箱子类

箱子是本游戏中可被“玩家”推着移动的元素，在所有箱子都被推至目的地后，本游戏会进入下一个关卡。Box 类是本游戏的箱子类。为了让用户容易区分箱子的到达状态，图片工具类提供两种箱子图片：未到达目的地的黄色箱子 GameImageUtil.boxImage1，效果如图 1.5 所示；已到达目的地红色箱子 GameImageUtil.boxImage2，效果如图 1.6 所示。

（黄色）

图 1.5 箱子未达到目的地的效果图

（红色）

图 1.6 箱子已到达目的地的效果图

为了让箱子能够灵活地更换图片，Box 类提供了 arrived 属性，用于记录箱子是否已经到达目的地。此外，Box 类还提供了 arrive()方法（该方法在箱子到达目的地时被触发）和 leave()方法（该方法在箱子离开目的地时被触发）。这两个方法被触发后，箱子的状态和图片可以被更换。

箱子类的代码如下：

```
public class Box extends RigidBody {
    private boolean arrived = false;                    // 是否到达目的地
    public Box() {
        super(GameImageUtil.boxImage1);
    }
    public Box(int x, int y) {
        super(x,y);
        setImage(GameImageUtil.boxImage1);
    }
    public void arrive() {                              // 到达
        setImage(GameImageUtil.boxImage2);
        arrived = true;
    }
    public void leave() {                               // 离开
        setImage(GameImageUtil.boxImage1);
        arrived = false;
    }
    public boolean isArrived() {
        return arrived;
    }
}
```

1.5.6 关卡类

Map 类是本游戏的关卡类，该类用来封装和加工关卡中的所有元素。关卡中出现的所有元素都是 RigidBody 模型类的子类，因此它们都被记录在一个 RigidBody[][]二维数组中。关卡在类初始化的时候会单独把玩家对象和箱子对象从二维数组中抽出来，并对它们单独进行保存，这样方便本游戏随时修改这两种元素的坐标。下面将从 4 方面介绍关卡类。

1．属性

关卡类有三个属性，分别用于记录三种数据：matrix[][]用于记录关卡中的所有元素，包含墙块、目的地和空白区域；player 用于记录玩家对象及其位置；boxes 用于记录关卡中所有的箱子对象及其位置。代码如下：

```
private RigidBody matrix[][];                                    // 用于保存关卡中所有元素的数组
private Player player;                                           // 关卡中的玩家对象
private ArrayList<Box> boxes;                                    // 关卡中包含的箱子列表
```

2．构造方法

构造方法的参数是一个二维数组，用于保存关卡中的所有元素。这个二维数组由关卡工具类从关卡文件中读取出来。关卡类在构造时要对这个二维数组进行加工，加工的代码位于 init()关卡初始化方法中。构造方法的代码如下：

```
public Map(RigidBody matrix[][]) {
    this.matrix = matrix;
    player = new Player();
    boxes = new ArrayList<>();
    init();                                                      // 关卡初始化
}
```

3．关卡初始化

因为传入的、用于保存关卡中所有元素的数组是包含玩家和箱子的，所以为了方便操作玩家和箱子的坐标，在创建关卡的同时，我们会将这两种类型的对象提取出来。init()方法是初始化关卡的方法，该方法会找出关卡数组中玩家和箱子的位置，并单独记录这些位置，同时清空原数组中玩家和箱子的对象。方法的代码如下：

```
private void init() {
    // 遍历关卡数组
    for (int i = 0, ilength = matrix.length; i < ilength; i++) {
        for (int j = 0, jlength = matrix[i].length; j < jlength; j++) {
            RigidBody rb = matrix[i][j];                         // 读出模型对象
            if (rb instanceof Player) {                          // 如果是玩家
                player.x = i;                                    // 记录用户横坐标索引
                player.y = j;                                    // 记录用户纵坐标索引
                matrix[i][j] = null;                             // 将该索引下的模型对象清除
            } else if (rb instanceof Box) {
                Box box = new Box(i, j);                         // 创建对应该横纵坐标索引的箱子对象
                boxes.add(box);                                  // 箱子列表保存此箱子
                matrix[i][j] = null;                             // 将该索引下的模型对象清除
            }
        }
    }
}
```

4．获取关卡中的所有元素

关卡类中的每一个属性都提供了对应的 getter()方法。获取关卡中的所有元素的代码如下：

```
public RigidBody[][] getMapData() {
    return matrix;
}
```

获取关卡中的玩家对象代码如下：

```
public Player getPlayer() {
    return player;
}
```

获取关卡中的箱子列表的代码如下：

```
public ArrayList<Box> getBoxes() {
    return boxes;
}
```

1.6 主窗体设计

主窗体是 Swing 程序中不可或缺的部分，它在人机交互中扮演着至关重要的角色。通过主窗体，用户能够看到游戏画面、操作游戏角色，从而实现与游戏的互动。主窗体除了提供可视化的最外层容器，还承担加载键盘事件监听器和鼠标事件监听器的任务。主窗体会被作为参数传入每个面板中，在面板类中编写的键盘事件和鼠标事件都必须交给主窗体对象加载，以确保这些事件能够被正确地监听到。

本游戏主窗体的设计过程如下：

（1）创建一个 Java 类，命名为 MainFrame，并让它继承自 JFrame 类，代码如下：

```
public class MainFrame extends JFrame {
}
```

（2）在类中创建两个属性，分别用于记录窗体的宽度和高度。本游戏界面采用 600（宽度）×600（高度）的尺寸，但是考虑到主窗体有外部边界，所以主窗体的宽度和高度分别采用 605 和 627。代码如下：

```
private int width = 605;                    // 不可调整大小时的宽度，可调整大小宽度选择 616
private int height = 627;                   // 不可调整大小时的高度，可调整大小高度选择 638
```

（3）编写主窗体的构造方法，在该构造方法中设置窗体的宽高、位置等属性。使用 Toolkit 工具类获取屏幕尺寸，并计算窗体居中显示的坐标。最后执行三步初始化本游戏的操作：添加监听、载入开始面板、删除自定义关卡。该构造方法的具体代码如下：

```
public MainFrame() {
    setTitle("推箱子");                                    // 设置标题
    setResizable(false);                                   // 不可调整大小
    setSize(width, height);                                // 设置宽高
    Toolkit tool = Toolkit.getDefaultToolkit();            // 创建系统该默认组件工具包
    Dimension d = tool.getScreenSize();                    // 获取屏幕尺寸，赋给一个二维坐标对象
    // 让主窗体在屏幕中间显示
    setLocation((d.width - getWidth()) / 2, (d.height - getHeight()) / 2);
    setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);         // 单击关闭窗体不做任何操作
    addListener();                                         // 添加监听
    setPanel(new StarPanel(this));                         // 载入开始面板
    GameMapUtil.clearCustomMap();                          // 删除自定义关卡
}
```

（4）构造方法中调用的 addListener()方法用于为窗体添加监听，在该方法中为窗体添加了窗体事件，在用户关闭窗体时，会弹出选择框让用户确认关闭操作，如果用户选择“是”，则关闭本游戏，否则不做任何操作。这种确认关闭的功能可以防止用户误关闭本游戏。addListener()方法的代码如下：

```
private void addListener() {
    addWindowListener(new WindowAdapter() {                    // 添加窗体事件监听
        public void windowClosing(WindowEvent e) {             // 窗体关闭时
            // 弹出选择对话框，并记录用户选择
            int closeCode = JOptionPane.showConfirmDialog(MainFrame.this,
                    "是否退出游戏？", "提示！", JOptionPane.YES_NO_OPTION);
            if (closeCode == JOptionPane.YES_OPTION) {         // 如果用户选择确定
                System.exit(0);                                // 关闭本游戏
            }
        }
    });
}
```

（5）构造方法中调用的 setPanel()方法是用于切换游戏场景的关键方法。该方法中的 panel 参数就是要显示的面板，传入 panel 之后，主窗体会获取主容器对象，然后删除容器中已存在的所有组件，最后重新将 panel 放入容器中并重新进行校验、显示。这样就完成了场景切换的功能。setPanel()方法的代码如下：

```
public void setPanel(JPanel panel) {
    Container c = getContentPane();                            // 获取主容器对象
    c.removeAll();                                             // 删除容器中所有组件
    c.add(panel, BorderLayout.CENTER);                         // 容器添加面板
    c.validate();                                              // 容器重新验证所有组件
    c.repaint();
}
```

1.7 功 能 设 计

在设计完毕主窗体后，会发现主窗体载入了一个开始面板。在开始面板上：如果用户选择“开始游戏”功能，则本游戏会跳转到游戏面板，开始闯关；如果用户选择“关卡编辑器”功能，则本游戏会跳转到关卡编辑面板，设计自定义关卡。接下来，我们将讲解用于实现“开始游戏”功能的开始面板、游戏面板，以及用于实现“关卡编辑器”功能的关卡编辑面板。

1.7.1 开始游戏

1. 开始面板

开始面板是用户启动本游戏后看到的第一个面板。开始面板中显示了本游戏的标题和功能选项，用户可以通过键盘的上下键来控制蓝色方块，进而对选项进行选择，按 Enter 键即可确认选项。开始面板的效果如图 1.7 所示。

图 1.7 开始面板效果

开始面板的设计过程如下：

（1）创建一个名为 StarPanel 的 Java 类，并让它继承自 JPanel 面板类，同时实现 KeyListener 键盘事件监听接口。在实现接口的同时要实现该接口中的三个抽象方法。这些方法的代码将在后续进行补充。StarPanel 类的代码如下：

```
public class StarPanel extends JPanel implements KeyListener {
    public void keyPressed(KeyEvent e) {
}
    public void keyReleased(KeyEvent e) {
    }
    public void keyTyped(KeyEvent e) {
    }
}
```

（2）在类中创建属性，分别用于记录主窗体对象、面板背景图片、背景图片的绘图对象以及选择图标的可用坐标，代码如下：

```
MainFrame frame;                                    // 主窗体
BufferedImage image;                                // 面板中显示的图片
Graphics2D g2;                                      // 图片绘图对象
int x = 160;                                        // 图标的横坐标
int y;                                              // 图标的纵坐标
final int y1 = 320;                                 // 第一个选项的纵坐标
final int y2 = 420;                                 // 第二个选项的纵坐标
```

（3）在构造方法中，需要传入主窗体对象，以便能够为窗体添加键盘监听、修改窗体标题功能。构造方法中调用主窗体的 setFocusable(true)方法是为了防止从关卡编辑器返回开始面板造成主窗体丢失焦点的问题。构造方法在最后定义了选择图标的 y 坐标，让图标停留在第一个选项上。构造方法的代码如下：

```
public StarPanel(MainFrame frame) {
    this.frame = frame;
    this.frame.addKeyListener(this);
    this.frame.setFocusable(true);
    this.frame.setTitle("推箱子");
    // 图片使用 600*600 的彩图
    image = new BufferedImage(600, 600, BufferedImage.TYPE_INT_RGB);
    g2 = image.createGraphics();
    y = y1;                                         // 默认选择第一个选项
}
```

（4）image 是开始面板中显示的图片，paintImage()方法用于在这张图片上绘制内容，其中包括绘制背景图片、绘制选项文字和选项图标。因为选项图标的纵坐标是变量，所以每次绘制时，图标都可能出现在不同的位置上。paintImage()方法的代码如下：

```
private void paintImage() {
    g2.drawImage(GameImageUtil.backgroundImage, 0, 0, this);
    g2.setColor(Color.BLACK);                       // 使用黑色
    g2.setFont(new Font("黑体", Font.BOLD, 40));    // 字体
    g2.drawString("开始游戏", 230, y1 + 30);        // 绘制第一个选项的文字
    g2.drawString("关卡编辑器", 230, y2 + 30);      // 绘制第二个选项的文字
    g2.drawImage(GameImageUtil.playerImage, x, y, this) ;   // 将玩家图片作为选择图标
}
```

（5）绘制完 image 图片之后，需要将该图片显示到面板上。重写 paint()绘图方法可以实现此功能。paint()方法是由面板类的父类（javax.swing.JComponent 组件抽象类）提供的，该方法在展示面板时会自动调用。重写的 paint()方法的代码如下：

```
public void paint(Graphics g) {
    paintImage();                                   // 绘制图片
    g.drawImage(image, 0, 0, this);                 // 将图片绘制到面板中
}
```

（6）开始面板实现键盘事件中的 keyPressed()方法，该方法会在用户按下键盘上的任意按键时被触发。在该方法中，使用 KeyEvent 键盘事件对象获取按下的按键的 keyCode 值。如果按下的是 Enter 键，则根据选择图标的 y 坐标判断用户选中的选项，进入对应的面板中；如果按下的是上箭头键或下箭头键，则更换

选择图标的 y 坐标。接口中剩下的 keyReleased()方法和 keyTyped()方法不需要实现。keyPressed()方法的代码如下：

```
public void keyPressed(KeyEvent e) {
    int key = e.getKeyCode();                                  // 获取按键的编码
    switch (key) {                                             // 判断按键
        case KeyEvent.VK_ENTER :                               // 如果是 Enter 键
            switch (y) {                                       // 判断图标的坐标
                case y1 :                                      // 如果选中第一个选项
                    frame.removeKeyListener(this);             // 删除当前键盘事件
                    frame.setPanel(new GamePanel(frame, 0));   // 进入游戏面板
                    break;
                case y2 :                                      // 如果选中第二个选项
                    frame.removeKeyListener(this);             // 删除当前键盘事件
                    frame.setPanel(new MapEditPanel(frame));   // 进入关卡编辑器面板
                    break;
            }
            break;
        case KeyEvent.VK_UP :                                  // 如果是上箭头键，就采用下箭头键的逻辑
        case KeyEvent.VK_DOWN :                                // 如果是下箭头键
            if (y == y1) {                                     // 如果图标选中第一个选项
                y = y2;                                        // 更换选中第二个选项
            } else {
                y = y1;                                        // 更换险种第一个选项
            }
            break;
    }
    repaint();                                                 // 重绘面板
}
```

2．游戏面板

游戏面板是本游戏的核心面板，本游戏的所有算法均在此面板中实现。用户可以在游戏面板中通过键盘的上下左右箭头按键来控制玩家角色移动，玩家在移动的同时可以推动箱子一起移动，当所有的箱子都到达目的地之后，玩家还要进入下一个关卡。如果在游戏过程中出现将箱子推入死胡同的情况，用户可以按 Esc 键来重新开始当前关卡。

游戏面板中的模型图片与关卡编辑器中的模型图片有些许差别：关卡编辑器中的每个图片的大小都是 20×20 像素，而游戏面板中每个图片的大小都是 30×30 像素。

游戏面板的效果如图 1.8 所示。

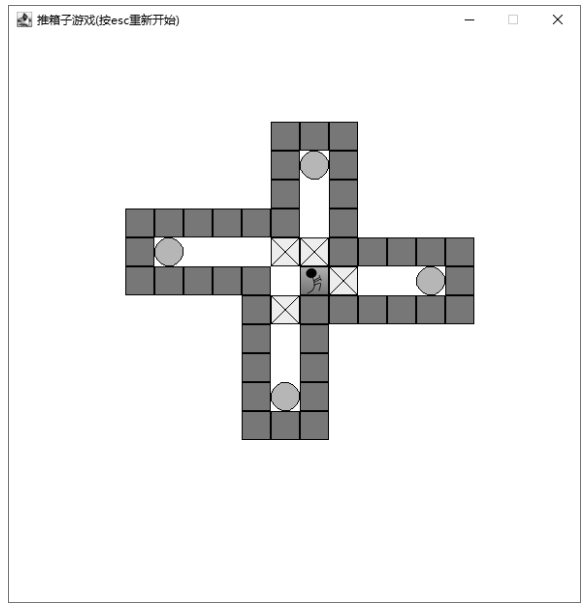


图 1.8　游戏面板效果

游戏面板的设计过程如下：

（1）创建一个名为 GamePanel 的 Java 类，让它继承自 JPanel 面板类，并实现 KeyListener 键盘事件监听接口，代码如下：

```
public class GamePanel extends JPanel implements KeyListener {
}
```

（2）在类中创建属性，分别用于记录主窗体对象、关卡编号、关卡中的所有元素、关卡的主图片、玩家对象和箱子列表等，代码如下：

```
private MainFrame frame;                                    // 主窗体对象
private int level;                                          // 关卡编号
private RigidBody[][] data = new RigidBody[20][20];         // 关卡中的所有元素
private BufferedImage image;                                // 关卡的主图片
private Graphics2D g2;                                      // 主图片的绘图对象
private Player player;                                      // 玩家对象
private ArrayList<Box> boxes;                               // 箱子列表
```

（3）编写 GamePanel 类的构造方法，该方法接收两个参数：frame 是主窗体对象，level 是关卡编号。根据此关卡编号读取对应的关卡对象。如果关卡编号小于 1，则构造方法将读取自定义关卡对象。构造方法最后为主窗体添加键盘事件监听器。构造方法代码如下：

```
public GamePanel(MainFrame frame, int level) {
    this.frame = frame;
    this.level = level;
    Map map;                                                // 关卡对象
    if (level < 1) {                                        // 如果关卡数小于 1
        map = GameMapUtil.readCustomMap();                  // 则读取自定义关卡对象
        this.level = 0;                                     // 将关卡数设为 0，方便定义下一关
        if (map == null) {                                  // 如果没有自定义关卡文件
            this.level = 1;                                 // 则将关卡设为第一关
            map = GameMapUtil.readMap(1);                   // 开始第一关
        }
    } else {                                                // 如果关卡数大于或等于 1
        map = GameMapUtil.readMap(level);                   // 则读取对应关卡数的关卡对象
    }
    data = map.getMapData();                                // 获取用于保存关卡中所有元素的数组
    player = map.getPlayer();                               // 获取关卡中的玩家对象
    boxes = map.getBoxes();                                 // 获取关卡中的箱子列表
    // 主图片为 600×600 的彩图
    image = new BufferedImage(600, 600, BufferedImage.TYPE_INT_RGB);
    g2 = image.createGraphics();                            // 获取主图片的绘图对象
    this.frame.addKeyListener(this);                        // 为主窗体添加键盘事件监听器
    this.frame.setFocusable(true);                          // 主窗体获取焦点
    this.frame.setTitle("推箱子(按 esc 重新开始)");          // 修改主窗体标题
}
```

（4）游戏面板实现键盘事件中 keyPressed()方法，该方法会在用户按下键盘上的任意按键时被触发。在 keyPressed()方法中，使用 KeyEvent 键盘事件对象来获取被按下的按键的 keyCode 值，同时获取玩家角色当前所处的坐标索引。坐标索引表示玩家角色在二维数组中的索引位置。使用 switch 判断按键 keyCode 值：如果用户按下的是上下左右键，则调用本类的 moveThePlayer()移动角色方法，根据按键的不同，传入不同的移动目标位置索引；如果用户按下的是 Esc 键，则调用本类的 gotoAnotherLevel()跳关方法，重新开始当前关卡。

keyPressed()方法的代码如下：

```
public void keyPressed(KeyEvent e) {
    int key = e.getKeyCode();                               // 获取按下的按键值
    int x = player.x;                                       // 记录玩家的横坐标索引
    int y = player.y;                                       // 记录玩家的纵坐标索引
    switch (key) {                                          // 判断按键值
        case KeyEvent.VK_UP :                               // 如果用户按下的是上箭头按键
            moveThePlayer(x, y - 1, x, y - 2);
            break;
        case KeyEvent.VK_DOWN :                             // 如果用户按下的是下箭头按键
            moveThePlayer(x, y + 1, x, y + 2);
            break;
        case KeyEvent.VK_LEFT :                             // 如果用户按下的是左箭头按键
            moveThePlayer(x - 1, y, x - 2, y);
            break;
        case KeyEvent.VK_RIGHT :                            // 如果用户按下的是右箭头按键
            moveThePlayer(x + 1, y, x + 2, y);
```

```
            break;
        case KeyEvent.VK_ESCAPE :                    // 如果用户按下的是 Esc 键
            gotoAnotherLevel(level);                 // 重新开始当前关卡
            break;
    }
    repaint();                                       // 重绘面板
}
```

（5）moveThePlayer()方法用来移动玩家角色和箱子。该方法接收四个参数：前两个参数表示玩家移动一步后所到达的目标位置的横坐标索引和纵坐标索引，后两个参数表示箱子被玩家移动一步后所到达的目标位置的横坐标索引和纵坐标索引。前两个参数用于判断玩家的移动路线是否被挡住，后两个参数用于判断箱子的移动路线是否被挡住。这个判断逻辑如图 1.9 所示。

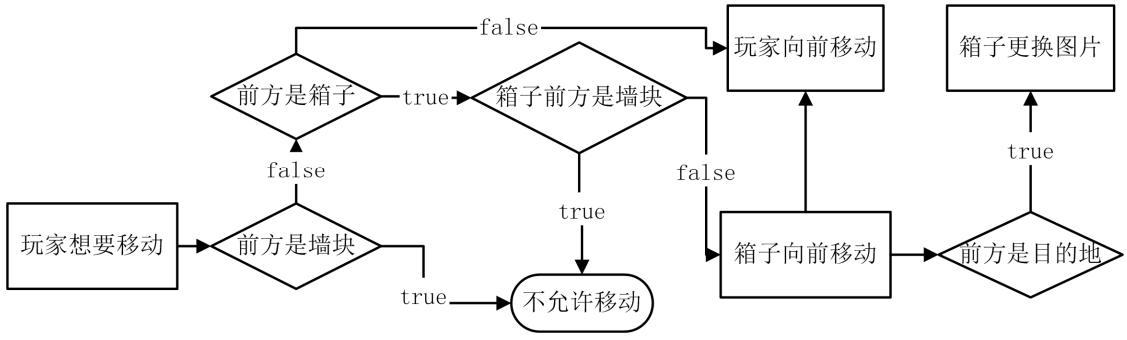


图 1.9　玩家移动时触发的逻辑

因为玩家可以在四个方向上移动，所以只需通过控制这四个参数就可以指定玩家下一步要移动到的位置。moveThePlayer()方法的代码如下：

```
private void moveThePlayer(int xNext1, int yNext1, int xNext2, int yNext2) {
    if (data[xNext1][yNext1] instanceof Wall) {          // 如果玩家前方是墙
        return;                                          // 什么都不做
    }
    Box box = new Box(xNext1, yNext1);                   // 在用户前方位置创建箱子对象
    if (boxes.contains(box)) {                           // 如果这个箱子在箱子列表中是存在的
        int index = boxes.indexOf(box);                  // 获取该箱子在列表中的索引
        box = boxes.get(index);                          // 取出列表中该箱子的对象
        if (data[xNext2][yNext2] instanceof Wall) {      // 如果箱子的前方是墙
            return;                                      // 什么都不做
        }
        if (boxes.contains(new Box(xNext2, yNext2))) {   // 如果箱子的前方还有其他箱子
            return;                                      // 什么都不做
        }
        if (data[xNext2][yNext2] instanceof Destination) {  // 如果箱子的前方是目的地
            box.arrive();                                // 箱子到达
        } else if (box.isArrived()) {                    // 如果箱子就在目的地上
            box.leave();                                 // 箱子离开
        }
        box.x = xNext2;                                  // 箱子被玩家推至了新位置上
        box.y = yNext2;
    }
    player.x = xNext1;                                   // 玩家移动到新位置
    player.y = yNext1;
}
```

（6）在确定关卡中所有元素的位置后，接下来就要绘制关卡，image 主图片绘图对象就是用来绘制关卡的。paintImage()方法用于绘制主图片，该方法首先会绘制一个白色的实心矩形来填充整张图片，这样就得到了白色的背景；然后遍历用于保存关卡中所有元素的数组，绘制数组中不是 null 的模型对象；最后绘

制箱子和玩家角色。paintImage()方法的代码如下：

```
private void paintImage() {
    g2.setColor(Color.WHITE);                                    // 使用白色
    g2.fillRect(0, 0, getWidth(), getHeight());                  // 绘制一个矩形来填充整张图片
    for (int i = 0, ilength = data.length; i < ilength; i++) {   // 遍历用于保存关卡中所有元素的数组
        for (int j = 0, jlength = data[i].length; j < jlength; j++) {
            RigidBody rb = data[i][j];                           // 获取模型对象
            if (rb != null) {
                Image image = rb.getImage();                     // 获取模型图片
                g2.drawImage(image, i * 30, j * 30, 30, 30, this);   // 在对应位置上绘制体图片
            }
        }
    }
    for (Box box : boxes) {                                      // 遍历箱子列表
        // 在对应位置上绘制箱子图片
        g2.drawImage(box.getImage(), box.x * 30, box.y * 30, 30, 30, this);
    }
    // 绘制玩家图片
    g2.drawImage(player.getImage(), player.x * 30, player.y * 30, 30, 30, this);
}
```

（7）绘制完 image 图片之后，需要将该图片显示到面板上，这就需要重写 paint()绘图方法。paint()方法除了要绘制图片，还要分析当前关卡的进程：如果所有箱子都到达了目的地，则进入下一关。paint()方法的代码如下：

```
public void paint(Graphics g) {
    paintImage();                                                // 绘制主图片
    g.drawImage(image, 0, 0, this);                              // 将主图片绘制到面板中

    boolean finish = true;                                       // 用于判断当前关卡是否结束的标志
    for (Box box : boxes) {                                      // 遍历箱子列表
        finish &= box.isArrived();                               // 将结束标志与箱子到达状态做与运算
    }
    if (finish && boxes.size() > 0) {                            // 如果所有箱子都到达目的地且箱子的个数大于 0
        gotoAnotherLevel(level + 1);                             // 进入下一关
    }
}
```

（8）gotoAnotherLevel()方法用于进入其他关卡，该方法接收一个参数，即关卡数。如果传入的关卡数大于最大关卡数，则认为玩家已经完成了所有关卡，直接进入开始面板。如果传入的参数小于最大关卡数，则主窗体加载下一关的游戏面板。跳转代码被封装在一个 Thread 线程对象中，这样可以在跳转前添加 0.5 秒的延时，以避免切换场景过快导致用户反应不过来。gotoAnotherLevel()方法的代码如下：

```
private void gotoAnotherLevel(int level) {
    frame.removeKeyListener(this);                               // 主窗体删除本类实现的键盘事件
    // 创建线程，创建 Runnable 接口的匿名类
    Thread t = new Thread(() -> {
        try {
            Thread.sleep(500);                                   // 0.5 秒之后
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (level > GameMapUtil.getLevelCount()) {               // 如果传入的关卡数大于最大关卡数
            frame.setPanel(new StarPanel(frame));                // 进入开始面板
            JOptionPane.showMessageDialog(frame, "通关啦！");    // 弹出通关对话框
        } else {
            frame.setPanel(new GamePanel(frame, level));         // 进入对应关卡
        }
    });
```

```
    t.start();                                                    // 启动线程
}
```

1.7.2 关卡编辑器

关卡编辑器是本游戏的一大亮点，它允许用户自己绘制自定义关卡。这不仅能让本游戏保持新鲜感，避免枯燥乏味，还能激发用户的创造力和主动性，进而增强游戏的趣味性。

关卡编辑器的所有功能都集中在关卡编辑器面板中，在该面板中，用户可以在空白区域通过单击来绘制关卡中的元素。如果按住鼠标左键并拖曳则会将鼠标经过的区域全部覆盖上关卡中的元素。另外，如果在已经绘制好的元素上右击，则会擦除该元素；如果按住鼠标右键并拖拽，则会擦除鼠标经过的所有区域内的关卡元素。

在面板下方的按钮区域，用户可以选择绘制不同类型的元素，如墙块、玩家、箱子和目的地。用户如果对绘制的关卡元素不满意，可以单击“清空”按钮来清空所有元素。单击“保存”按钮后，用户可以将绘制的关卡保存为自定义关卡文件。用户在开始面板上选择“开始游戏”功能后，即可体验自己设计的关卡。用户如果单击“返回”按钮，将直接返回开始界面，且不会进行任何保存操作。关卡编辑器面板的效果如图 1.10 所示。

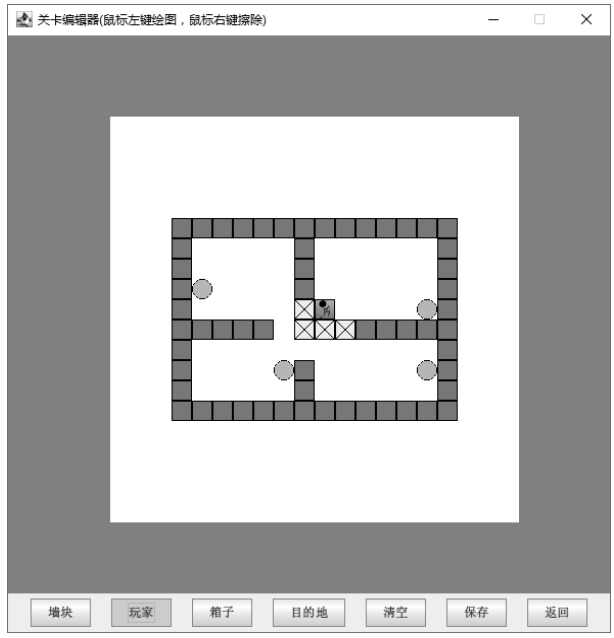


图 1.10 关卡编辑器面板效果

关卡编辑器的设计过程如下：

（1）创建一个名为 MapEditPanel 的 Java 类，并让它继承自 JPanel 面板类，代码如下：

```
public class MapEditPanel extends JPanel {
}
```

（2）在类中创建属性，分别记录主窗体对象、用于绘制关卡的主图片、各种按钮对象、用于保存关卡中所有元素的数组以及关卡图片在绘制面板上的坐标偏移量等。由于关卡图片并不是紧挨着面板左上角，而是在面板居中的位置，这会对鼠标坐标的计算产生影响，因此我们需要坐标偏移量参与鼠标的计算。

关卡编辑器中需要创建一个 DrawMapPanel 绘图面板类，该面板类用于提供鼠标绘图的空间。该类是一个成员内部类。

属性代码如下：

```
private MainFrame frame;                                  // 主窗体绘图面板
private BufferedImage image;                              // 绘制关卡的主图片
private Graphics2D g2;                                    // 关卡图片的绘图对象
private DrawMapPanel editPanel;                           // 绘制面板
private JToggleButton wall;                               // 墙块按钮，可显示被选中状态
private JToggleButton player;                             // 玩家按钮，可显示被选中状态
private JToggleButton box;                                // 箱子按钮，可显示被选中状态
private JToggleButton destination;                        // 目的地按钮，可显示被选中状态
private JButton save, clear, back;                        // 保存按钮，清空按钮，返回按钮
private RigidBody data[][];                               // 用于保存关卡中所有元素的数组
private int offsetX = 100, offsetY = 80;                  // 关卡图片在绘制面板上的横纵坐标偏移量
```

（3）在 MapEditPanel 类中，我们创建一个名为 DrawMapPanel 的成员内部类，这个内部类继承自 JPanel 面板类，同时实现 MouseListener 鼠标事件监听接口和 MouseMotionListener 鼠标移动事件监听接口。在 DrawMapPanel 类中，我们创建两个属性：paintFlag 用于记录用户是否正在利用鼠标进行绘图，而 clickButton 用于记录用户按下了哪个鼠标按键。DrawMapPanel 类的代码如下：

```
class DrawMapPanel extends JPanel implements MouseListener, MouseMotionListener {
    boolean paintFlag = false;                                  // 绘制墙体标志
    int clickButton;                                            // 鼠标单击的按键
}
```

（4）由于 DrawMapPanel 类实现了两个鼠标事件监听接口，因此需要重写接口的抽象方法。本游戏使用鼠标按键按下、释放和拖曳操作，因此需要实现 mousePressed()方法、mouseReleased()方法和 mouseDragged()方法。

当鼠标按键被按下时，会触发 mousePressed()方法，该方法首先将绘制 paintFlag 属性变为 true，表示正在绘图，然后记录用户按下了哪个按键，最后调用类中的 drawRigid()方法执行绘图。后续内容将详细介绍 drawRigid()方法。mousePressed()方法的代码如下：

```
public void mousePressed(MouseEvent e) {
    paintFlag = true;                                           // 绘制墙体标志为 true
    clickButton = e.getButton();                                // 记录当前被按下的鼠标按键
    drawRigid(e);                                               // 绘制墙块
}
```

当鼠标按键被释放时，会触发 mouseReleased()方法，该方法将 paintFlag 属性变为 false，就表示停止了绘图操作。mouseReleased()方法的代码如下：

```
public void mouseReleased(MouseEvent e) {
    paintFlag = false;                                          // 绘制墙体标志为 false
}
```

当鼠标被拖曳时，会触发 mouseDragged()方法，该方法直接触发 drawRigid()方法来绘制图片。mouseDragged()方法的代码如下：

```
public void mouseDragged(MouseEvent e) {
    drawRigid(e);                                               // 绘制墙块
}
```

（5）鼠标事件中调用的 drawRigid()方法也是 DrawMapPanel 类中的一个方法，该方法可以在空白图片上绘制关卡中的元素。

drawRigid()方法接收一个 MouseEvent 参数，该参数是一个鼠标事件对象，用于获取当前鼠标在面板中的坐标。获得鼠标坐标之后，该方法首先需要判断鼠标是否在空白区域内，如果鼠标在空白区域之外，就不会绘制任何元素。如果鼠标处于空白区域内并且 paintFlag 绘图状态是 true，那么该方法首先会获取鼠标坐标对应二维数组的索引，这个索引通过(鼠标坐标-偏移量)/模型宽度获得，例如第一个墙块的坐标为(0,0)，横向第二个墙块的坐标应该是(20,0)，当鼠标处在(0,0)～(20,20)的坐标范围时，该方法就认为鼠标选中的是第一个墙块。然后，该方法会判断用户按下了哪个鼠标键，左键则根据选中按钮在这个位置上创建对应的模型类，右键则将这个位置上的对象清空。最后，该方法会调用 repaint()方法重新把关卡图片绘制到面板上。drawRigid()方法的代码如下：

```
private void drawRigid(MouseEvent e) {
    RigidBody rb;                                               // 创建模型对象
    // 创建鼠标是否在关卡内标志，由于窗体压缩了绘图面板，因此有些数值需要微调
    boolean inMap = e.getX() > offsetX && e.getX() < 400 + offsetX
            && e.getY() > offsetY + 20 && e.getY() < 400 + offsetY + 20;
    if (inMap && paintFlag) {                                   // 如果鼠标在关卡内的标志和绘制墙体标志都为 true
```

```
            int x = (e.getX() - offsetX) / 20;                    // 计算鼠标所在区域的模型的横坐标索引
            int y = (e.getY() - offsetY - 20) / 20;               // 计算鼠标所在区域的模型的纵坐标索引
            if (wall.isSelected()) {                              // 如果墙块按钮被选中
                rb = new Wall();                                  // 模型按照墙进行实例化
            } else if (player.isSelected()) {                     // 如果玩家按钮被选中
                rb = new Player();                                // 模型按照玩家进行实例化
            } else if (box.isSelected()) {                        // 如果箱子钮被选中
                rb = new Box();                                   // 模型按照箱子进行实例化
            } else {                                              // 如果目的地按钮被选中
                rb = new Destination();                           // 模型按照目的地进行实例化
            }
            if (clickButton == MouseEvent.BUTTON1) {              // 如果按下的是鼠标左键
                if (data[x][y] == null) {                         // 选中的位置没有任何模型
                    data[x][y] = rb;                              // 填充选中按钮对应的模型
                }
            } else if (clickButton == MouseEvent.BUTTON3) {       // 如果按下的是鼠标右键
                data[x][y] = null;                                // 将选中的位置的对象清空
            }
            repaint();                                            // 重绘组件
        }
    }
```

（6）重写 DrawMapPanel 类的 paint()绘图方法，以便将关卡图片绘制到面板中，这样就可以显示用户绘制完毕的关卡。paint()方法的代码如下：

```
public void paint(Graphics g) {
    paintImage();                                                 // 绘制主图片
    g.setColor(Color.gray);                                       // 使用灰色
    g.fillRect(0, 0, getWidth(), getHeight());                    // 绘制一个矩形填充整个面板
    g.drawImage(image, offsetX, offsetY, this);                   // 将主图片绘制到面板中
}
```

（7）在 DrawMapPanel 类的 paint()绘图方法中调用一个 paintImage()方法，这个方法是写在外部类 MapEditPanel 中的。paintImage()方法负责解析用于保存关卡中所有元素的数组中的模型对象，将这些对象转化成图片并绘制相应的位置上。该方法中的 g2 是 MapEditPanel 类中的属性，它是关卡图片的绘图对象。首先，该方法在关卡中绘制一个白色的矩形以填充整个图片，从而为图片提供白色的背景。然后，该方法遍历关卡中的所有元素，如果取出的对象不是 null，则调用模型对象的 getImage()方法得到模型图片。最后，该方法将图片绘制到关卡图片中。不管模型图片多大，都按照宽和高都是 20 的正方形绘制。

paintImage()方法的代码如下：

```
void paintImage() {
    g2.setColor(Color.WHITE);                                     // 使用白色
    // 填充一个覆盖整个图片的白色矩形
    g2.fillRect(0, 0, image.getWidth(), image.getHeight());
    for (int i = 0, ilength = data.length; i < ilength; i++) {    // 遍历用于保存关卡中所有元素的数组
        for (int j = 0, jlength = data[i].length; j < jlength; j++) {
            RigidBody rb = data[i][j];                            // 取出模型对象
            if (rb != null) {                                     // 如果不是 null
                Image image = rb.getImage();                      // 获取此模型的图片
                g2.drawImage(image, i * 20, j * 20, 20, 20, this);// 会制在主图片中
            }
        }
    }
}
```

（8）编写完与绘图相关的组件之后，接下来就需要初始化关卡编辑器中的其他组件了。init()方法用于为所有组件执行初始化操作，包括实例化关卡图片对象，实例化绘图面板对象以及各种按钮对象。init()方法的代码如下：

```
private void init() {
    // 使用 400×400 的彩色图片，本游戏画面为 600×600，此处为本游戏画面的缩放版
    image = new BufferedImage(400, 400, BufferedImage.TYPE_INT_RGB);
    g2 = image.createGraphics();                                    // 获取图片的绘图对象
    editPanel = new DrawMapPanel();                                 // 实例化绘制面板
    data = new RigidBody[20][20];                                   // 关卡数组与关卡中行列数保持一致
    setLayout(new BorderLayout());                                  // 使用边界布局

    save = new JButton("保存");                                     // 实例化按钮对象
    clear = new JButton("清空");
    back = new JButton("返回");
    wall = new JToggleButton("墙块");
    player = new JToggleButton("玩家");
    box = new JToggleButton("箱子");
    destination = new JToggleButton("目的地");
    wall.setSelected(true);                                         // 默认选中墙按钮

    ButtonGroup group = new ButtonGroup();                          // 创建按钮组
    group.add(wall);                                                // 添加墙按钮到按钮组中
    group.add(player);                                              // 添加玩家按钮到按钮组中
    group.add(box);                                                 // 添加箱子按钮到按钮组中
    group.add(destination);                                         // 添加目的地按钮到按钮组中

    FlowLayout flow = new FlowLayout();                             // 创建流布局
    flow.setHgap(20);                                               // 设置水平间隔 20 像素
    JPanel buttonPanel = new JPanel(flow);                          // 创建按钮面板，采用流布局

    buttonPanel.add(wall);                                          // 向按钮面板中依次添加按钮
    buttonPanel.add(player);
    buttonPanel.add(box);
    buttonPanel.add(destination);
    buttonPanel.add(clear);
    buttonPanel.add(save);
    buttonPanel.add(back);

    add(editPanel, BorderLayout.CENTER);                            // 将绘图面板放在中央位置
    add(buttonPanel, BorderLayout.SOUTH);                           // 将按钮面板放在南部位置
}
```

（9）addListener()方法用于为关卡编辑器添加监听事件。由于鼠标事件都被写在内部类 DrawMapPanel 类中，因此为主窗体添加鼠标事件时，只需直接传入已经创建好的 DrawMapPanel 类对象即可。清空按钮通过 lambda 表达式添加动作事件，单击此按钮时，重新创建用于保存关卡中所有元素的数组，原有的数据就被清除了。保存按钮通过 lambda 表达式添加动作事件，单击此按钮时，会将用于保存关卡中所有元素的数组交给关卡工具类以生成自定义关卡文件，最后触发返回按钮的单击事件。返回按钮通过 lambda 表达式添加动作事件，单击此按钮时，首先会删除之前交给主窗体的所有鼠标事件，然后跳转到开始面板。addListener()方法的代码如下：

```
private void addListener() {
    frame.addMouseListener(editPanel);                              // 向主窗体中添加绘图面板中的鼠标事件
    frame.addMouseMotionListener(editPanel);                        // 向主窗体中添加绘图面板中的鼠标拖曳事件

    clear.addActionListener(e -> {                                  // 单击清空按钮时
        data = new RigidBody[20][20];                               // 用于保存关卡中所有元素的数组重新赋值
        repaint();                                                  // 重绘组件
    });

    save.addActionListener(e -> {                                   // 单击保存按钮时
        String name = GameMapUtil.CUSTOM_MAP_NAME;                  // 设置关卡名称为自定义关卡名称
```

```
            GameMapUtil.createMap(data, name);                    // 保存此关卡文件
            JOptionPane.showMessageDialog(MapEditPanel.this,
                    "自定义关卡创建成功！请直接开始游戏。");          // 弹出对话框并提示创建成功
            back.doClick();                                       // 触发返回按钮的单击事件
        });

        back.addActionListener(e -> {                             // 当单击返回按钮时
            frame.removeMouseListener(editPanel);                 // 从主窗体删除绘图面板中的鼠标事件
            frame.removeMouseMotionListener(editPanel);           // 从主窗体删除绘图面板中的鼠标拖曳事件
            frame.setPanel(new StarPanel(frame));                 // 主窗体载入开始面板
        });
    }
```

（10）编写 MapEditPanel 类的构造方法，在构造方法中修改主窗体标题，同时初始化所有组件，并添加监听。构造方法的代码如下：

```
public MapEditPanel(MainFrame frame) {
    this.frame = frame;
    this.frame.setTitle("关卡编辑器(鼠标左键绘图，鼠标右键擦除)");
    init();                                                       // 初始化组件
    addListener();                                                // 添加组件事件监听器
}
```

1.8 项 目 运 行

通过前述步骤，我们已成功设计并完成了“推箱子游戏”的开发。接下来，我们将运行本游戏，以检验我们的开发成果。如图 1.11 所示，在 Eclipse 中打开本游戏的项目结构，右击 default package 下的 Start.java 文件，然后选择 Run As→Java Application，即可运行本游戏。

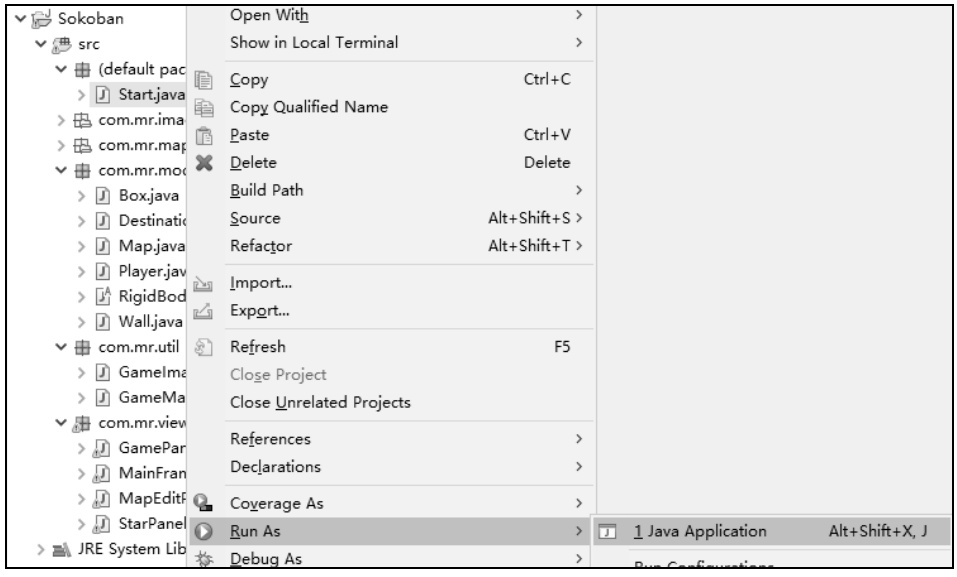


图 1.11 Eclipse 中的项目结构

本游戏成功运行后，会自动打开本游戏的主窗体，如图 1.12 所示。在主窗体中，用户既可以选择“开始游戏”功能，又可以选择“关卡编辑器”功能。用户在选择“开始游戏”功能后，即可开始闯关。用户在选择“关卡编辑器”功能后，即可设计自定义关卡。

图 1.12　已经载入开始面板的主窗体

本章编码实现的推箱子小游戏是平面类游戏，对画面效果要求不高，JDK 提供的技术完全能够满足开发需求。在开发过程中，主要的难点在于如何理解和使用 paint()方法。本游戏将关卡中的所有元素都绘制在 BufferedImage 图片中，当本游戏进入下一个关卡时，paint()方法会重新将这个 BufferedImage 图片绘制到游戏面板中，从而实现连续闯关的效果。

1.9　源 码 下 载

本章虽然详细地讲解了如何编码实现“推箱子游戏”的各个功能，但给出的代码都是代码片段，而非完整的源代码。为了方便读者学习，本书提供了用于下载完整源代码的二维码。

源码下载