第 5 章

初出茅庐：“俄罗斯方块”游戏项目

在第 4 章完成了一个用 JavaScript 开发并且运行在鸿蒙智能手机上的经典游戏 App——“数字华容道”，在本章将会讲解如何用 Java 从零开发一个运行在鸿蒙智能手机上的经典游戏 App——“俄罗斯方块”。

主页面的主体为两个按钮，按钮上显示的文本分别为“开始”和“关于”，主页面如图 5-1 所示。

单击“关于”按钮，便会跳转到副页面。副页面用于显示应用的相关信息，包括应用名称、作者和版本号，版本号下方有一个按钮，在按钮上显示的文本为“返回”，单击“返回”按钮便会跳转到主页面，副页面如图 5-2 所示。

图 5-1　主页面

图 5-2　副页面

单击“开始”按钮，便会跳转到游戏页面。游戏页面的主体为一个 15×10 的网格，网格下方有 5 个按钮，按钮名称分别为“←”“变”“→”“重新开始”“返回”，游戏页面如图 5-3 所示。

每次均在网格的顶部中间位置随机生成一个新的方块，每 750ms 方块会下落一格，直至下落到网格的底部或者其他方块的顶部为止，这时会重新随机生成一个新的方块。每次单击“←”按钮时，正在下落的方块会向左移动一格，如果正在下落的方块位于网格的左端或其左端存在其他方块，则不会再向左移动了；每次单击“→”按钮时，正在下落的方块会向右移动一格，如果正在下落的方块位于网格的右端或其右端存在其他方块，则不会再向右移动了。每次单击“变”按钮时，方块便会改变一次形态，但方块的颜色保持不变。

当单击“重新开始”按钮时，网格中的所有方块便会被清空，游戏将会重新开始。当单击“返回”按钮时，便会跳转到主页面。当网格中存在整行色彩方格时，该行方格将被消除，该行上方所有方格将会整体向下移动一格。当网格中无法生成新的方块时，将会在网格的上方显示“游戏结束”的文本，如图 5-4 所示。

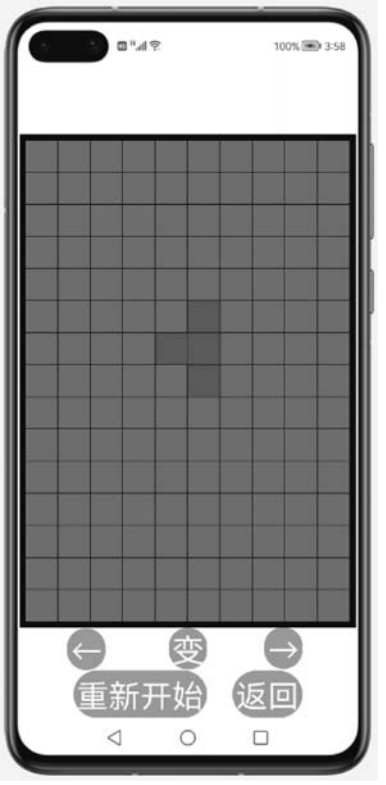

图 5-3　游戏页面

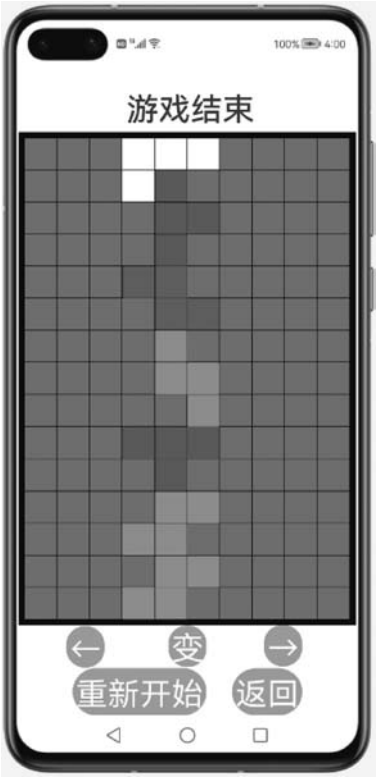

图 5-4　游戏结束页面

上述功能就是在学习完本章的内容后所完成的 App 的运行效果。

通过本次实战，可以掌握 Java 开发 HarmonyOS 智能手机 App 的众多核心技能，并且

通过实战项目的学习，不仅能降低学习成本，更能快速上手 HarmonyOS 应用开发。接下来正式开启俄罗斯方块项目的实战之旅！

5.1　创建 Hello World

在第 2 章创建了 JavaScript 版的 Hello World 项目进行开发，但是在本章使用 Java 来开发"俄罗斯方块"App，因此需要先创建 Java 版的 Hello World 项目，并在这个 Hello World 项目的基础上不断地进行修改和完善，最终开发出一个完整的经典游戏 App——"俄罗斯方块"。

首先打开集成开发环境 DevEco Studio，默认打开的文件是第 4 章的实战训练项目"数字华容道"MyGame，选中菜单栏中的 File，在展开的菜单中选择 New，在展开的子菜单中再选择 New Project 命令，如图 5-5 所示。

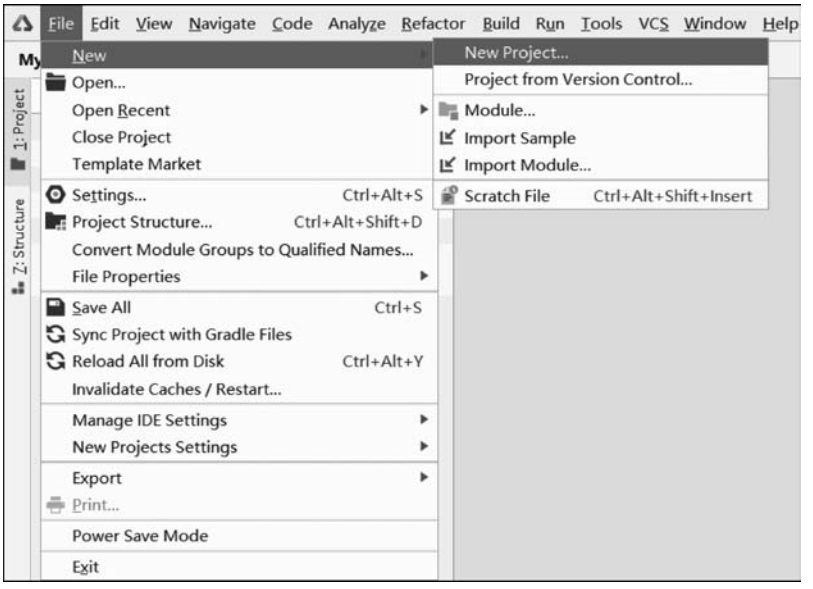

图 5-5　菜单栏中的菜单项 New Project

选中的 Template 是 Empty Ability，单击 Next 按钮，如图 5-6 所示。

在新打开的窗口中配置新建的项目，需要分别配置项目名、项目类型、包名、项目的保存位置、可兼容的 API 版本和设备类型。将 Project name 修改为 Game，DevEco Studio 会自动生成一个 Bundle name，其名称为 com. test. game。在 Project type 下选中 Application，在 Language 下勾选 Java，在 Compatible API version 下选择 SDK：API Version 6，在 Device type 下勾选 Phone，如图 5-7 所示。

单击 Finish 按钮后就创建了一个运行在智能手机上的 Java 版 Hello World 项目，如图 5-8 所示。

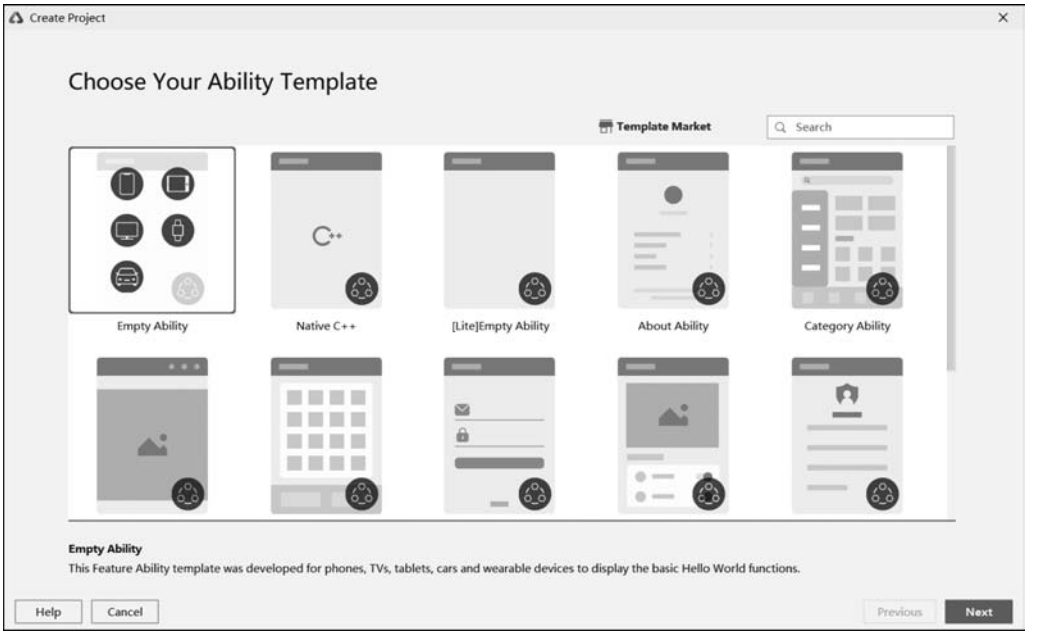

图 5-6　Empty Ability

图 5-7　配置新建的项目

图 5-8　新建的 Java 版 Hello World 项目

用本机模拟器来运行代码，在智能手机的主页面显示了文本"你好 世界"，运行效果如图 5-9 所示。

图 5-9　模拟器运行效果

5.2 在主页面中删除标题栏和修改其背景颜色

本节实现的运行效果：在主页面中删除 entry_MainAbility 标题栏，将背景颜色修改为淡黄色。

本节的实现思路：在布局文件 ability_main.xml 的定向布局 DirectionalLayout 中添加一个背景属性 background_element，赋值为对应的背景颜色，并且对配置文件 config.json 作隐藏标题栏操作的修改。

打开 entry/src/main/resources/base/layout/ability_main.xml 文件。

在定向布局 DirectionalLayout 中添加一个背景属性 background_element(背景图层)，将其属性的值设置为"#EFE5D4"，其为一个用 RGB 十六进制表示的颜色代码，以显示其背景颜色。其中，DirectionalLayout 表示将一组组件 Component 按照水平或者垂直方向排布。属性 height 和 width 分别表示组件的高度和组件的宽度，match_parent 表示组件大小将扩展为父组件允许的最大值，它将占据父组件方向上的剩余大小。属性 alignment(对齐方式)的值被设置为 center(居中对齐)，属性 orientation(子布局排列方向)的值被设置为 vertical(垂直方向布局)，代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<!-- 第 5 章 ability_main.xml -->
<!-- DirectionalLayout 表示定向布局 -->
< DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:height = "match_parent"
    ohos:width = "match_parent"
    ohos:alignment = "center"
    ohos:orientation = "vertical"
    ohos:background_element = "#EFE5D3">

    < Text
        ohos:id = "$ + id:text_helloworld"
        ohos:height = "match_content"
        ohos:width = "match_content"
        ohos:background_element = "$graphic:background_ability_main"
        ohos:layout_alignment = "horizontal_center"
        ohos:text = "$string:mainability_HelloWorld"
        ohos:text_size = "40vp"
        />

</DirectionalLayout >
```

打开 entry/src/main/config.json 文件。

在文件 config.json 的最下方的"launchType"："standard"的后面添加一个","，并添加如下代码：

```
第 5 章 config.json
…
        "launchType": "standard",
        "metaData": {
          "customizeData": [
            {
              "name": "hwc - theme",
              "value": "androidhwext:style/Theme.Emui.Light.NoTitleBar",
              "extra": ""
            }
          ]
        }
…
```

修改背景颜色后的主页面的运行效果，如图 5-10 所示。

图 5-10　修改背景颜色后的主页面

5.3 在主页面中添加两个按钮并响应其单击事件

本节实现的运行效果：在主页面显示两个按钮，在按钮上显示的文本分别为“开始”和“关于”。分别单击这两个按钮后，在 Log 窗口中打印一条文本“开始被单击了”和“关于被单击了”。

本节的实现思路：使用 Button 组件显示按钮，并设置好相关的属性值，通过 Button 组件的 id 属性指定唯一的标识。这样就能获取唯一的标识，以便设置单击事件，当单击按钮时就会触发按钮的单击事件，从而自动调用单击事件的自定义函数。

打开 ability_main.xml 文件。

删除原有的 Text 组件。添加一个 Button 组件，将属性 id 的值设置为 $ +id：button_game，以通过 MainAbilitySlice.java 文件根据唯一标识设置单击事件。将组件的 width（宽）和组件的 height（高）属性的值分别设置为 50vp 和 match_parent（组件大小将扩展为父组件允许的最大值）。将属性 text（文本）的值设置为“开始”，以显示按钮上的文本。将属性 text_size（文本大小）的值设置为 25vp，将属性 text_color（文本颜色）的值设置为 #FFFFFF（白色），将属性 text_alignment（文本的对齐方式）的值设置为 center（居中对齐），属性 background_element（背景图层）用于引用文件 background_ability_main.xml 的布局，代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<!-- 第 5 章 ability_main.xml -->
<!-- DirectionalLayout 表示定向布局 -->
< DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:height = "match_parent"
    ohos:width = "match_parent"
    ohos:alignment = "center"
    ohos:orientation = "vertical"
    ohos:background_element = "#EFE5D3">

    < Text
        ohos:id = "$ + id:text_helloworld"
        ohos:height = "match_content"
        ohos:width = "match_content"
        ohos:background_element = "$graphic:background_ability_main"
        ohos:layout_alignment = "horizontal_center"
        ohos:text = "$string:mainability_HelloWorld"
        ohos:text_size = "40vp"
        />
    <!-- 将文本设置为"开始"的按钮样式 -->
    < Button
```

```
        ohos:id = " $ + id:button_game"
        ohos:height = "50vp"
        ohos:width = "match_parent"
        ohos:text = "开始"
        ohos:text_size = "25vp"
        ohos:text_color = " # FFFFFF"
        ohos:text_alignment = "center"
        ohos:background_element = " $graphic:background_ability_main"/>

</DirectionalLayout >
```

再添加一个 Button 组件，将属性 id 的值设置为 $ +id:button_author。将属性 text(文本)的值设置为"关于"，将属性 top_margin(上外边距)的值设置为 25vp，其余属性值与上一个 Button 组件的属性值一致，代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<!-- 第 5 章 ability_main.xml -->
<!-- DirectionalLayout 表示定向布局 -->
< DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:height = "match_parent"
    ohos:width = "match_parent"
    ohos:alignment = "center"
    ohos:orientation = "vertical"
    ohos:background_element = " # EFE5D3">

    <!-- 将文本设置为"开始"的按钮样式 -->
    < Button
        ohos:id = " $ + id:button_game"
        ohos:height = "50vp"
        ohos:width = "match_parent"
        ohos:text = "开始"
        ohos:text_size = "25vp"
        ohos:text_color = " # FFFFFF"
        ohos:text_alignment = "center"
        ohos:background_element = " $graphic:background_ability_main"/>

    <!-- 将文本设置为"关于"的按钮样式 -->
    < Button
        ohos:id = " $ + id:button_author"
        ohos:height = "50vp"
        ohos:width = "match_parent"
        ohos:top_margin = "40vp"
        ohos:text = "关于"
        ohos:text_size = "25vp"
```

```
            ohos:text_color = " # FFFFFF"
            ohos:text_alignment = "center"
            ohos:background_element = " $graphic:background_ability_main"/>

    </DirectionalLayout >
```

打开 entry/src/main/resources/base/graphic/background_ability_main. xml 文件。

配置该文件以显示按钮组件的布局。将属性 color(颜色)的值设置为 # 78C6C5,将属性 radius(半径)的值并设置为 100,代码如下:

```
<?xml version = "1.0" encoding = "UTF - 8" ?>
<!-- 第 5 章 background_ability_main.xml -->
< shape xmlns:ohos = "http://schemas.huawei.com/res/ohos"
        ohos:shape = "rectangle">
    < corners
        ohos:radius = "100"/>
    < solid
        ohos:color = " # 78C6C5"/>
</shape >
```

主页面显示了两个按钮的运行效果,如图 5-11 所示。

图 5-11　主页面显示了两个按钮

接下来要实现的运行效果：分别单击两个按钮后各打印一条文本。

打开 MainAbilitySlice.java 文件。

初始化一个控制台输出窗口 HiLogLabel。分别定义两个按钮 button_game 和 button_author，通过唯一标识 ID 为刚才布局中的两个按钮赋值，并为这两个按钮添加一个单击事件，在单击事件的函数体内通过 HiLog.info()函数分别打印一条文本"开始被单击了"和"关于被单击了"。这样，当单击按钮时就会触发按钮的单击事件了，从而在 Log 窗口中相应地打印一条文本"开始被单击了"和"关于被单击了"，代码如下：

```
//第 5 章 MainAbilitySlice.java
package com.test.game.slice;

import com.test.game.ResourceTable;
import ohos.aafwk.ability.AbilitySlice;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Button;
import ohos.hiviewdfx.HiLog;
import ohos.hiviewdfx.HiLogLabel;

public class MainAbilitySlice extends AbilitySlice {
    //初始化控制台输出窗口
    private static final HiLogLabel Information = new HiLogLabel
            (HiLog.LOG_APP,0x00101,"控制台");

    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_main);

        //获取按钮组件对象
        Button button_game = (Button) findComponentById
                (ResourceTable.Id_button_game);
        //设置单击监听器
        button_game.setClickedListener(listener -> {
            //控制台输出语句
            HiLog.info(Information,"开始被单击了");
        });

        //获取按钮组件对象
        Button button_author = (Button) findComponentById
                (ResourceTable.Id_button_author);
        //设置单击监听器
        button_author.setClickedListener(listener -> {
            //控制台输出语句
            HiLog.info(Information,"关于被单击了");
```

```
            });
    }
        @Override
        public void onActive() {
            super.onActive();
        }

        @Override
        public void onForeground(Intent intent) {
            super.onForeground(intent);
        }
    }
```

找到 Log 窗口，在打开的窗口中，在第 3 个框中选择 com.test.game，在第 5 个框中输入“控制台”，如图 5-12 所示。

图 5-12　配置 Log 工具窗口

这样，单击主页面中的按钮后，在 Log 窗口中就会打印出对应的两条文本“开始被单击了”和“关于被单击了”，运行效果如图 5-13 所示。

图 5-13　打印文本的 Log 工具窗口

5.4　添加副页面并实现主页面向其跳转

本节实现的运行效果：单击主页面中的“关于”按钮，跳转到副页面。副页面的顶端会显示一条文本“副页面”。

本节的实现思路：新建一个布局，把此布局应用于新的 AbilitySlice 中，这样就可以实现副页面了。调用 present()语句实现页面间的跳转，在调用该语句时，通过指定 AbilitySlice 名称

到达指定跳转目标的页面。

右击项目的 layout 子目录，在弹出的菜单中选择 New，再在弹出的子菜单中选择 Layout Resource File，以新建一个 layout.xml 文件，如图 5-14 所示。

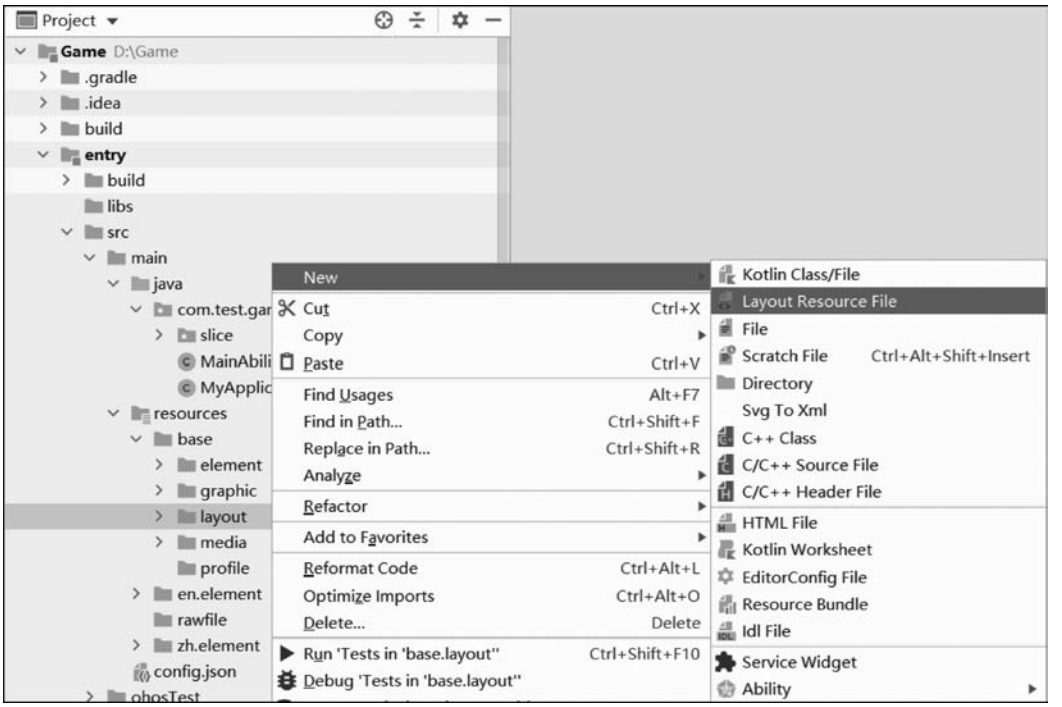

图 5-14　新建一个 layout.xml 文件

在打开的窗口中，将布局文件的名称设置为 ability_second，然后单击 OK 按钮，如图 5-15 所示。

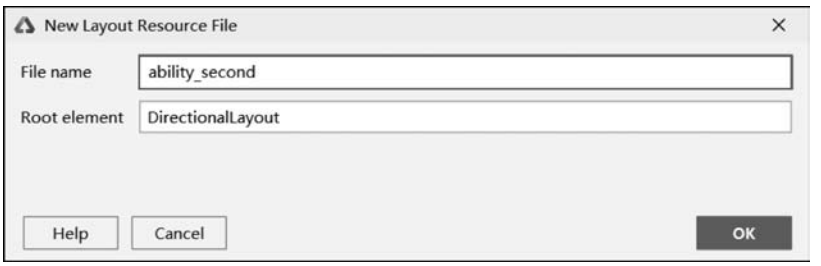

图 5-15　配置布局文件的名称

这样，在 layout 的目录下就自动创建了一个名为 ability_second.xml 的布局文件。

打开 ability_second.xml 文件。

添加一个 Text 组件，将组件的 width（宽）和组件的 height（高）属性的值都设置为

match_content(组件大小与它的内容占据的大小范围相适应),将属性 layout_alignment(对齐方式)的值设置为 horizontal_center(水平居中对齐),将属性 text(文本)的值设置为“副页面”,最后将属性 text_size(文本大小)的值设置为 40vp,代码如下:

```
<?xml version = "1.0" encoding = "utf - 8"?>
<!-- 第 5 章 ability_second.xml -->
<!-- DirectionalLayout 表示定向布局 -->
< DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:height = "match_parent"
    ohos:width = "match_parent"
    ohos:orientation = "vertical">

    <!-- 文本为"副页面"的文本组件 -->
    < Text
        ohos:height = "match_content"
        ohos:width = "match_content"
        ohos:layout_alignment = "horizontal_center"
        ohos:text = "副页面"
        ohos:text_size = "40vp"/>

</DirectionalLayout >
```

打开 Previewer,在副页面的顶端显示了一条文本“副页面”,运行效果如图 5-16 所示。

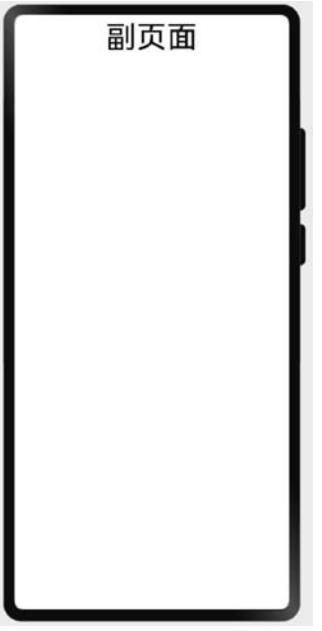

图 5-16 Previewer 运行效果

接下来要实现的运行效果：单击主页面中的"关于"按钮跳转到副页面。

右击项目的 slice 子目录，在弹出的菜单中选择 New，再在弹出的子菜单中选择 Java Class，以新建一个 Java 页面，如图 5-17 所示。

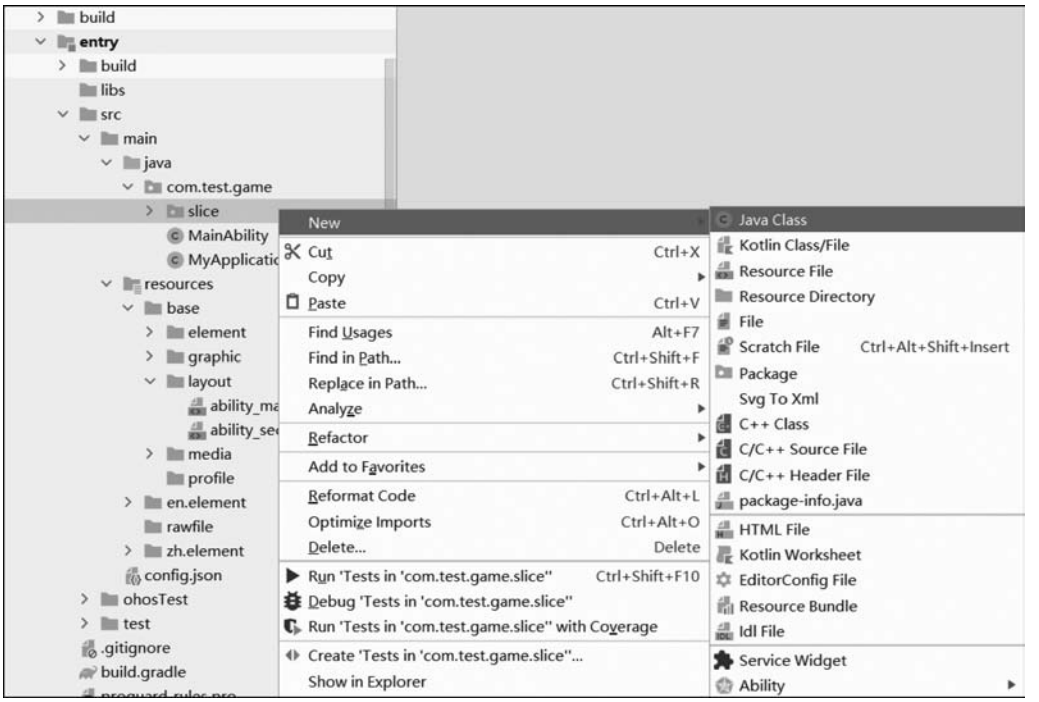

图 5-17　新建一个 Java 页面

将 Java 页面的名称设置为 SecondAbilitySlice，将其类型选择为 Class，然后按 Enter 键，如图 5-18 所示。

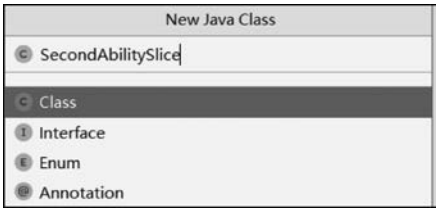

图 5-18　配置 Java 页面的名称

这样，在 slice 的目录下就自动创建了一个名为 SecondAbilitySlice.java 的文件。

打开 SecondAbilitySlice.java 文件。

将 SecondAbilitySlice 类继承自 AbilitySlice 类，添加一个名为 onStart()的函数，这是应用运行时会自动调用的函数，这在 5.6 节会详细讲述。并在 onStart()函数体内通过设置 UI 布局引用布局文件 ability_second.xml，代码如下：

```
//第 5 章 MainAbilitySlice.java
package com.test.game.slice;

import com.test.game.ResourceTable;
import ohos.aafwk.ability.AbilitySlice;
import ohos.aafwk.content.Intent;

public class SecondAbilitySlice extends AbilitySlice {
    @Override
    protected void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_second);
    }
}
```

打开 MainAbilitySlice.java 文件。

在“关于”按钮的单击事件中通过 present()语句跳转到 SecondAbilitySlice，当单击按钮时就会触发按钮的单击事件，从而跳转到副页面，代码如下：

```
//第 5 章 MainAbilitySlice.java
package com.test.game.slice;

import com.test.game.ResourceTable;
import ohos.aafwk.ability.AbilitySlice;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Button;
import ohos.hiviewdfx.HiLog;
import ohos.hiviewdfx.HiLogLabel;

public class MainAbilitySlice extends AbilitySlice {
    //初始化控制台输出窗口
    private static final HiLogLabel Information = new HiLogLabel
            (HiLog.LOG_APP,0x00101,"控制台");

    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_main);

        //获取按钮组件对象
        Button button_game = (Button) findComponentById
                (ResourceTable.Id_button_game);
        //设置单击监听器
        button_game.setClickedListener(listener -> {
            //控制台输出语句
            HiLog.info(Information,"开始被单击了");
        });
```

```
        //获取按钮组件对象
        Button button_author = (Button) findComponentById
                (ResourceTable.Id_button_author);
        //设置单击监听器
        button_author.setClickedListener(listener -> {
            //控制台输出语句
            HiLog.info(Information,"关于被单击了");
            //跳转到 SecondAbilitySlice()语句
            present(new SecondAbilitySlice(), intent);
        });
    }

    @Override
    public void onActive() {
        super.onActive();
    }

    @Override
    public void onForeground(Intent intent) {
        super.onForeground(intent);
    }
}
```

单击主页面中的“关于”按钮，即可跳转到副页面，运行效果如图 5-19 和图 5-20 所示。

图 5-19　主页面

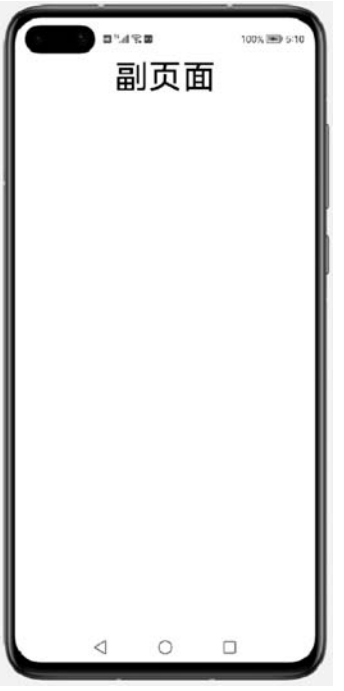

图 5-20　副页面

5.5 完善副页面的信息并实现其向主页面跳转

本节实现的运行效果：副页面显示应用的相关信息，包括应用名称、作者和版本号，版本号下方有一个按钮，按钮上显示的文本为“返回”，单击“返回”按钮便会跳转到主页面。

本节的实现思路：在布局文件中添加相应的组件，通过 Button 组件的 ID 属性指定唯一的标识。这样就能获取唯一的标识，以便设置单击事件，当单击按钮时就会触发按钮的单击事件，调用 present()语句实现页面间的跳转，从而跳转到主页面。

打开 ability_second.xml 文件。

在定向布局 DirectionalLayout 中添加一个背景属性 background_element，将其属性的值设置为＃EFE5D4，以显示其背景颜色。删除原有的 Text 组件。添加 Text 组件，将组件的 width(宽)和组件的 height(高)属性的值都设置为 match_content(组件大小与它的内容占据的大小范围相适应)，将属性 text(文本)的值设置为“程序：俄罗斯方块”，将属性 text_size(文本大小)的值设置为 25vp，将属性 text_color(文本颜色)的值设置为＃000000(黑色)。将属性 top_margin(上外边距)的值设置为 20vp，将属性 left_margin(左外边距)的值设置为 5vp。

再添加两个 Text 组件，将属性 text(文本)的值分别设置为“作者：张诏添”和“版本：v1.1.0”，其余属性值与上一个 Text 组件的属性值完全一致。

添加一个 Button 组件，将属性 id 的值设置为 $ + id：button_back，以通过 SecondAbilitySlice.java 文件根据唯一标识设置单击事件。将组件的 width(宽)和组件的 height(高)属性的值分别设置为 50vp 和 match_parent(组件大小将扩展为父组件允许的最大值)。将属性 top_margin(上外边距)的值设置为 30vp，将属性 text(文本)的值设置为返回。将属性 text_size(文本大小)的值设置为 25vp，将属性 text_color(文本颜色)的值设置为＃FFFFFF(白色)。将属性 text_alignment(对齐方式)的值设置为 center(居中对齐)，属性 background_element(背景图层)用于引用文件 background_ability_main.xml 的布局，代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<!-- 第 5 章 ability_second.xml -->
<!-- DirectionalLayout 表示定向布局 -->
< DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:height = "match_parent"
    ohos:width = "match_parent"
    ohos:orientation = "vertical"
    ohos:background_element = "＃EFE5D3">
    <!--文本为"副页面"的文本组件-->
    <Text
        ohos:height = "match_content"
        ohos:width = "match_content"
        ohos:layout_alignment = "horizontal_center"
```

```
            ohos:text = "副页面"
            ohos:text_size = "40vp"/>
        <!-- 文本为"程序:俄罗斯方块"的文本组件 -->
        < Text
            ohos:height = "match_content"
            ohos:width = "match_content"
            ohos:text = "程序:俄罗斯方块"
            ohos:text_size = "25vp"
            ohos:text_color = "#000000"
            ohos:top_margin = "20vp"
            ohos:left_margin = "5vp"/>

        <!-- 文本为"作者:张诏添"的文本组件 -->
        < Text
            ohos:height = "match_content"
            ohos:width = "match_content"
            ohos:text = "作者:张诏添"
            ohos:text_size = "25vp"
            ohos:text_color = "#000000"
            ohos:top_margin = "20vp"
            ohos:left_margin = "5vp"/>

        <!-- 文本为"版本:v1.1.0"的文本组件 -->
        < Text
            ohos:height = "match_content"
            ohos:width = "match_content"
            ohos:text = "版本:v1.1.0"
            ohos:text_size = "25vp"
            ohos:text_color = "#000000"
            ohos:top_margin = "20vp"
            ohos:left_margin = "5vp"/>

        <!-- 将文本设置为"返回"的按钮样式 -->
        < Button
            ohos:id = "$ + id:button_back"
            ohos:height = "50vp"
            ohos:width = "match_parent"
            ohos:top_margin = "30vp"
            ohos:text = "返回"
            ohos:text_size = "25vp"
            ohos:text_color = "#FFFFFF"
            ohos:text_alignment = "center"
            ohos:background_element = "$graphic:background_ability_main"/>

    </DirectionalLayout >
```

打开 SecondAbilitySlice.java 文件。

定义按钮 button_back，通过唯一标识 ID 赋值为刚才布局中的按钮。并为其添加一个单击事件，在单击事件的函数体内通过 present()语句跳转到 MainAbilitySlice，当单击按钮时就会触发按钮的单击事件，从而跳转到主页面，代码如下：

```
//第 5 章 MainAbilitySlice.java
package com.test.game.slice;

import com.test.game.ResourceTable;
import ohos.aafwk.ability.AbilitySlice;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Button;

public class SecondAbilitySlice extends AbilitySlice {
    @Override
    protected void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_second);

        //获取按钮组件对象
        Button button_back = (Button) findComponentById
                (ResourceTable.Id_button_back);
        //设置单击监听器
        button_back.setClickedListener(listener -> {
            //跳转到 MainAbilitySlice()语句
            present(new MainAbilitySlice(), intent);
        });
    }
}
```

单击副页面中的“返回”按钮，即可跳转到主页面，运行效果如图 5-21 和图 5-22 所示。

图 5-21　副页面

图 5-22　主页面

5.6　验证应用和每个页面的生命周期事件

本节实现的运行效果：主页面显示后，在 Log 窗口中依次打印文本"主页面 onStart()函数正在被调用"和"主页面 onActive()函数正在被调用"。

从主页面返回手机主页面后，在 Log 窗口中依次打印文本"主页面 onInactive()函数正在被调用"和"主页面 onBackground()函数正在被调用"。

从手机主界面返回主页面继续运行应用后，在 Log 窗口中依次打印文本"主页面 onForeground()函数正在被调用"和"主页面 onActive()函数正在被调用"。

退出应用后，在 Log 窗口中依次打印文本"主页面 onInactive()函数正在被调用""主页面 onBackground()函数正在被调用"和"主页面 onStop()函数正在被调用"。

本节的实现思路：对于鸿蒙智能手机的应用中的每个页面 Page Ability，在其应用开始运行到应用结束的整个过程，会在不同的阶段自动触发相应的生命周期事件。

Page Ability 的生命周期事件如图 5-23 所示。

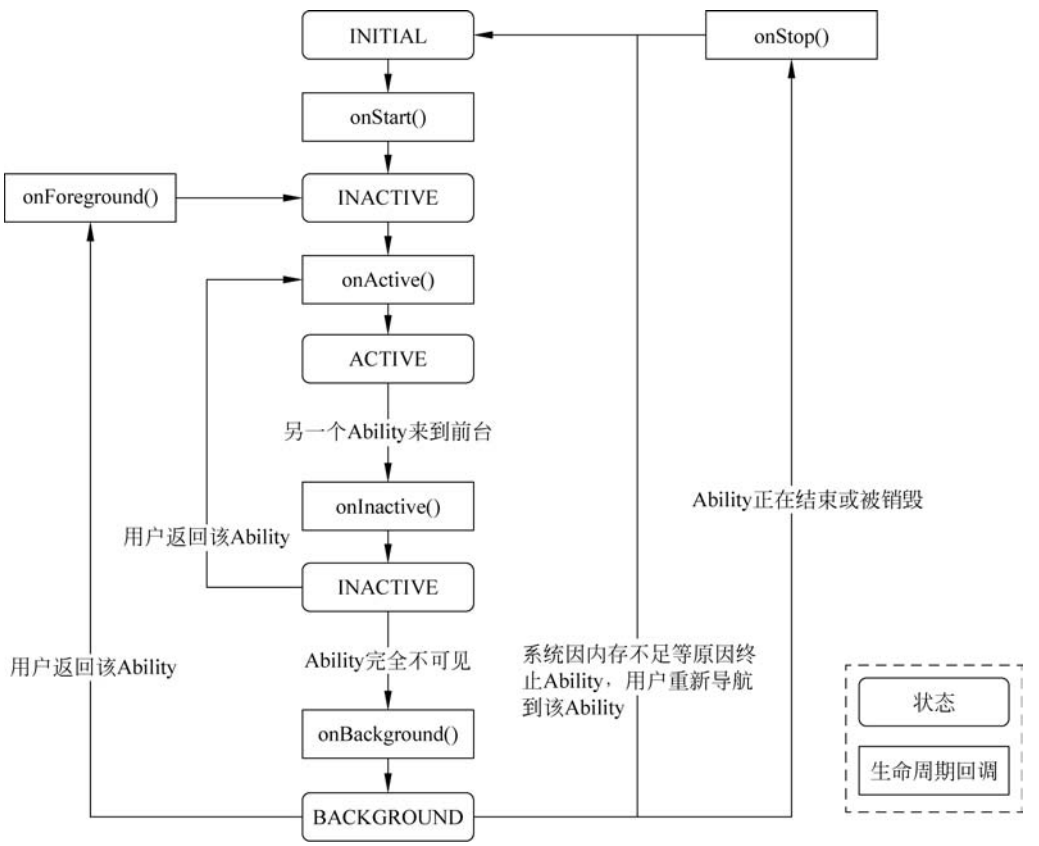

图 5-23　Page Ability 的生命周期事件

页面的生命周期事件主要有 6 个，分别是 onStart、onActive、onInactive、onBackground、onForeground 和 onStop。

（1）onStart 为当系统首次创建 Page 实例时触发。应用需重写该方法，并在此初始化，以便配置为展示 AbilitySlice。Page 在此后进入 INACTIVE 状态，用户不可交互。

（2）onActive 为当 Page 从 INACTIVE 状态切换到前台时触发。Page 在此之后进入 ACTIVE 状态，在该状态下，应用与用户处于可交互的状态。

（3）onInactive 为当 Page 即将进入不可交互状态时会被触发，Page 在此之后进入 INACTIVE 状态，应用与用户不可交互。

（4）onBackground 为当 Page 不再对用户可见时触发。Page 在此之后进入 BACKGROUND 状态。

（5）onForeground 为当 Page 从 BACKGROUND 状态重新回到前台时触发。Page 在此之后回到 INACTIVE 状态。

（6）onStop 为当系统将要销毁 Page 时触发。

打开 MainAbilitySlice.java 文件。

分别实现主页面的 6 个生命周期事件函数：onStart(Intent intent)、onActive()、onInactive()、onBackground()、onForeground(Intent intent)和 onStop()，在函数体中分别实现打印文本“主页面 onStart()函数正在被调用”“主页面 onActive()函数正在被调用”“主页面 onInactive()函数正在被调用”“主页面 onBackground()函数正在被调用”“主页面 onForeground()函数正在被调用”和“主页面 onStop()函数正在被调用”，代码如下：

```
//第 5 章 MainAbilitySlice.java
package com.test.game.slice;

import com.test.game.ResourceTable;
import ohos.aafwk.ability.AbilitySlice;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Button;
import ohos.hiviewdfx.HiLog;
import ohos.hiviewdfx.HiLogLabel;

public class MainAbilitySlice extends AbilitySlice {
    //初始化控制台输出窗口
    private static final HiLogLabel Information = new HiLogLabel
            (HiLog.LOG_APP,0x00101,"控制台");

    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_main);

        //控制台输出语句"主页面的 onStart()函数正在被调用"
```

```
        HiLog.info(Information,"主页面的 onStart()函数正在被调用");

        //获取按钮组件对象
        Button button_game = (Button) findComponentById
                (ResourceTable.Id_button_game);
        //设置单击监听器
        button_game.setClickedListener(listener -> {
            //控制台输出语句
            HiLog.info(Information,"开始被单击了");
        });

        //获取按钮组件对象
        Button button_author = (Button) findComponentById
                (ResourceTable.Id_button_author);
        //设置单击监听器 i
        button_author.setClickedListener(listener -> {
            //控制台输出语句
            HiLog.info(Information,"关于被单击了");
            //跳转到 SecondAbilitySlice()语句
            present(new SecondAbilitySlice(), intent);
        });
    }

    @Override
    public void onActive() {
        super.onActive();
        //控制台输出语句"主页面的 onActive()函数正在被调用"
        HiLog.info(Information,"主页面的 onActive()函数正在被调用");
    }

    @Override
    protected void onInactive() {
        super.onInactive();
        //控制台输出语句"主页面的 onInactive()函数正在被调用"
        HiLog.info(Information,"主页面的 onInactive()函数正在被调用");
    }

    @Override
    protected void onBackground() {
        super.onBackground();
        //控制台输出语句"主页面的 onBackground()函数正在被调用"
        HiLog.info(Information,"主页面的 onBackground()函数正在被调用");
    }

    @Override
    public void onForeground(Intent intent) {
```

```
            super.onForeground(intent);
            //控制台输出语句"主页面的 onForeground()函数正在被调用"
            HiLog.info(Information,"主页面的 onForeground()函数正在被调用");
        }

        @Override
        protected void onStop() {
            super.onStop();
            //控制台输出语句"主页面的 onStop()函数正在被调用"
            HiLog.info(Information,"主页面的 onStop()函数正在被调用");
        }
    }
```

找到 Log 窗口,在打开的窗口中,在第 3 个框中选择 com.test.game,在第 5 个框中输入“控制台”,使用模拟器运行。在 Log 窗口中首先打印文本“主页面 onStart()函数正在被调用”,然后打印文本“主页面 onActive()函数正在被调用”,运行效果如图 5-24 所示。

图 5-24　主页面显示后打印的文本

单击手机页面下方的圆形图标,即第 2 个按钮。在 Log 窗口中首先打印文本“主页面 onInactive()函数正在被调用”,然后打印文本“主页面 onBackground()函数正在被调用”,运行效果如图 5-25 所示。

图 5-25　从主页面返回手机主页面后打印的文本

单击手机页面下方的正方形图标,即第 3 个按钮,选择该应用页面继续运行。在 Log 窗口中首先打印文本“主页面 onForeground()函数正在被调用”,然后打印文本“主页面 onActive()函数正在被调用”,运行效果如图 5-26 所示。

图 5-26　从手机主页面返回主页面继续运行应用后打印的文本

单击手机页面下方的斜三角形图标，即第 1 个按钮。在 Log 窗口中首先打印文本"主页面 onInactive()函数正在被调用"，然后依次打印文本"主页面 onBackground()函数正在被调用"和"主页面 onStop()函数正在被调用"，运行效果如图 5-27 所示。

图 5-27　退出应用后打印的文本

5.7　在游戏页面绘制网格并实现从主页面向其跳转

本节实现的运行效果：单击主页面中的"开始"按钮，跳转到游戏页面。在游戏页面中显示一个 15×10 的网格。

本节的实现思路：在单击事件内调用 present()语句实现页面间的跳转，在调用该语句时通过指定 AbilitySlice 的名称达到指定跳转目标的页面。在游戏页面中直接用代码创建布局，先初始化定向布局，然后通过 Component.DrawTask 中的函数体 onDraw 绘制网格。

右击项目的 slice 子目录，在弹出的菜单中选择 New，再在弹出的子菜单中选择 Java Class，以新建一个 Java 页面，如图 5-28 所示。

将 Java 页面的名称设置为 ThirdAbilitySlice，将其类型选择为 Class，然后按 Enter 键，如图 5-29 所示。

这样，在 slice 的目录下就自动创建了一个名为 ThirdAbilitySlice.java 的文件。

打开 MainAbilitySlice.java 文件。

在"开始"按钮的单击事件上通过 present()语句跳转到 ThirdAbilitySlice，当单击按钮时就会触发按钮的单击事件，从而跳转到副页面，代码如下：

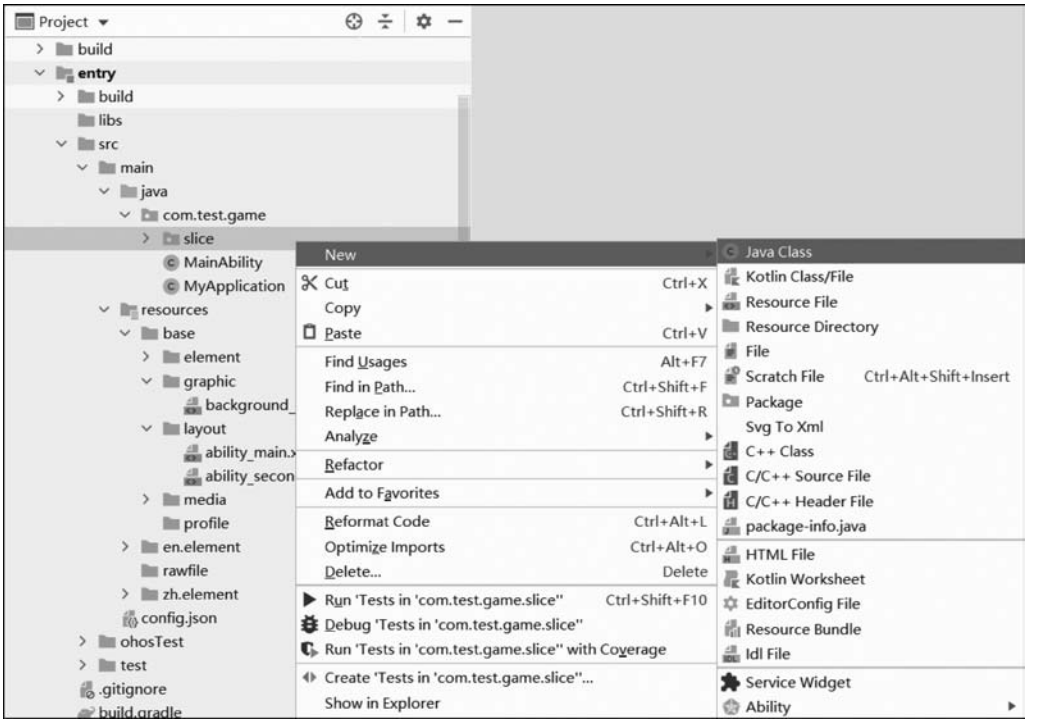

图 5-28　新建一个 Java 页面

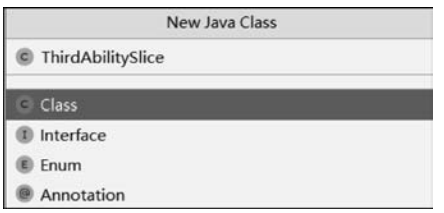

图 5-29　配置 Java 页面的名称

```
//第 5 章 MainAbilitySlice.java
package com.test.game.slice;

import com.test.game.ResourceTable;
import ohos.aafwk.ability.AbilitySlice;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Button;
import ohos.hiviewdfx.HiLog;
import ohos.hiviewdfx.HiLogLabel;

public class MainAbilitySlice extends AbilitySlice {
```

```
//初始化控制台输出窗口
private static final HiLogLabel Information = new HiLogLabel
        (HiLog.LOG_APP,0x00101,"控制台");

@Override
public void onStart(Intent intent) {
    super.onStart(intent);
    super.setUIContent(ResourceTable.Layout_ability_main);

    //控制台输出语句"主页面的 onStart()函数正在被调用"
    HiLog.info(Information,"主页面的 onStart()函数正在被调用");

    //获取按钮组件对象
    Button button_game = (Button) findComponentById
            (ResourceTable.Id_button_game);
    //设置单击监听器
    button_game.setClickedListener(listener -> {
        //控制台输出语句
        HiLog.info(Information,"开始被单击了");
        //跳转到 ThirdAbilitySlice()语句
        present(new ThirdAbilitySlice(), intent);
    });

    //获取按钮组件对象
    Button button_author = (Button) findComponentById
            (ResourceTable.Id_button_author);
    //设置单击监听器 i
    button_author.setClickedListener(listener -> {
        //控制台输出语句
        HiLog.info(Information,"关于被单击了");
        //跳转到 SecondAbilitySlice()语句
        present(new SecondAbilitySlice(), intent);
    });
}

@Override
public void onActive() {
    super.onActive();
    //控制台输出语句"主页面的 onActive()函数正在被调用"
    HiLog.info(Information,"主页面的 onActive()函数正在被调用");
}

@Override
protected void onInactive() {
    super.onInactive();
    //控制台输出语句"主页面的 onInactive()函数正在被调用"
```

```
        HiLog.info(Information,"主页面的 onInactive()函数正在被调用");
    }

    @Override
    protected void onBackground() {
        super.onBackground();
        //控制台输出语句"主页面的 onBackground()函数正在被调用"
        HiLog.info(Information,"主页面的 onBackground()函数正在被调用");
    }

    @Override
    public void onForeground(Intent intent) {
        super.onForeground(intent);
        //控制台输出语句"主页面的 onForeground()函数正在被调用"
        HiLog.info(Information,"主页面的 onForeground()函数正在被调用");
    }

    @Override
    protected void onStop() {
        super.onStop();
        //控制台输出语句"主页面的 onStop()函数正在被调用"
        HiLog.info(Information,"主页面的 onStop()函数正在被调用");
    }
}
```

打开 ThirdAbilitySlice.java 文件。

将 ThirdAbilitySlice 类继承自 AbilitySlice 类，将网格中方格的边长 length 设置为 100，将网格中方格的间距 interval 设置为 2，将网格中竖列方格的数量设置为 15，将网格中横列方格的数量设置为 10，将网格的左端距手机边界的距离设置为 30，将网格的顶端距手机边界的距离设置为 250，将网格的外围距离设置为 20，因为上述这些数值都是恒定不变的，所以都设置为常量。再定义一个定向布局 layout，用于创建游戏页面的布局，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;

public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;                    //自定义定向布局
    private static final int length = 100;               //网格中方格的边长
    private static final int interval = 2;               //网格中方格的间距
    private static final int height = 15;                //网格中竖列方格的数量
```

```
    private static final int width = 10;            //网格中横列方格的数量
    private static final int left = 30;             //网格的左端距手机边界的距离
    private static final int top = 250;             //网格的顶端距手机边界的距离
    private static final int margin = 20;           //网格的外围距离
}
```

添加一个名为 initialize()的函数，对定向布局 layout 初始化。添加一个名为 drawGrids()的函数，将布局 layout 的宽和高设置为占满整个界面 MATCH_PARENT。添加一个自定义绘制任务 task，声明画笔 paint 并将颜色设置为 Color. BLACK（黑色），利用语句 RectFloat()绘制背景大矩形，RectFloat()语句含有 4 个参数，第 1 个参数用于指定矩形左上角的横坐标，第 2 个参数用于指定矩形左上角的纵坐标，第 3 个参数用于指定矩形右下角的横坐标，第 4 个参数用于指定矩形右下角的纵坐标。最后将画笔设置为灰色 GRAY，用于绘制小矩形，将绘制任务添加到布局中。

在 initialize()函数体内调用函数 drawGrids()，添加一个生命周期事件 onStart()，调用函数 initialize()，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;

public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;           //自定义定向布局
    private static final int length = 100;      //网格中方格的边长
    private static final int interval = 2;      //网格中方格的间距
    private static final int height = 15;       //网格中竖列方格的数量
    private static final int width = 10;        //网格中横列方格的数量
    private static final int left = 30;         //网格的左端距手机边界的距离
    private static final int top = 250;         //网格的顶端距手机边界的距离
    private static final int margin = 20;       //网格的外围距离

    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);

        initialize();
    }
```

```
    //初始化数据的函数
    public void initialize() {
        layout = new DirectionalLayout(this); //对定向布局 layout 初始化

        drawGrids();
    }

    //绘制网格的函数
    public void drawGrids() {
        //将定向布局 layout 的宽和高设置为占满整个界面
        layout.setLayoutConfig((new ComponentContainer.LayoutConfig
                (ComponentContainer.LayoutConfig.MATCH_PARENT,
                ComponentContainer.LayoutConfig.MATCH_PARENT)));

        //绘制任务
        Component.DrawTask task = new Component.DrawTask() {
            @Override
            public void onDraw(Component component, Canvas canvas) {
                Paint paint = new Paint();          //初始化画笔

                paint.setColor(Color.BLACK);        //将画笔的颜色设置为黑色
                //绘制矩形
                RectFloat rect = new RectFloat(left - margin, top - margin,
                        length * width + interval * (width - 1) + left + margin,
                        length * height + interval * (height - 1) + top + margin);
                canvas.drawRect(rect, paint);

                for (int row = 0; row < height; row ++ ) {
                    for (int column = 0; column < width; column ++ ) {
                        paint.setColor(Color.GRAY);     //将画笔的颜色设置为灰色
                        RectFloat rectFloat = new RectFloat
                                (left + column * (length + interval),
                                top + row * (length + interval),
                                left + length + column * (length + interval),
                                top + length + row * (length + interval));
                        canvas.drawRect(rectFloat, paint);
                    }
                }
            }
        };

        layout.addDrawTask(task);
        setUIContent(layout);
    }
}
```

单击主页面中的"开始"按钮，即可跳转到游戏页面，运行效果如图 5-30 和图 5-31 所示。

图 5-30　主页面

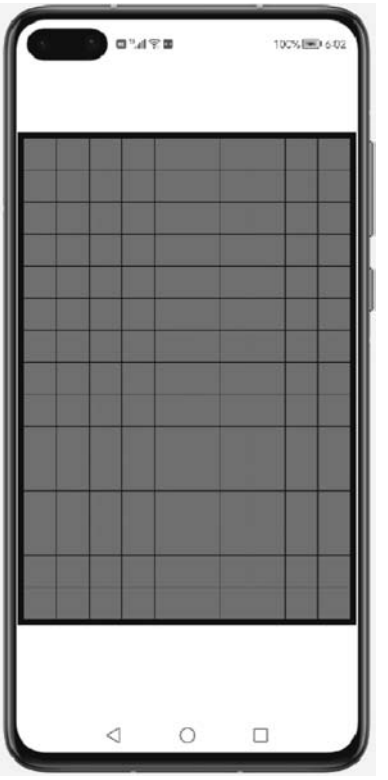

图 5-31　游戏页面

5.8　在游戏页面网格中随机生成方块

本节实现的运行效果：在游戏页面网格的顶部中间位置随机生成一个新的方块，每次运行时生成的方块都不一样。

本节的实现思路：用不同的数字表示方块的颜色，每种方块采用一个单独的二维数组存储方块所占网格的位置所对应的数组下标，当每次随机生成方块时均将该单独的二维数组加到网格中对应的位置，以此实现绘制不同的方块。

打开 ThirdAbilitySlice.java 文件。

定义网格的二维数组 grids，定义当前方块形态的二维数组 NowGrids，定义当前方块的总行数 row_number，定义当前方块的总列数 column_number，因为方块的行数和列数都只能为 1、2、3 或 4，所以 row_number 和 column_number 的取值只能为 1、2、3 或 4 中的任意一个数值。定义方块的第 1 个方格所在二维数组的列数 column_start，定义方块的颜色

GridsColor，其中，用 0 表示灰色，1 代表红色，2 代表绿色，3 代表蓝绿色，4 代表品红色，5 代表蓝色，6 代表白色，7 代表黄色。因为上述这些数值都是根据当前方块的信息而发生变化的，所以都定义为变量。

分别用一个单独的二维数组存储 19 种方块所占网格的位置所对应的数组下标，这其中一共包括 7 种颜色的方块。例如 GreenGrids1 = {{0, 5}, {0, 4}, {1, 4}, {1, 3}}所表示的方块如图 5-32 所示，RedGrids1 = {{0, 3}, {0, 4}, {1, 4}, {1, 5}}所表示的方块如图 5-33 所示。

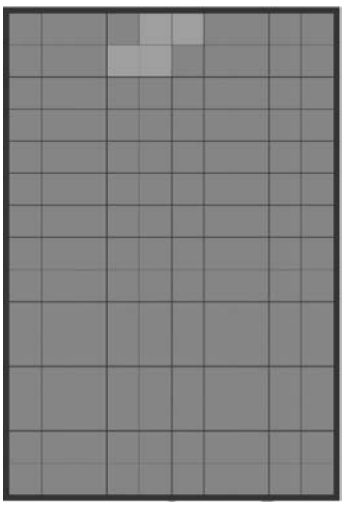

图 5-32 GreenGrids1

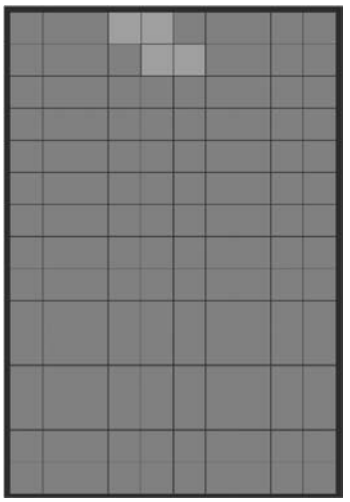

图 5-33 RedGrids1

将方块的方格数量 grids_number 定义为 4。因为上述这些数值都表示方块的信息恒定不变，所以都被定义为常量，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;

public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;           //自定义定向布局
    private static final int length = 100;      //网格中方格的边长
```

```
    private static final int interval = 2;              //网格中方格的间距
    private static final int height = 15;               //网格中竖列方格的数量
    private static final int width = 10;                //网格中横列方格的数量
    private static final int left = 30;                 //网格的左端距手机边界的距离
    private static final int top = 250;                 //网格的顶端距手机边界的距离
    private static final int margin = 20;               //网格的外围距离
    private int[][] grids;                              //15×10 网格的二维数组
    private int[][] NowGrids;                           //当前方块形态的二维数组
    private int row_number;                             //当前方块的总行数
    private int column_number;                          //当前方块的总列数
    private int column_start;                           //当前方块所在 grids 的列数
    //当前方块的颜色,0 表示灰色,1 代表红色,2 代表绿色,3 代表蓝绿色
    //4 代表品红色,5 代表蓝色,6 代表白色,7 代表黄色
    private int GridsColor;
    //19 种方块所占网格的位置所对应的数值
    private static final int[][] RedGrids1 = {{0, 3}, {0, 4}, {1, 4}, {1, 5}};
    private static final int[][] RedGrids2 = {{0, 5}, {1, 5}, {1, 4}, {2, 4}};
    private static final int[][] RedGrids1 = {{0, 3}, {0, 4}, {1, 4}, {1, 5}};
    private static final int[][] RedGrids2 = {{0, 5}, {1, 5}, {1, 4}, {2, 4}};
    private static final int[][] RedGrids1 = {{0, 3},{0, 4},{1, 4},{1, 5}};
    private static final int[][] RedGrids2 = {{0, 5},{1, 5},{1, 4},{2, 4}};
    private static final int[][] GreenGrids1 = {{0, 5},{0, 4},{1, 4},{1, 3}};
    private static final int[][] GreenGrids2 = {{0, 4},{1, 4},{1, 5},{2, 5}};
    private static final int[][] CyanGrids1 = {{0, 4},{1, 4},{2, 4},{3, 4}};
    private static final int[][] CyanGrids2 = {{0, 3},{0, 4},{0, 5},{0, 6}};
    private static final int[][] MagentaGrids1 = {{0,4},{1, 3},{1, 4},{1, 5}};
    private static final int[][] MagentaGrids2 = {{0,4},{1, 4},{1, 5},{2, 4}};
    private static final int[][] MagentaGrids3 = {{0,3},{0, 4},{0, 5},{1, 4}};
    private static final int[][] MagentaGrids4 = {{0,5},{1, 5},{1, 4},{2, 5}};
    private static final int[][] BlueGrids1 = {{0, 3},{1, 3},{1, 4},{1, 5}};
    private static final int[][] BlueGrids2 = {{0, 5},{0, 4},{1, 4},{2, 4}};
    private static final int[][] BlueGrids3 = {{0, 3},{0, 4},{0, 5},{1, 5}};
    private static final int[][] BlueGrids4 = {{0, 5},{1, 5},{2, 5},{2, 4}};
    private static final int[][] WhiteGrids1 = {{0, 5},{1, 5},{1, 4},{1, 3}};
    private static final int[][] WhiteGrids2 = {{0, 4},{1, 4},{2, 4},{2, 5}};
    private static final int[][] WhiteGrids3 = {{0, 5},{0, 4},{0, 3},{1, 3}};
    private static final int[][] WhiteGrids4 = {{0, 4},{0, 5},{1, 5},{2, 5}};
    private static final int[][] YellowGrids = {{0, 4},{0, 5},{1, 5},{1, 4}};
    private static final int grids_number = 4;          //方块的方格数量

    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);

        initialize();
    }
    ...
}
```

添加一个名为 createRedGrids1()的函数，对红色方块的形态 1 赋予 NowGrids 为 RedGrids1，row_number 为 2，column_number 为 3，GridsColor 为 1，column_start 为 2；添加一个名为 createRedGrids2()的函数，对红色方块的形态 2 赋予 NowGrids 为 RedGrids2，row_number 为 3，column_number 为 2，GridsColor 为 1，column_start 为 4；同理，分别添加名为 createGreenGrids1()、createGreenGrids2()、createCyanGrids1()、createCyanGrids2()、createMagentaGrids1()、createMagentaGrids2()、createMagentaGrids3()、createMagentaGrids4()、createBlueGrids1()、createBlueGrids2()、createBlueGrids3()、createBlueGrids4()、createWhiteGrids1()、createWhiteGrids2()、createWhiteGrids3()、createWhiteGrids4()和 createYellowGrids()的函数，对对应颜色方块的不同形态赋予 NowGrids、row_number、column_number、Grids、column_start 的值，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;

public class ThirdAbilitySlice extends AbilitySlice {
    ...
    //绘制网格的函数
    public void drawGrids() {
        //将定向布局 layout 的宽和高设置为占满整个界面
        layout.setLayoutConfig((new ComponentContainer.LayoutConfig(
                ComponentContainer.LayoutConfig.MATCH_PARENT,
                ComponentContainer.LayoutConfig.MATCH_PARENT)));

        //绘制任务
        Component.DrawTask task = new Component.DrawTask() {
            @Override
            public void onDraw(Component component, Canvas canvas) {
                Paint paint = new Paint();          //初始化画笔

                paint.setColor(Color.BLACK);        //将画笔的颜色设置为黑色
                //绘制矩形
                RectFloat rect = new RectFloat(left - margin, top - margin,
                        length * width + interval * (width - 1) + left + margin
```

```
                    length * height + interval * (height - 1) + top + margin);
            canvas.drawRect(rect, paint);

            for (int row = 0; row < height; row ++ ) {
                for (int column = 0; column < width; column ++ ) {
                    paint.setColor(Color.GRAY); //将画笔的颜色设置为灰色
                    RectFloat rectFloat = new RectFloat(
                            left + column * (length + interval),
                            top + row * (length + interval),
                            left + length + column * (length + interval),
                            top + length + row * (length + interval));
                    canvas.drawRect(rectFloat, paint);
                }
            }
        }
    };

    layout.addDrawTask(task);
    setUIContent(layout);
}

//对对应颜色方块的不同形态赋予 NowGrids、row_number、column_number
//GridsColor、column_start 的值
public void createRedGrids1() {
    NowGrids = RedGrids1;
    row_number = 2;
    column_number = 3;
    GridsColor = 1;
    column_start = 3;
}

public void createRedGrids2() {
    NowGrids = RedGrids2;
    row_number = 3;
    column_number = 2;
    GridsColor = 1;
    column_start = 4;
}

public void createGreenGrids1() {
    NowGrids = GreenGrids1;
    row_number = 2;
    column_number = 3;
    GridsColor = 2;
    column_start = 3;
}
```

```
    public void createGreenGrids2() {
        NowGrids = GreenGrids2;
        row_number = 3;
        column_number = 2;
        GridsColor = 2;
        column_start = 4;
    }

    public void createCyanGrids1() {
        NowGrids = CyanGrids1;
        row_number = 4;
        column_number = 1;
        GridsColor = 3;
        column_start = 4;
    }

    public void createCyanGrids2() {
        NowGrids = CyanGrids2;
        row_number = 1;
        column_number = 4;
        GridsColor = 3;
        column_start = 3;
    }

    public void createMagentaGrids1() {
        NowGrids = MagentaGrids1;
        row_number = 2;
        column_number = 3;
        GridsColor = 4;
        column_start = 3;
    }

    public void createMagentaGrids2() {
        NowGrids = MagentaGrids2;
        row_number = 3;
        column_number = 2;
        GridsColor = 4;
        column_start = 4;
    }

    public void createMagentaGrids3() {
        NowGrids = MagentaGrids3;
        row_number = 2;
        column_number = 3;
        GridsColor = 4;
        column_start = 3;
    }
```

```
    public void createMagentaGrids4() {
        NowGrids = MagentaGrids4;
        row_number = 3;
        column_number = 2;
        GridsColor = 4;
        column_start = 4;
    }

    public void createBlueGrids1() {
        NowGrids = BlueGrids1;
        row_number = 2;
        column_number = 3;
        GridsColor = 5;
        column_start = 3;
    }

    public void createBlueGrids2() {
        NowGrids = BlueGrids2;
        row_number = 3;
        column_number = 2;
        GridsColor = 5;
        column_start = 4;
    }

    public void createBlueGrids3() {
        NowGrids = BlueGrids3;
        row_number = 2;
        column_number = 3;
        GridsColor = 5;
        column_start = 3;
    }

    public void createBlueGrids4() {
        NowGrids = BlueGrids4;
        row_number = 3;
        column_number = 2;
        GridsColor = 5;
        column_start = 4;
    }

    public void createWhiteGrids1() {
        NowGrids = WhiteGrids1;
        row_number = 2;
        column_number = 3;
        GridsColor = 6;
```

```
            column_start = 3;
        }

        public void createWhiteGrids2() {
            NowGrids = WhiteGrids2;
            row_number = 3;
            column_number = 2;
            GridsColor = 6;
            column_start = 4;
        }

        public void createWhiteGrids3() {
            NowGrids = WhiteGrids3;
            row_number = 2;
            column_number = 3;
            GridsColor = 6;
            column_start = 3;
        }

        public void createWhiteGrids4() {
            NowGrids = WhiteGrids4;
            row_number = 3;
            column_number = 2;
            GridsColor = 6;
            column_start = 4;
        }

        public void createYellowGrids() {
            NowGrids = YellowGrids;
            row_number = 2;
            column_number = 2;
            GridsColor = 7;
            column_start = 4;
        }
    }
```

在 initialize()函数体内将 grids 初始化为 15×10 的二维数组,数组中的值全部为 0。在 drawGrids()函数体内的绘制任务 task 中绘制小方格时,先对 grids 中的数值进行颜色判断,如果颜色为 0,则绘制灰色方格;如果颜色为 1,则绘制红色方格;如果颜色为 2,则绘制绿色方格;如果颜色为 3,则绘制蓝绿色方格;如果颜色为 4,则绘制品红色方格;如果颜色为 5,则绘制蓝色方格;如果颜色为 6,则绘制白色方格;如果颜色为 7,则绘制黄色方格;代码如下:

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;

public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;           //自定义定向布局
    private static final int length = 100;      //网格中方格的边长
    private static final int interval = 2;      //网格中方格的间距
    private static final int height = 15;       //网格中竖列方格的数量
    private static final int width = 10;        //网格中横列方格的数量
    private static final int left = 30;         //网格的左端距手机边界的距离
    private static final int top = 250;         //网格的顶端距手机边界的距离
    private static final int margin = 20;       //网格的外围距离
    private int[][] grids;                      //15×10 网格的二维数组
    private int[][] NowGrids;                   //当前方块形态的二维数组
    private int row_number;                     //当前方块的总行数
    private int column_number;                  //当前方块的总列数
    private int column_start;                   //当前方块所在 grids 的列数
    //当前方块的颜色,0 表示灰色,1 代表红色,2 代表绿色,3 代表蓝绿色
    //4 代表品红色,5 代表蓝色,6 代表白色,7 代表黄色
    private int GridsColor;
    //19 种方块所占网格的位置所对应的数值
    private static final int[][] RedGrids1 = {{0, 3}, {0, 4}, {1, 4}, {1, 5}};
    private static final int[][] RedGrids2 = {{0, 5}, {1, 5}, {1, 4}, {2, 4}};
    private static final int[][] RedGrids1 = {{0, 3}, {0, 4}, {1, 4}, {1, 5}};
    private static final int[][] RedGrids2 = {{0, 5}, {1, 5}, {1, 4}, {2, 4}};
    private static final int[][] RedGrids1 = {{0, 3},{0, 4},{1, 4},{1, 5}};
    private static final int[][] RedGrids2 = {{0, 5},{1, 5},{1, 4},{2, 4}};
    private static final int[][] GreenGrids1 = {{0, 5},{0, 4},{1, 4},{1, 3}};
    private static final int[][] GreenGrids2 = {{0, 4},{1, 4},{1, 5},{2, 5}};
    private static final int[][] CyanGrids1 = {{0, 4},{1, 4},{2, 4},{3, 4}};
    private static final int[][] CyanGrids2 = {{0, 3},{0, 4},{0, 5},{0, 6}};
    private static final int[][] MagentaGrids1 = {{0,4},{1, 3},{1, 4},{1, 5}};
    private static final int[][] MagentaGrids2 = {{0,4},{1, 4},{1, 5},{2, 4}};
    private static final int[][] MagentaGrids3 = {{0,3},{0, 4},{0, 5},{1, 4}};
    private static final int[][] MagentaGrids4 = {{0,5},{1, 5},{1, 4},{2, 5}};
    private static final int[][] BlueGrids1 = {{0, 3},{1, 3},{1, 4},{1, 5}};
```

```
private static final int[][] BlueGrids2 = {{0, 5},{0, 4},{1, 4},{2, 4}};
private static final int[][] BlueGrids3 = {{0, 3},{0, 4},{0, 5},{1, 5}};
private static final int[][] BlueGrids4 = {{0, 5},{1, 5},{2, 5},{2, 4}};
private static final int[][] WhiteGrids1 = {{0, 5},{1, 5},{1, 4},{1, 3}};
private static final int[][] WhiteGrids2 = {{0, 4},{1, 4},{2, 4},{2, 5}};
private static final int[][] WhiteGrids3 = {{0, 5},{0, 4},{0, 3},{1, 3}};
private static final int[][] WhiteGrids4 = {{0, 4},{0, 5},{1, 5},{2, 5}};
private static final int[][] YellowGrids = {{0, 4},{0, 5},{1, 5},{1, 4}};
private static final int grids_number = 4; //方块的方格数量

@Override
public void onStart(Intent intent) {
    super.onStart(intent);

    initialize();
}

//初始化数据的函数
public void initialize() {
    layout = new DirectionalLayout(this); //对定向布局 layout 初始化
    //将二维数组 grids 初始化为 0
    grids = new int[height][width];
    for (int row = 0; row < height; row ++ )
        for (int column = 0; column < width; column ++ )
            grids[row][column] = 0;

    drawGrids();
}

 //绘制网格的函数
 public void drawGrids() {
    //将定向布局 layout 的宽和高设置为占满整个界面
    layout.setLayoutConfig((new ComponentContainer.LayoutConfig
            (ComponentContainer.LayoutConfig.MATCH_PARENT,
            ComponentContainer.LayoutConfig.MATCH_PARENT)));

    //绘制任务
    Component.DrawTask task = new Component.DrawTask() {
        @Override
        public void onDraw(Component component, Canvas canvas) {
            Paint paint = new Paint(); //初始化画笔

            paint.setColor(Color.BLACK); //将画笔的颜色设置为黑色
            //绘制矩形
            RectFloat rect = new RectFloat(left - margin, top - margin,
                    length * width + interval * (width - 1) + left + margin,
```

```
                    length * height + interval * (height - 1) + top + margin);
            canvas.drawRect(rect, paint);

            for (int row = 0; row < height; row ++ ) {
                for (int column = 0; column < width; column ++ ) {
                    paint.setColor(Color.GRAY);
                    //对数值进行判断,并将画笔设置为相应的颜色
                    if (grids[row][column] == 0)
                        paint.setColor(Color.GRAY);
                    else if (grids[row][column] == 1)
                        paint.setColor(Color.RED);
                    else if (grids[row][column] == 2)
                        paint.setColor(Color.GREEN);
                    else if (grids[row][column] == 3)
                        paint.setColor(Color.CYAN);
                    else if (grids[row][column] == 4)
                        paint.setColor(Color.MAGENTA);
                    else if (grids[row][column] == 5)
                        paint.setColor(Color.BLUE);
                    else if (grids[row][column] == 6)
                        paint.setColor(Color.WHITE);
                    else if (grids[row][column] == 7)
                        paint.setColor(Color.YELLOW);
                    RectFloat rectFloat = new RectFloat
                            (left + column * (length + interval),
                            top + row * (length + interval),
                            left + length + column * (length + interval),
                            top + length + row * (length + interval));
                    canvas.drawRect(rectFloat, paint);
                }
            }
        }
    };

    layout.addDrawTask(task);
    setUIContent(layout);
}

//对对应颜色方块的不同形态赋予 NowGrids、row_number、column_number
//GridsColor、column_start 的值
public void createRedGrids1() {
    NowGrids = RedGrids1;
    row_number = 2;
    column_number = 3;
    GridsColor = 1;
    column_start = 3;
}
...
}
```

添加一个名为 createGrids()的函数，在函数体内通过 random()方法生成一个 0～1 的随机数 random。根据随机数 random 的值，调用赋予不同颜色方块的不同形态的 NowGrids、row_number、column_number、Grids、column_start 的函数。再将 grids 对应位置的数值修改为方块的颜色数值，最后在 initialize()函数体内调用 createGrids()函数，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;

public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;                    //自定义定向布局
    private static final int length = 100;               //网格中方格的边长
    private static final int interval = 2;               //网格中方格的间距
    private static final int height = 15;                //网格中竖列方格的数量
    private static final int width = 10;                 //网格中横列方格的数量
    private static final int left = 30;                  //网格的左端距手机边界的距离
    private static final int top = 250;                  //网格的顶端距手机边界的距离
    private static final int margin = 20;                //网格的外围距离
    private int[][] grids;                               //15 × 10 网格的二维数组
    private int[][] NowGrids;                            //当前方块形态的二维数组
    private int row_number;                              //当前方块的总行数
    private int column_number;                           //当前方块的总列数
    private int column_start;                            //当前方块所在 grids 的列数
    //当前方块的颜色,0 表示灰色,1 代表红色,2 代表绿色,3 代表蓝绿色
    //4 代表品红色,5 代表蓝色,6 代表白色,7 代表黄色
    private int GridsColor;
    //19 种方块所占网格的位置所对应的数值
    private static final int[][] RedGrids1 = {{0, 3}, {0, 4}, {1, 4}, {1, 5}};
    private static final int[][] RedGrids2 = {{0, 5}, {1, 5}, {1, 4}, {2, 4}};
    private static final int[][] RedGrids1 = {{0, 3}, {0, 4}, {1, 4}, {1, 5}};
    private static final int[][] RedGrids2 = {{0, 5}, {1, 5}, {1, 4}, {2, 4}};
    private static final int[][] RedGrids1 = {{0, 3},{0, 4},{1, 4},{1, 5}};
    private static final int[][] RedGrids2 = {{0, 5},{1, 5},{1, 4},{2, 4}};
    private static final int[][] GreenGrids1 = {{0, 5},{0, 4},{1, 4},{1, 3}};
    private static final int[][] GreenGrids2 = {{0, 4},{1, 4},{1, 5},{2, 5}};
```

```
private static final int[][] CyanGrids1 = {{0, 4},{1, 4},{2, 4},{3, 4}};
private static final int[][] CyanGrids2 = {{0, 3},{0, 4},{0, 5},{0, 6}};
private static final int[][] MagentaGrids1 = {{0,4},{1, 3},{1, 4},{1, 5}};
private static final int[][] MagentaGrids2 = {{0,4},{1, 4},{1, 5},{2, 4}};
private static final int[][] MagentaGrids3 = {{0,3},{0, 4},{0, 5},{1, 4}};
private static final int[][] MagentaGrids4 = {{0,5},{1, 5},{1, 4},{2, 5}};
private static final int[][] BlueGrids1 = {{0, 3},{1, 3},{1, 4},{1, 5}};
private static final int[][] BlueGrids2 = {{0, 5},{0, 4},{1, 4},{2, 4}};
private static final int[][] BlueGrids3 = {{0, 3},{0, 4},{0, 5},{1, 5}};
private static final int[][] BlueGrids4 = {{0, 5},{1, 5},{2, 5},{2, 4}};
private static final int[][] WhiteGrids1 = {{0, 5},{1, 5},{1, 4},{1, 3}};
private static final int[][] WhiteGrids2 = {{0, 4},{1, 4},{2, 4},{2, 5}};
private static final int[][] WhiteGrids3 = {{0, 5},{0, 4},{0, 3},{1, 3}};
private static final int[][] WhiteGrids4 = {{0, 4},{0, 5},{1, 5},{2, 5}};
private static final int[][] YellowGrids = {{0, 4},{0, 5},{1, 5},{1, 4}};
private static final int grids_number = 4;     //方块的方格数量

@Override
public void onStart(Intent intent) {
    super.onStart(intent);

    initialize();
}

//初始化数据的函数
public void initialize() {
    layout = new DirectionalLayout(this); //对定向布局 layout 初始化
    //将二维数组 grids 初始化为 0
    grids = new int[height][width];
    for (int row = 0; row < height; row ++ )
        for (int column = 0; column < width; column ++ )
            grids[row][column] = 0;

    createGrids();
    drawGrids();
}

//随机重新生成一种颜色方块的函数
public void createGrids() {
    double random = Math.random(); //生成[0,1)的随机数
    //根据随机数的大小,调用相关的函数
    if (random >= 0 && random < 0.2) {
        if (random >= 0 && random < 0.1)
            createRedGrids1();
        else
            createRedGrids2();
```

```
        } else if (random >= 0.2 && random < 0.4) {
            if (random >= 0.2 && random < 0.3)
                createGreenGrids1();
            else
                createGreenGrids2();
        } else if (random >= 0.4 && random < 0.45) {
            if (random >= 0.4 && random < 0.43)
                createCyanGrids1();
            else
                createCyanGrids2();
        } else if (random >= 0.45 && random < 0.6) {
            if (random >= 0.45 && random < 0.48)
                createMagentaGrids1();
            else if (random >= 0.48 && random < 0.52)
                createMagentaGrids2();
            else if (random >= 0.52 && random < 0.56)
                createMagentaGrids3();
            else
                createMagentaGrids4();
        } else if (random >= 0.6 && random < 0.75) {
            if (random >= 0.6 && random < 0.63)
                createBlueGrids1();
            else if (random >= 0.63 && random < 0.67)
                createBlueGrids2();
            else if (random >= 0.67 && random < 0.71)
                createBlueGrids3();
            else
                createBlueGrids4();
        } else if (random >= 0.75 && random < 0.9) {
            if (random >= 0.75 && random < 0.78)
                createWhiteGrids1();
            else if (random >= 0.78 && random < 0.82)
                createWhiteGrids2();
            else if (random >= 0.82 && random < 0.86)
                createWhiteGrids3();
            else
                createWhiteGrids4();
        } else {
            createYellowGrids();
        }

        //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
        for (int row = 0; row < grids_number; row ++ ) {
            grids[NowGrids[row][0]][NowGrids[row][1]] = GridsColor;
        }
    }
```

```
        //绘制网格的函数
        public void drawGrids() {
            …
        }
        …
    }
```

单击主页面中的"开始"按钮，即可跳转到游戏页面，在游戏页面网格的顶部中间位置会显示一个方块。需要注意，每次运行时显示的方块可能不一致，运行效果如图 5-34 和图 5-35 所示。

图 5-34　主页面

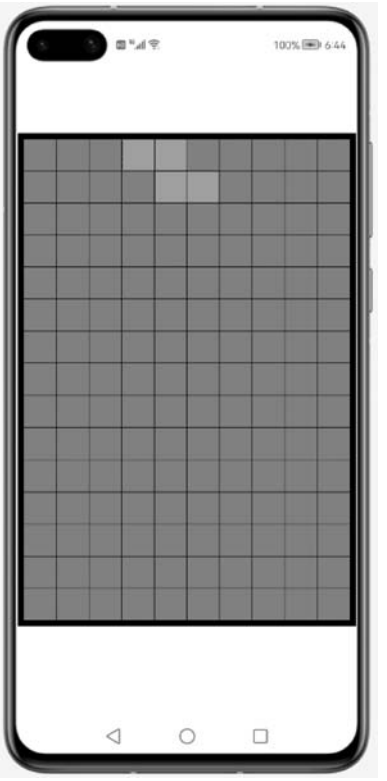

图 5-35　游戏页面

5.9　在游戏页面实现方块的下落

本节实现的运行效果：每 750ms 方块下落一格，直至下落到网格的底部或者其他方块的顶部为止，这时会重新随机生成一个新的方块。

本节的实现思路：通过 Timer 添加一个时间变量实现时间的流逝。先判断方块是否能

够下落，再实现将方块对应的二维数组整体向下移动一行，实现方块下落一格。

打开 ThirdAbilitySlice.java 文件。

定义变量方块下落时移动的行数 Nowrow，定义一个时间变量 timer。在 createGrids() 函数体内将 Nowrow 赋值为 0，因为每次生成方块时，方块均还没开始下落，所以将 Nowrow 赋值为 0。

添加一个名为 down()的函数，以判断方块能否下落。当 Nowrow＋row_number 为 15 时，即方块下落的行数与方块的行数之和为网格的竖列方格的数量，则表示方块已经下落到网格的底部，因此返回值为 false。当方块下方的方格的数值不为 0 时，即方块下方存在其他方块，则表示方块已经下落到其他方块的顶部，因此返回值为 false。如果不满足上述情况，则表示方块可以继续下落，因此返回值为 true，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;

import java.util.Timer;

public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;           //自定义定向布局
    private static final int length = 100;      //网格中方格的边长
    private static final int interval = 2;      //网格中方格的间距
    private static final int height = 15;       //网格中竖列方格的数量
    private static final int width = 10;        //网格中横列方格的数量
    private static final int left = 30;         //网格的左端距手机边界的距离
    private static final int top = 250;         //网格的顶端距手机边界的距离
    private static final int margin = 20;       //网格的外围距离
    private int[][] grids;                      //15 × 10 网格的二维数组
    private int[][] NowGrids;                   //当前方块形态的二维数组
    private int row_number;                     //当前方块的总行数
    private int column_number;                  //当前方块的总列数
    private int column_start;                   //当前方块所在 grids 的列数
    //当前方块的颜色，0 表示灰色，1 代表红色，2 代表绿色，3 代表蓝绿色
    //4 代表品红色，5 代表蓝色，6 代表白色，7 代表黄色
```

```
private int GridsColor;
//19 种方块所占网格的位置所对应的数值
private static final int[][] RedGrids1 = {{0, 3},{0, 4},{1, 4},{1, 5}};
private static final int[][] RedGrids2 = {{0, 5},{1, 5},{1, 4},{2, 4}};
private static final int[][] GreenGrids1 = {{0, 5},{0, 4},{1, 4},{1, 3}};
private static final int[][] GreenGrids2 = {{0, 4},{1, 4},{1, 5},{2, 5}};
private static final int[][] CyanGrids1 = {{0, 4},{1, 4},{2, 4},{3, 4}};
private static final int[][] CyanGrids2 = {{0, 3},{0, 4},{0, 5},{0, 6}};
private static final int[][] MagentaGrids1 = {{0,4},{1, 3},{1, 4},{1, 5}};
private static final int[][] MagentaGrids2 = {{0,4},{1, 4},{1, 5},{2, 4}};
private static final int[][] MagentaGrids3 = {{0,3},{0, 4},{0, 5},{1, 4}};
private static final int[][] MagentaGrids4 = {{0,5},{1, 5},{1, 4},{2, 5}};
private static final int[][] BlueGrids1 = {{0, 3},{1, 3},{1, 4},{1, 5}};
private static final int[][] BlueGrids2 = {{0, 5},{0, 4},{1, 4},{2, 4}};
private static final int[][] BlueGrids3 = {{0, 3},{0, 4},{0, 5},{1, 5}};
private static final int[][] BlueGrids4 = {{0, 5},{1, 5},{2, 5},{2, 4}};
private static final int[][] WhiteGrids1 = {{0, 5},{1, 5},{1, 4},{1, 3}};
private static final int[][] WhiteGrids2 = {{0, 4},{1, 4},{2, 4},{2, 5}};
private static final int[][] WhiteGrids3 = {{0, 5},{0, 4},{0, 3},{1, 3}};
private static final int[][] WhiteGrids4 = {{0, 4},{0, 5},{1, 5},{2, 5}};
private static final int[][] YellowGrids = {{0, 4},{0, 5},{1, 5},{1, 4}};
private static final int grids_number = 4;        //方块的方格数量
private int Nowrow;                               //方块下落移动的行数
private Timer timer;                              //时间变量

@Override
public void onStart(Intent intent) {
    super.onStart(intent);

    initialize();
}

//初始化数据的函数
public void initialize() {
    layout = new DirectionalLayout(this);         //对定向布局 layout 初始化
    Gameover = true;
    //将二维数组 grids 初始化为 0
    grids = new int[height][width];
    for (int row = 0; row < height; row ++ )
        for (int column = 0; column < width; column ++ )
            grids[row][column] = 0;

    createGrids();
    drawGrids();
}
```

```
//随机重新生成一种颜色方块的函数
public void createGrids() {
    Nowrow = 0;

    double random = Math.random();        //生成[0,1)的随机数
    //根据随机数的大小,调用相关的函数
    if (random >= 0 && random < 0.2) {
        if (random >= 0 && random < 0.1)
            createRedGrids1();
        else
            createRedGrids2();
    } else if (random >= 0.2 && random < 0.4) {
        if (random >= 0.2 && random < 0.3)
            createGreenGrids1();
        else
            createGreenGrids2();
    } else if (random >= 0.4 && random < 0.45) {
        if (random >= 0.4 && random < 0.43)
            createCyanGrids1();
        else
            createCyanGrids2();
    } else if (random >= 0.45 && random < 0.6) {
        if (random >= 0.45 && random < 0.48)
            createMagentaGrids1();
        else if (random >= 0.48 && random < 0.52)
            createMagentaGrids2();
        else if (random >= 0.52 && random < 0.56)
            createMagentaGrids3();
        else
            createMagentaGrids4();
    } else if (random >= 0.6 && random < 0.75) {
        if (random >= 0.6 && random < 0.63)
            createBlueGrids1();
        else if (random >= 0.63 && random < 0.67)
            createBlueGrids2();
        else if (random >= 0.67 && random < 0.71)
            createBlueGrids3();
        else
            createBlueGrids4();
    } else if (random >= 0.75 && random < 0.9) {
        if (random >= 0.75 && random < 0.78)
            createWhiteGrids1();
        else if (random >= 0.78 && random < 0.82)
            createWhiteGrids2();
        else if (random >= 0.82 && random < 0.86)
            createWhiteGrids3();
```

```
                else
                    createWhiteGrids4();
            } else {
                createYellowGrids();
            }

            //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
            for (int row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0]][NowGrids[row][1]] = GridsColor;
            }
        }

        //绘制网格的函数
        public void drawGrids() {
            …
        }

        //判断方块能否下落的函数
        public boolean down() {
            boolean k;
            //表示方块已经接触到网格的底端
            if (Nowrow + row_number == height) {
                return false;
            }

            //判断方块的下一行是否存在其他方块
            for (int row = 0; row < grids_number; row ++ ) {
                k = true;
                for (int i = 0; i < grids_number; i ++ ) {
                     if (NowGrids[row][0] + 1 == NowGrids[i][0] && NowGrids[row][1] ==
NowGrids[i][1]) {
                        k = false;
                    }
                }
                if (k) {
                    if (grids[NowGrids[row][0] + Nowrow + 1][NowGrids[row][1]] != 0)
                        return false;
                }
            }

            return true;
        }

        //对对应颜色方块的不同形态赋予 NowGrids、row_number、column_number
        //GridsColor、column_start 的值
        public void createRedGrids1() {
```

```
            NowGrids = RedGrids1;
            row_number = 2;
            column_number = 3;
            GridsColor = 1;
            column_start = 3;
        }
        …
    }
```

添加一个名为 run()的函数,以实现方块随着时间流逝逐渐下落。对时间变量 timer 初始化,添加时间任务,延迟为 0,间隔为 750ms。在时间任务中,判断 down()函数的返回值,当返回值为 true 时,实现方块下落一行,并且 Nowrow 加 1。当返回值为 false 时,调用函数 createGrids(),重新随机生成一个新方块。在生命周期事件 onStart()中,调用函数 run(),代码如下:

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;           //自定义定向布局
    private static final int length = 100;      //网格中方格的边长
    private static final int interval = 2;      //网格中方格的间距
    private static final int height = 15;       //网格中竖列方格的数量
    private static final int width = 10;        //网格中横列方格的数量
    private static final int left = 30;         //网格的左端距手机边界的距离
    private static final int top = 250;         //网格的顶端距手机边界的距离
    private static final int margin = 20;       //网格的外围距离
    private int[][] grids;                      //15×10 网格的二维数组
    private int[][] NowGrids;                   //当前方块形态的二维数组
```

```
private int row_number;                          //当前方块的总行数
private int column_number;                       //当前方块的总列数
private int column_start;                        //当前方块所在 grids 的列数
//当前方块的颜色,0 表示灰色,1 代表红色,2 代表绿色,3 代表蓝绿色
//4 代表品红色,5 代表蓝色,6 代表白色,7 代表黄色
private int GridsColor;
//19 种方块所占网格的位置对应的数值
private static final int[][] RedGrids1 = {{0, 3},{0, 4},{1, 4},{1, 5}};
private static final int[][] RedGrids2 = {{0, 5},{1, 5},{1, 4},{2, 4}};
private static final int[][] GreenGrids1 = {{0, 5},{0, 4},{1, 4},{1, 3}};
private static final int[][] GreenGrids2 = {{0, 4},{1, 4},{1, 5},{2, 5}};
private static final int[][] CyanGrids1 = {{0, 4},{1, 4},{2, 4},{3, 4}};
private static final int[][] CyanGrids2 = {{0, 3},{0, 4},{0, 5},{0, 6}};
private static final int[][] MagentaGrids1 = {{0,4},{1, 3},{1, 4},{1, 5}};
private static final int[][] MagentaGrids2 = {{0,4},{1, 4},{1, 5},{2, 4}};
private static final int[][] MagentaGrids3 = {{0,3},{0, 4},{0, 5},{1, 4}};
private static final int[][] MagentaGrids4 = {{0,5},{1, 5},{1, 4},{2, 5}};
private static final int[][] BlueGrids1 = {{0, 3},{1, 3},{1, 4},{1, 5}};
private static final int[][] BlueGrids2 = {{0, 5},{0, 4},{1, 4},{2, 4}};
private static final int[][] BlueGrids3 = {{0, 3},{0, 4},{0, 5},{1, 5}};
private static final int[][] BlueGrids4 = {{0, 5},{1, 5},{2, 5},{2, 4}};
private static final int[][] WhiteGrids1 = {{0, 5},{1, 5},{1, 4},{1, 3}};
private static final int[][] WhiteGrids2 = {{0, 4},{1, 4},{2, 4},{2, 5}};
private static final int[][] WhiteGrids3 = {{0, 5},{0, 4},{0, 3},{1, 3}};
private static final int[][] WhiteGrids4 = {{0, 4},{0, 5},{1, 5},{2, 5}};
private static final int[][] YellowGrids = {{0, 4},{0, 5},{1, 5},{1, 4}};
private static final int grids_number = 4;   //方块的方格数量
private int Nowrow;                              //方块下落移动的行数
private Timer timer;                             //时间变量

@Override
public void onStart(Intent intent) {
    super.onStart(intent);

    initialize();
    run();
}

//初始化数据的函数
public void initialize() {
    …
}
//随机重新生成一种颜色方块的函数
public void createGrids() {
    …
}
```

```
    //绘制网格的函数
    public void drawGrids() {
        …
    }

    //方块自动下落的函数
    public void run() {
        timer = new Timer(); //初始化时间变量
        //设置时间任务,延迟为 0,间隔为 750ms
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                getUITaskDispatcher().asyncDispatch(() -> {
                    //如果能够下落,则下落一行
                    if (down()) {
                        //将原来方块的颜色清除
                        for (int row = 0; row < grids_number; row ++ ) {
                            grids[NowGrids[row][0] + Nowrow][NowGrids[row][1]] = 0;
                        }
                        Nowrow ++ ;
                        //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
                        for (int row = 0; row < grids_number; row ++ ) {
                            grids[NowGrids[row][0] + Nowrow][NowGrids[row][1]] =
GridsColor;
                        }
                    } else {
                        //如果不能下落,则重新随机生成一种颜色方块
                        createGrids();
                    }
                    //重新绘制网格
                    drawGrids();
                });
            }
        }, 0, 750);
    }

    //判断方块能否下落的函数
    public boolean down() {
        boolean k;
        //表示方块已经接触到网格的底端
        if (Nowrow + row_number == height) {
            return false;
        }

        //判断方块的下一行是否存在其他方块
        for (int row = 0; row < grids_number; row ++ ) {
            k = true;
            for (int i = 0; i < grids_number; i ++ ) {
```

```
                    if (NowGrids[row][0] + 1 == NowGrids[i][0] && NowGrids[row][1] ==
NowGrids[i][1]) {
                        k = false;
                    }
                }
                if (k) {
                    if (grids[NowGrids[row][0] + Nowrow + 1][NowGrids[row][1]] != 0)
                        return false;
                }
            }

            return true;
        }
        …
}
```

单击主页面中的"开始"按钮，即可跳转到游戏页面，每 750ms 方块下落一格，直至下落到网格的底部或者其他方块的顶部为止，这时会重新随机生成一个新的方块。需要注意，因为每次生成的方块都是随机的，所以每次的运行效果都可能不一致，运行效果如图 5-36 和图 5-37 所示。

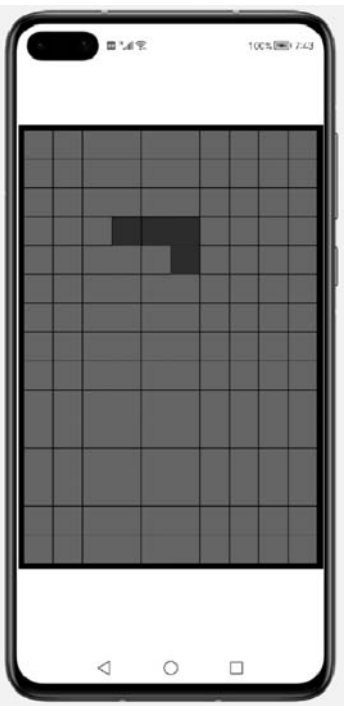

图 5-36　随机生成颜色方块

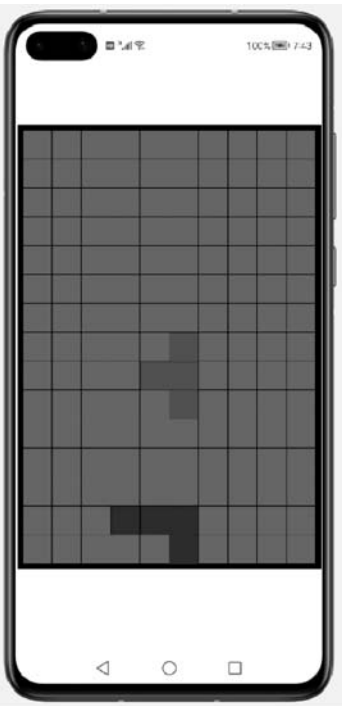

图 5-37　颜色方块下落

5.10 在游戏页面添加5个按钮并向主页面跳转

本节实现的运行效果：在游戏页面网格的下方显示5个按钮，在按钮上显示的文本分别为“←”“变”“→”“重新开始”“返回”。单击游戏页面中的“返回”按钮，跳转到主页面。

本节的实现思路：通过ShapeElement设置按钮的背景样式，配置按钮的属性。通过在单击事件中调用present()语句实现页面间的跳转，在调用该语句时通过指定AbilitySlice名称达到指定跳转目标的页面。

打开ThirdAbilitySlice.java文件。

添加一个名为drawButton()的函数，以实现绘制5个按钮。在函数体内定义并初始化按钮的背景样式background，将RGB颜色setRgbColor设置为(120,198,197)，将圆角半径setCornerRadius设置为100。

添加一个按钮button_left，将文本setText设置为“←”，将文本的对齐方式setTextAlignment设置为TextAlignment.CENTER(居中)，将文本的颜色setTextColor设置为Color.WHITE(白色)，将文本的大小setTextSize设置为100。将按钮的上边距setMarginTop设置为1800，将按钮的左边距setMarginLef设置为160，将按钮的内边距setPadding设置为(10,0,10,0)，将按钮的背景样式setBackground设置为background。最后将设置好样式的按钮添加到布局layout中。

添加一个按钮button_change，将文本设置为“变”，将文本的对齐方式setTextAlignment设置为TextAlignment.CENTER(居中)，将文本的颜色setTextColor设置为Color.WHITE(白色)，将文本的大小setTextSize设置为100。将按钮的上边距setMarginTop设置为－135，将按钮的左边距setMarginLef设置为480，将按钮的内边距setPadding设置为(10,0,10,0)，将按钮的背景样式setBackground设置为background。最后将设置好样式的按钮添加到布局layout中。

添加一个按钮button_right，将文本设置为“→”，将文本的对齐方式setTextAlignment设置为TextAlignment.CENTER(居中)，将文本的颜色setTextColor设置为Color.WHITE(白色)，将文本的大小setTextSize设置为100。将按钮的上边距setMarginTop设置为－135，将按钮的左边距setMarginLef设置为780，将按钮的内边距setPadding设置为(10,0,10,0)，将按钮的背景样式setBackground设置为background。最后将设置好样式的按钮添加到布局layout中。

在函数initialize()中调用函数drawButton()，实现所添加按钮的绘制，代码如下：

```
//第5章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
```

```
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }

    //初始化数据的函数
    public void initialize() {
        layout = new DirectionalLayout(this);            //对定向布局 layout 初始化
        Gameover = true;
        //将二维数组 grids 初始化为 0
        grids = new int[height][width];
        for (int row = 0; row < height; row ++ )
            for (int column = 0; column < width; column ++ )
                grids[row][column] = 0;

        createGrids();
        drawButton();
        drawGrids();
    }

    //随机重新生成一种颜色方块的函数
    public void createGrids() {
        …
    }

    //绘制按钮的函数
    public void drawButton() {
        //设置背景图层
```

```
        ShapeElement background = new ShapeElement();
        background.setRgbColor(new RgbColor(120, 198, 197));
        background.setCornerRadius(100);

        Button button_left = new Button(this);                    //初始化按钮
        button_left.setText("←");                                 //设置按钮的文本
        //设置按钮文本的对齐方式
        button_left.setTextAlignment(TextAlignment.CENTER);
        button_left.setTextColor(Color.WHITE);                    //设置文本的颜色
        button_left.setTextSize(100);                             //设置按钮文本的大小
        button_left.setMarginTop(1800);                           //设置按钮的上外边距
        button_left.setMarginLeft(160);                           //设置按钮的左外边距
        button_left.setPadding(10, 0, 10, 0);                     //设置按钮的内边距
        button_left.setBackground(background);                    //设置按钮的背景图层
        layout.addComponent(button_left);

        Button button_change = new Button(this);
        button_change.setText("变");
        button_change.setTextAlignment(TextAlignment.CENTER);
        button_change.setTextColor(Color.WHITE);
        button_change.setTextSize(100);
        button_change.setMarginLeft(480);
        button_change.setMarginTop( - 135);
        button_change.setPadding(10, 0, 10, 0);
        button_change.setBackground(background);
        layout.addComponent(button_change);

        Button button_right = new Button(this);
        button_right.setText("→");
        button_right.setTextAlignment(TextAlignment.CENTER);
        button_right.setTextColor(Color.WHITE);
        button_right.setTextSize(100);
        button_right.setMarginLeft(780);
        button_right.setMarginTop( - 135);
        button_right.setPadding(10, 0, 10, 0);
        button_right.setBackground(background);
        layout.addComponent(button_right);
    }

    //绘制网格的函数
    public void drawGrids() {
        …
    }
    …
}
```

添加一个按钮 button_start，将文本 setText 设置为"重新开始"，将文本的对齐方式 setTextAlignment 设置为 TextAlignment.CENTER(居中)，将文本的颜色 setTextColor 设置为 Color.WHITE(白色)，将文本的大小 setTextSize 设置为 100。将按钮的上边距 setMarginTop 设置为 5，将按钮的左边距 setMarginLef 设置为 180，将按钮的内边距 setPadding 设置为(10,10,10,10)，将按钮的背景样式 setBackground 设置为 background。最后将设置好样式的按钮添加到布局 layout 中。

添加一个按钮 button_back，将文本 setText 设置为"返回"，将文本的对齐方式 setTextAlignment 设置为 TextAlignment.CENTER(居中)，将文本的颜色 setTextColor 设置为 Color.WHITE(白色)，将文本的大小 setTextSize 设置为 100。将按钮的上边距 setMarginTop 设置为－150，将按钮的左边距 setMarginLef 设置为 680，将按钮的内边距 setPadding 设置为(10,10,10,10)，将按钮的背景样式 setBackground 设置为 background。增加单击事件，通过 present()语句跳转到 MainAbilitySlice，当单击按钮时就会触发按钮的单击事件，从而跳转到主页面。最后将设置好样式的按钮添加到布局 layout 中，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    ...
    @Override
    public void onStart(Intent intent) {
        ...
    }

    //初始化数据的函数
```

```
public void initialize() {
    ...
}

//随机重新生成一种颜色方块的函数
public void createGrids() {
    ...
}

//绘制按钮的函数
public void drawButton() {
    //设置背景图层
    ShapeElement background = new ShapeElement();
    background.setRgbColor(new RgbColor(120, 198, 197));
    background.setCornerRadius(100);

    Button button_left = new Button(this);              //初始化按钮
    button_left.setText("←");                           //设置按钮的文本
    //设置按钮文本的对齐方式
    button_left.setTextAlignment(TextAlignment.CENTER);
    button_left.setTextColor(Color.WHITE);              //设置文本的颜色
    button_left.setTextSize(100);                       //设置按钮文本的大小
    button_left.setMarginTop(1800);                     //设置按钮的上外边距
    button_left.setMarginLeft(160);                     //设置按钮的左外边距
    button_left.setPadding(10, 0, 10, 0);               //设置按钮的内边距
    button_left.setBackground(background);              //设置按钮的背景图层
    layout.addComponent(button_left);

    Button button_change = new Button(this);
    button_change.setText("变");
    button_change.setTextAlignment(TextAlignment.CENTER);
    button_change.setTextColor(Color.WHITE);
    button_change.setTextSize(100);
    button_change.setMarginLeft(480);
    button_change.setMarginTop(-135);
    button_change.setPadding(10, 0, 10, 0);
    button_change.setBackground(background);
    layout.addComponent(button_change);

    Button button_right = new Button(this);
    button_right.setText("→");
    button_right.setTextAlignment(TextAlignment.CENTER);
    button_right.setTextColor(Color.WHITE);
    button_right.setTextSize(100);
    button_right.setMarginLeft(780);
    button_right.setMarginTop(-135);
```

```
        button_right.setPadding(10, 0, 10, 0);
        button_right.setBackground(background);
        layout.addComponent(button_right);

        Button button_start = new Button(this);
        button_start.setText("重新开始");
        button_start.setTextSize(100);
        button_start.setTextAlignment(TextAlignment.CENTER);
        button_start.setTextColor(Color.WHITE);
        button_start.setMarginTop(5);
        button_start.setMarginLeft(180);
        button_start.setPadding(10, 10, 10, 10);
        button_start.setBackground(background);
        layout.addComponent(button_start);

        Button button_back = new Button(this);
        button_back.setText("返回");
        button_back.setTextSize(100);
        button_back.setTextAlignment(TextAlignment.CENTER);
        button_back.setTextColor(Color.WHITE);
        button_back.setMarginTop( - 150);
        button_back.setMarginLeft(680);
        button_back.setPadding(10, 10, 10, 10);
        button_back.setBackground(background);
        //设置按钮的单击事件
        button_back.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                //跳转到 MainAbilitySlice()语句
                present(new MainAbilitySlice(),new Intent());
            }
        });
        layout.addComponent(button_back);
    }

    //绘制网格的函数
    public void drawGrids() {
        …
    }
    …
}
```

单击主页面中的"开始"按钮，即可跳转到游戏页面，在游戏页面网格的下方会显示 5 个按钮，按钮上显示的文本分别为"←""变""→""重新开始""返回"。单击游戏页面中的"返回"按钮，跳转到主页面，运行效果如图 5-38 和图 5-39 所示。

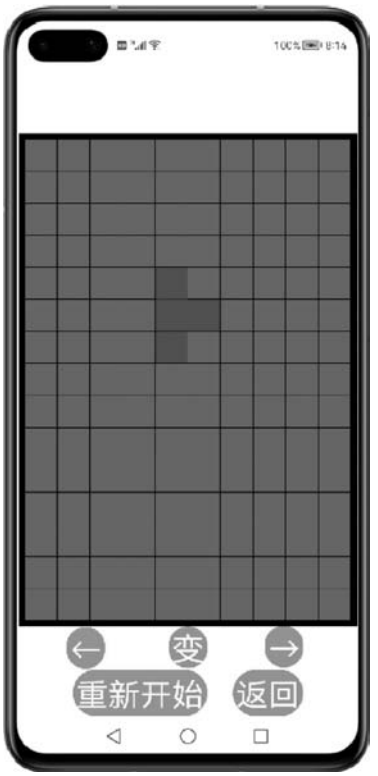

图 5-38　游戏页面

图 5-39　主页面

5.11　在游戏页面实现方块向左移动

本节实现的运行效果：当单击“←”按钮时，正在下落的方块会向左移动一格，如果正在下落的方块位于网格的左端或其左端存在其他方块，则不会再向左移动了。

本节的实现思路：先判断方块是否能够向左移动，再实现将方块对应的二维数组整体向左移动一列，实现方块向左移动一格。

打开 ThirdAbilitySlice.java 文件。

定义变量方块的左右移动的列数 Nowcolumn。在 createGrids() 函数体内将 Nowcolumn 赋值为 0，因为每次生成方块时，方块均没开始向左移动或向右移动，所以将 Nowcolumn 赋值为 0。特别需要注意的是，这里向左移动一格表示减 1，向右移动一格表示加 1。

添加一个名为 left() 的函数，以判断方块能否向左移动。当 Nowcolumn＋column_start 为 0 时，即方块向左移动的列数与方块的列数之和为 0，则表示方块已经向左移动到网格的左端，因此返回值为 false。当方块左端的方格的数值不为 0 时，即方块左端存在其他

方块，则表示方块已经向左移动到其他方块的右端，因此返回值为 false。如果不满足上述情况，则表示方块可以继续向左移动，因此返回值为 true，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;           //自定义定向布局
    private static final int length = 100;      //网格中方格的边长
    private static final int interval = 2;      //网格中方格的间距
    private static final int height = 15;       //网格中竖列方格的数量
    private static final int width = 10;        //网格中横列方格的数量
    private static final int left = 30;         //网格的左端距手机边界的距离
    private static final int top = 250;         //网格的顶端距手机边界的距离
    private static final int margin = 20;       //网格的外围距离
    private int[][] grids;                      //15×10 网格的二维数组
    private int[][] NowGrids;                   //当前方块形态的二维数组
    private int row_number;                     //当前方块的总行数
    private int column_number;                  //当前方块的总列数
    private int column_start;                   //当前方块所在 grids 的列数
    //当前方块的颜色，0 表示灰色，1 代表红色，2 代表绿色，3 代表蓝绿色
    //4 代表品红色，5 代表蓝色，6 代表白色，7 代表黄色
    private int GridsColor;
    //19 种方块所占网格的位置所对应的数值
    private static final int[][] RedGrids1 = {{0, 3},{0, 4},{1, 4},{1, 5}};
    private static final int[][] RedGrids2 = {{0, 5},{1, 5},{1, 4},{2, 4}};
    private static final int[][] GreenGrids1 = {{0, 5},{0, 4},{1, 4},{1, 3}};
    private static final int[][] GreenGrids2 = {{0, 4},{1, 4},{1, 5},{2, 5}};
```

```
private static final int[][] CyanGrids1 = {{0, 4},{1, 4},{2, 4},{3, 4}};
private static final int[][] CyanGrids2 = {{0, 3},{0, 4},{0, 5},{0, 6}};
private static final int[][] MagentaGrids1 = {{0,4},{1, 3},{1, 4},{1, 5}};
private static final int[][] MagentaGrids2 = {{0,4},{1, 4},{1, 5},{2, 4}};
private static final int[][] MagentaGrids3 = {{0,3},{0, 4},{0, 5},{1, 4}};
private static final int[][] MagentaGrids4 = {{0,5},{1, 5},{1, 4},{2, 5}};
private static final int[][] BlueGrids1 = {{0, 3},{1, 3},{1, 4},{1, 5}};
private static final int[][] BlueGrids2 = {{0, 5},{0, 4},{1, 4},{2, 4}};
private static final int[][] BlueGrids3 = {{0, 3},{0, 4},{0, 5},{1, 5}};
private static final int[][] BlueGrids4 = {{0, 5},{1, 5},{2, 5},{2, 4}};
private static final int[][] WhiteGrids1 = {{0, 5},{1, 5},{1, 4},{1, 3}};
private static final int[][] WhiteGrids2 = {{0, 4},{1, 4},{2, 4},{2, 5}};
private static final int[][] WhiteGrids3 = {{0, 5},{0, 4},{0, 3},{1, 3}};
private static final int[][] WhiteGrids4 = {{0, 4},{0, 5},{1, 5},{2, 5}};
private static final int[][] YellowGrids = {{0, 4},{0, 5},{1, 5},{1, 4}};
private static final int grids_number = 4;                    //方块的方格数量
private int Nowrow;                                           //方块下落移动的行数
private int Nowcolumn;                                        //方块左右移动的列数
private Timer timer;                                          //时间变量
private boolean Gameover;

@Override
public void onStart(Intent intent) {
    ...
}
//初始化数据的函数
public void initialize() {
    ...
}

//随机重新生成一种颜色方块的函数
public void createGrids() {
    Nowrow = 0;
    Nowcolumn = 0;

     double random = Math.random();                           //生成[0,1)的随机数
    //根据随机数的大小,调用相关的函数
    if (random >= 0 && random < 0.2) {
        if (random >= 0 && random < 0.1)
            createRedGrids1();
        else
            createRedGrids2();
    } else if (random >= 0.2 && random < 0.4) {
        if (random >= 0.2 && random < 0.3)
            createGreenGrids1();
        else
```

```
                createGreenGrids2();
        } else if (random >= 0.4 && random < 0.45) {
            if (random >= 0.4 && random < 0.43)
                createCyanGrids1();
            else
                createCyanGrids2();
        } else if (random >= 0.45 && random < 0.6) {
            if (random >= 0.45 && random < 0.48)
                createMagentaGrids1();
            else if (random >= 0.48 && random < 0.52)
                createMagentaGrids2();
            else if (random >= 0.52 && random < 0.56)
                createMagentaGrids3();
            else
                createMagentaGrids4();
        } else if (random >= 0.6 && random < 0.75) {
            if (random >= 0.6 && random < 0.63)
                createBlueGrids1();
            else if (random >= 0.63 && random < 0.67)
                createBlueGrids2();
            else if (random >= 0.67 && random < 0.71)
                createBlueGrids3();
            else
                createBlueGrids4();
        } else if (random >= 0.75 && random < 0.9) {
            if (random >= 0.75 && random < 0.78)
                createWhiteGrids1();
            else if (random >= 0.78 && random < 0.82)
                createWhiteGrids2();
            else if (random >= 0.82 && random < 0.86)
                createWhiteGrids3();
            else
                createWhiteGrids4();
        } else {
            createYellowGrids();
        }

        //将颜色方块添加到 15×10 网格的二维数组 grids 中
        for (int row = 0; row < grids_number; row++) {
            grids[NowGrids[row][0]][NowGrids[row][1]] = GridsColor;
        }
    }

    //绘制按钮的函数
    public void drawButton() {
        …
```

```
    }

    //绘制网格的函数
    public void drawGrids() {
        …
    }

    //方块自动下落的函数
    public void run() {
        …
    }

    //判断方块能否下落的函数
    public boolean down() {
        …
    }

    //判断方块能否向左移动的函数
    public boolean left() {
        boolean k;
        //表示方块已经接触到网格的左端
        if (Nowcolumn + column_start == 0) {
            return false;
        }

        //表示方块的左一列是否存在其他方块
        for (int column = 0; column < grids_number; column ++ ) {
            k = true;
            for (int j = 0; j < grids_number; j ++ ) {
                if (NowGrids[column][0] == NowGrids[j][0] &&
                      NowGrids[column][1] - 1 == NowGrids[j][1]) {
                    k = false;
                }
            }
            if (k) {
                if (grids[NowGrids[column][0] + Nowrow][NowGrids[column][1] + Nowcolumn -
1] != 0)
                    return false;
            }
        }

        return true;
    }

    //对对应颜色方块的不同形态赋予 NowGrids、row_number、column_number
    //GridsColor、column_start 的值
```

```
    public void createRedGrids1() {
        NowGrids = RedGrids1;
        row_number = 2;
        column_number = 3;
        GridsColor = 1;
        column_start = 3;
    }
    …
}
```

添加一个名为 leftShift()的函数,以实现方块向左移动。在函数体内判断函数 left()的返回值,当返回值为 true 时,实现方块向左移动一格,并且 Nowrow 减 1。在函数 drawButton()内的按钮 button_left 增加单击事件,调用函数 leftShift(),代码如下:

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }
    //初始化数据的函数
    public void initialize() {
        …
    }
```

```
//随机重新生成一种颜色方块的函数
public void createGrids() {
    ...
}

 //绘制按钮的函数
 public void drawButton() {
    //设置背景图层
    ShapeElement background = new ShapeElement();
    background.setRgbColor(new RgbColor(120, 198, 197));
    background.setCornerRadius(100);

    Button button_left = new Button(this);              //初始化按钮
    button_left.setText("←");                           //设置按钮的文本
    //设置按钮文本的对齐方式
    button_left.setTextAlignment(TextAlignment.CENTER);
    button_left.setTextColor(Color.WHITE);              //设置文本的颜色
    button_left.setTextSize(100);                       //设置按钮文本的大小
    button_left.setMarginTop(1800);                     //设置按钮的上外边距
    button_left.setMarginLeft(160);                     //设置按钮的左外边距
    button_left.setPadding(10, 0, 10, 0);               //设置按钮的内边距
    button_left.setBackground(background);              //设置按钮的背景图层
    //设置按钮的单击事件
    button_left.setClickedListener(new Component.ClickedListener() {
        @Override
        public void onClick(Component component) {
            leftShift();
        }
    });
    layout.addComponent(button_left);

    Button button_change = new Button(this);
    button_change.setText("变");
    button_change.setTextAlignment(TextAlignment.CENTER);
    button_change.setTextColor(Color.WHITE);
    button_change.setTextSize(100);
    button_change.setMarginLeft(480);
    button_change.setMarginTop(-135);
    button_change.setPadding(10, 0, 10, 0);
    button_change.setBackground(background);
    layout.addComponent(button_change);

    Button button_right = new Button(this);
    button_right.setText("→");
    button_right.setTextAlignment(TextAlignment.CENTER);
    button_right.setTextColor(Color.WHITE);
```

```
        button_right.setTextSize(100);
        button_right.setMarginLeft(780);
        button_right.setMarginTop(-135);
        button_right.setPadding(10, 0, 10, 0);
        button_right.setBackground(background);
        layout.addComponent(button_right);

        Button button_start = new Button(this);
        button_start.setText("重新开始");
        button_start.setTextSize(100);
        button_start.setTextAlignment(TextAlignment.CENTER);
        button_start.setTextColor(Color.WHITE);
        button_start.setMarginTop(5);
        button_start.setMarginLeft(180);
        button_start.setPadding(10, 10, 10, 10);
        button_start.setBackground(background);
        layout.addComponent(button_start);

        Button button_back = new Button(this);
        button_back.setText("返回");
        button_back.setTextSize(100);
        button_back.setTextAlignment(TextAlignment.CENTER);
        button_back.setTextColor(Color.WHITE);
        button_back.setMarginTop(-150);
        button_back.setMarginLeft(680);
        button_back.setPadding(10, 10, 10, 10);
        button_back.setBackground(background);
        //设置按钮的单击事件
        button_back.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                //跳转到 MainAbilitySlice()语句
                present(new MainAbilitySlice(),new Intent());
            }
        });
        layout.addComponent(button_back);
    }

    //绘制网格的函数
    public void drawGrids() {
        …
    }

    //方块自动下落的函数
    public void run() {
        …
```

```
    }

    //判断方块能否下落的函数
    public boolean down() {
        …
    }

    //实现方块向左移动的函数
    public void leftShift() {
        if (left()) {
            //将原来方块的颜色清除
            for (int row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] = 0;
            }
            Nowcolumn -- ;
            //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
            for (int row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] = GridsColor;
            }
        }
        //重新绘制网格
        drawGrids();
    }

    //判断方块能否向左移动的函数
    public boolean left() {
        …
    }
    …
}
```

为了保证方块向左移动时，方块的下落仍然能够正常实现，需要对判断方块能否下落和绘制下落方块的语句进行修改。在函数 down()内，判断方块能否下落并对方块的左右移动的列数 Nowcolumn 进行修改。同样在函数 run()内，绘制方块下落并对方块的左右移动的列数 Nowcolumn 进行修改，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
```

```
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }
    //初始化数据的函数
    public void initialize() {
        …
    }

    //随机重新生成一种颜色方块的函数
    public void createGrids() {
        …
    }

    //绘制按钮的函数
    public void drawButton() {
        …
    }

    //绘制网格的函数
    public void drawGrids() {
        …
    }

    //方块自动下落的函数
    public void run() {
        timer = new Timer();            //初始化时间变量
        //设置时间任务,延迟为 0,间隔为 750ms
        timer.schedule(new TimerTask() {
            @Override
```

```
            public void run() {
                getUITaskDispatcher().asyncDispatch(() -> {
                    //如果能够下落,则下落一行
                    if (down()) {
                        //将原来方块的颜色清除
                        for (int row = 0; row < grids_number; row ++ ) {
                            grids[NowGrids[row][0] + Nowrow][NowGrids[row][1]] = 0;
                            grids[ NowGrids [ row ] [ 0 ]  +  Nowrow ] [ NowGrids [ row ] [ 1 ]  +
Nowcolumn] = 0;
                        }
                        Nowrow ++ ;
                        //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
                        for (int row = 0; row < grids_number; row ++ ) {
                            grids[NowGrids[row][0] + Nowrow][NowGrids[row][1]] = GridsColor;
                            grids[ NowGrids [ row ] [ 0 ]  +  Nowrow ] [ NowGrids [ row ] [ 1 ]  +
Nowcolumn] = GridsColor;
                        }
                    } else {
                        //如果不能下落,则重新随机生成一种颜色方块
                        createGrids();
                    }
                    //重新绘制网格
                    drawGrids();
                });
            }
        }, 0, 750);
    }

    //判断方块能否下落的函数
    public boolean down() {
        boolean k;
        //表示方块已经接触到网格的底端
        if (Nowrow + row_number == height) {
            return false;
        }

        //判断方块的下一行是否存在其他方块
        for (int row = 0; row < grids_number; row ++ ) {
            k = true;
            for (int i = 0; i < grids_number; i ++ ) {
                 if (NowGrids[row][0] + 1 == NowGrids[i][0] && NowGrids[row][1] ==
NowGrids[i][1]) {
                    k = false;
                }
            }
```

```
                if (k) {
                    if (grids[NowGrids[row][0] + Nowrow + 1][NowGrids[row][1]] != 0)
                    if (grids[NowGrids[row][0] + Nowrow + 1][NowGrids[row][1] + Nowcolumn] != 0)
                        return false;
                }
            }

            return true;
        }

        //实现方块向左移动的函数
        public void leftShift() {
            …
        }
        …
    }
```

进入游戏页面，当每次单击"←"按钮时，正在下落的方块会向左移动一格，如果正在下落的方块位于网格的左端或其左端存在其他方块，则不会再向左移动了，运行效果如图 5-40 和图 5-41 所示。

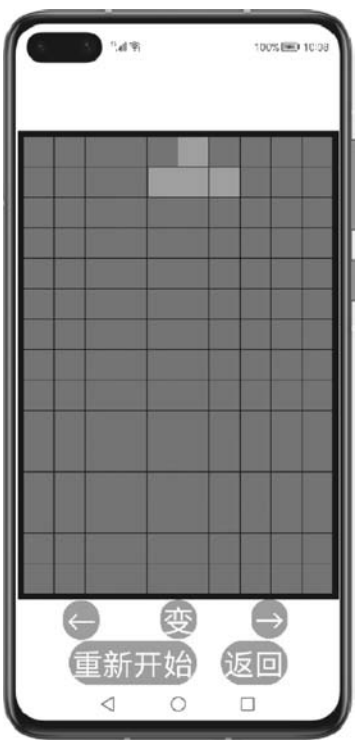

图 5-40　向左移动前

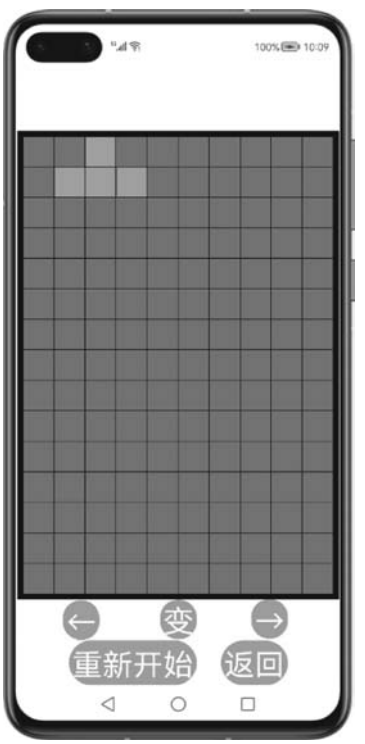

图 5-41　向左移动后

5.12 在游戏页面实现方块向右移动

本节实现的运行效果：当单击“→”按钮时，正在下落的方块会向右移动一格，如果正在下落的方块位于网格的右端或其右端存在其他方块，则不会再向右移动了。

本节的实现思路：先判断方块是否能够向右移动，再实现将方块对应的二维数组整体向右移动一列，实现方块向右移动一格。

打开 ThirdAbilitySlice.java 文件。

添加一个名为 right()的函数，以判断方块能否向右移动。当 Nowcolumn＋column_number ＋column_start 为 10 时，即方块向右移动的列数、方块的列数与方块的第 1 个方格所在二维数组的列数之和为网格的横列方格的数量，则表示方块已经向右移动到网格的右端，因此返回值为 false。当方块右端的方格的数值不为 0 时，即方块右端存在其他方块，则方块已经向右移动到其他方块的左端，因此返回值为 false。如果不满足上述情况，则表示方块可以继续向右移动，因此返回值为 true，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }
```

```
    …

    //判断方块能否向左移动的函数
    public boolean left() {
        …
    }

    //判断方块能否向右移动的函数
    public boolean right() {
        boolean k;
        //表示方块已经接触到网格的右端
        if (Nowcolumn + column_number + column_start == width) {
            return false;
        }

        //表示方块的右一列是否存在其他方块
        for (int column = 0; column < grids_number; column ++ ) {
            k = true;
            for (int j = 0; j < grids_number; j ++ ) {
                if (NowGrids[column][0] == NowGrids[j][0]
                      && NowGrids[column][1] + 1 == NowGrids[j][1]) {
                    k = false;
                }
            }
            if (k) {
                if (grids[NowGrids[column][0] + Nowrow][NowGrids[column][1] + Nowcolumn +
1] != 0)
                    return false;
            }
        }

        return true;
    }

    //对对应颜色方块的不同形态赋予 NowGrids、row_number、column_number
    //GridsColor、column_start 的值
    public void createRedGrids1() {
        NowGrids = RedGrids1;
        row_number = 2;
        column_number = 3;
        GridsColor = 1;
        column_start = 3;
    }
    …
}
```

添加一个名为 rightShift()的函数,以实现方块向右移动。在函数体内判断 right()函数的返回值,当返回值为 true 时,实现方块向右移动一格,并且 Nowrow 加 1。在函数 drawButton()内的按钮 button_right 中增加单击事件,调用函数 rightShift(),代码如下:

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }
    //初始化数据的函数
    public void initialize() {
        …
    }

    //随机重新生成一种颜色方块的函数
    public void createGrids() {
        …
    }

    //绘制按钮的函数
    public void drawButton() {
        //设置背景图层
        ShapeElement background = new ShapeElement();
```

```
background.setRgbColor(new RgbColor(120, 198, 197));
background.setCornerRadius(100);

Button button_left = new Button(this);                    //初始化按钮
button_left.setText("←");                                 //设置按钮的文本
//设置按钮文本的对齐方式
button_left.setTextAlignment(TextAlignment.CENTER);
button_left.setTextColor(Color.WHITE);                    //设置文本的颜色
button_left.setTextSize(100);                             //设置按钮文本的大小
button_left.setMarginTop(1800);                           //设置按钮的上外边距
button_left.setMarginLeft(160);                           //设置按钮的左外边距
button_left.setPadding(10, 0, 10, 0);                     //设置按钮的内边距
button_left.setBackground(background);                    //设置按钮的背景图层
//设置按钮的单击事件
button_left.setClickedListener(new Component.ClickedListener() {
    @Override
    public void onClick(Component component) {
        leftShift();
    }
});
layout.addComponent(button_left);

Button button_change = new Button(this);
button_change.setText("变");
button_change.setTextAlignment(TextAlignment.CENTER);
button_change.setTextColor(Color.WHITE);
button_change.setTextSize(100);
button_change.setMarginLeft(480);
button_change.setMarginTop(-135);
button_change.setPadding(10, 0, 10, 0);
button_change.setBackground(background);
layout.addComponent(button_change);

Button button_right = new Button(this);
button_right.setText("→");
button_right.setTextAlignment(TextAlignment.CENTER);
button_right.setTextColor(Color.WHITE);
button_right.setTextSize(100);
button_right.setMarginLeft(780);
button_right.setMarginTop(-135);
button_right.setPadding(10, 0, 10, 0);
button_right.setBackground(background);
//设置按钮的单击事件
button_right.setClickedListener(new Component.ClickedListener() {
    @Override
    public void onClick(Component component) {
```

```
                rightShift();
            }
        });
        layout.addComponent(button_right);

        Button button_start = new Button(this);
        button_start.setText("重新开始");
        button_start.setTextSize(100);
        button_start.setTextAlignment(TextAlignment.CENTER);
        button_start.setTextColor(Color.WHITE);
        button_start.setMarginTop(5);
        button_start.setMarginLeft(180);
        button_start.setPadding(10, 10, 10, 10);
        button_start.setBackground(background);
        layout.addComponent(button_start);

        Button button_back = new Button(this);
        button_back.setText("返回");
        button_back.setTextSize(100);
        button_back.setTextAlignment(TextAlignment.CENTER);
        button_back.setTextColor(Color.WHITE);
        button_back.setMarginTop(-150);
        button_back.setMarginLeft(680);
        button_back.setPadding(10, 10, 10, 10);
        button_back.setBackground(background);
        //设置按钮的单击事件
        button_back.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                //跳转到 MainAbilitySlice()语句
                present(new MainAbilitySlice(),new Intent());
            }
        });
        layout.addComponent(button_back);
    }

    //绘制网格的函数
    public void drawGrids() {
        …
    }

    //方块自动下落的函数
    public void run() {
        …
    }
```

```
    //判断方块能否下落的函数
    public boolean down() {
        …
    }
    //实现方块向左移动的函数

    public void leftShift() {
        …
    }

    //判断方块能否向左移动的函数
    public boolean left() {
        …
    }

    //实现方块向右移动的函数
    public void rightShift() {
        if (right()) {
            //将原来方块的颜色清除
            for (int row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] = 0;
            }
            Nowcolumn ++ ;
            //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
            for (int row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
            }
        }
        //重新绘制网格
        drawGrids();
}

    //判断方块能否向右移动的函数
    public boolean right() {
        …
    }
    …
}
```

进入游戏页面，当每次单击"→"按钮时，正在下落的方块会向右移动一格，如果正在下落的方块位于网格的右端或其右端存在其他方块，则不会再向右移动了，运行效果如图 5-42 和图 5-43 所示。

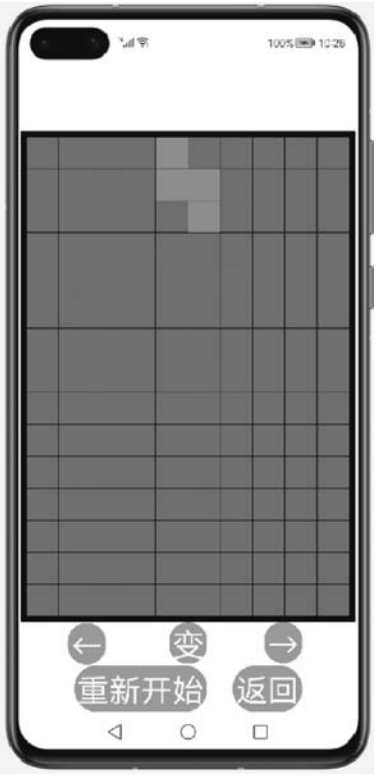

图 5-42　向右移动前

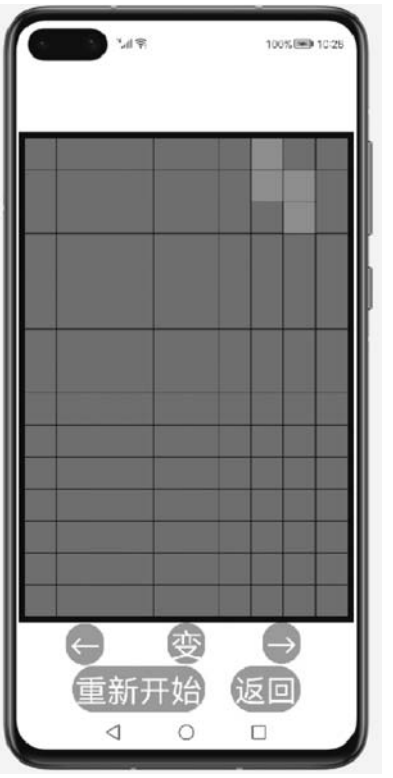

图 5-43　向右移动后

5.13　在游戏页面实现方块形态的改变

本节实现的运行效果：当单击“变”按钮时，正在下落的方块会循环改变一次形态。

本节的实现思路：通过对当前方块的颜色进行判断，在同一颜色的不同形态中依次改变 NowGrids、row_number、column_number、GridsColor、column_start 的值，实现方块形态的改变。

打开 ThirdAbilitySlice.java 文件。

(1) 添加一个名为 changRedGrids()的函数，对红色方块的形态进行循环改变。当 NowGrids 为 RedGrids1 时，调用函数 createRedGrids2()，当 NowGrids 为 RedGrids2 时，调用函数 createRedGrids1()。

(2) 添加一个名为 changeGreenGrids()的函数，对绿色方块的形态进行循环改变。当 NowGrids 为 GreenGrids1 时，调用函数 createGreenGrids2()，当 NowGrids 为 GreenGrids2 时，调用函数 createGreenGrids1()。

(3) 添加一个名为 changeCyanGrids()的函数，对蓝绿色方块的形态进行循环改变。当

NowGrids 为 CyanGrids1 时，调用函数 createCyanGrids2()，当 NowGrids 为 CyanGrids2 时，调用函数 createCyanGrids1()。

(4) 添加一个名为 changeMagentaGrids()的函数，对品红色方块的形态进行循环改变。当 NowGrids 为 MagentaGrids1 时，调用函数 createMagentaGrids2()，当 NowGrids 为 MagentaGrids2 时，调用函数 createMagentaGrids3()，当 NowGrids 为 MagentaGrids3 时，调用函数 createMagentaGrids4()，当 NowGrids 为 MagentaGrids4 时，调用函数 createMagentaGrids1()。

(5) 添加一个名为 changeBlueGrids()的函数，对蓝色方块的形态进行循环改变。当 NowGrids 为 BlueGrids1 时，调用函数 createBlueGrids2()，当 NowGrids 为 BlueGrids2 时，调用函数 createBlueGrids3()，当 NowGrids 为 BlueGrids3 时，调用函数 createBlueGrids4()，当 NowGrids 为 BlueGrids4 时，调用函数 createBlueGrids1()。

(6) 添加一个名为 changeWhiteGrids()的函数，对白色方块的形态进行循环改变。当 NowGrids 为 WhiteGrids1 时，调用函数 createWhiteGrids2()，当 NowGrids 为 WhiteGrids2 时，调用函数 createWhiteGrids3()，当 NowGrids 为 WhiteGrids3 时，调用函数 createWhiteGrids4()，当 NowGrids 为 WhiteGrids4 时，调用函数 createWhiteGrids1()。

注意不需要添加名为 changeYellowGrids()的函数对黄色方块的形态进行循环改变，因为黄色方块的形态只有一种，无法进行改变，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    ...
```

```
@Override
public void onStart(Intent intent) {
    ...
}

...

//判断方块能否向右移动的函数
public boolean right() {
   ...
}

//在同一种颜色的不同形态之间切换
public void changRedGrids() {
    if (NowGrids == RedGrids1) {
        createRedGrids2();
    } else if (NowGrids == RedGrids2) {
        createRedGrids1();
    }
}

public void changeGreenGrids() {
    if (NowGrids == GreenGrids1) {
        createGreenGrids2();
    } else if (NowGrids == GreenGrids2) {
        createGreenGrids1();
    }
}

public void changeCyanGrids() {
    if (NowGrids == CyanGrids1) {
        createCyanGrids2();
    } else if (NowGrids == CyanGrids2) {
        createCyanGrids1();
    }
}

public void changeMagentaGrids() {
    if (NowGrids == MagentaGrids1) {
        createMagentaGrids2();
    } else if (NowGrids == MagentaGrids2) {
        createMagentaGrids3();
    } else if (NowGrids == MagentaGrids3) {
        createMagentaGrids4();
    } else if (NowGrids == MagentaGrids4) {
        createMagentaGrids1();
```

```
        }
    }

    public void changeBlueGrids() {
        if (NowGrids == BlueGrids1) {
            createBlueGrids2();
        } else if (NowGrids == BlueGrids2) {
            createBlueGrids3();
        } else if (NowGrids == BlueGrids3) {
            createBlueGrids4();
        } else if (NowGrids == BlueGrids4) {
            createBlueGrids1();
        }
    }

    public void changeWhiteGrids() {
        if (NowGrids == WhiteGrids1) {
            createWhiteGrids2();
        } else if (NowGrids == WhiteGrids2) {
            createWhiteGrids3();
        } else if (NowGrids == WhiteGrids3) {
            createWhiteGrids4();
        } else if (NowGrids == WhiteGrids4) {
            createWhiteGrids1();
        }
    }

    //对对应颜色方块的不同形态赋予 NowGrids、row_number、column_number
    //GridsColor、column_start 的值
    public void createRedGrids1() {
        NowGrids = RedGrids1;
        row_number = 2;
        column_number = 3;
        GridsColor = 1;
        column_start = 3;
    }
    …
}
```

添加一个名为 change()的函数，以判断方块能否改变形态。当假设方块的形态改变后，方块形态改变后的位置如果超过网格的范围，则不能改变方块的形态，因此返回值为 false。方块形态改变后的位置如果存在其他方块，则不能改变方块的形态，因此返回值也为 false。如果不满足上述情况，则表示方块可以改变形态，因此返回值为 true，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
}

…

    //判断方块能否向右移动的函数
    public boolean right() {
       …
    }

    //判断方块能否改变形态的函数
    private boolean change(){
        for (int row = 0; row < grids_number; row ++ ) {
            if (grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] != 0) {
                return false;
            }

            if (NowGrids[row][0] + Nowrow < 0 ||NowGrids[row][0] + Nowrow >= height ||
NowGrids[row][1] + Nowcolumn < 0 ||NowGrids[row][1] + Nowcolumn >= width) {
                return false;
            }
```

```
            }

            return true;
        }

        //在同一种颜色的不同形态之间切换
        public void changRedGrids() {
            if (NowGrids == RedGrids1) {
                createRedGrids2();
            } else if (NowGrids == RedGrids2) {
                createRedGrids1();
            }
        }
        …
    }
```

添加一个名为 changeGrids()的函数，以实现方块改变形态。假设方块可以改变形态，消除当前的方块，定义一个局部变量 Grids 并初始化为 NowGrids，根据 GridsColor 的数值调用函数 changRedGrids()、changeGreenGrids()、changeCyanGrids()、changeMagentaGrids()、changeBlueGrids()或 changeWhiteGrids()。判断 change()函数的返回值，当返回值为 true 时，实现方块改变形态；当返回值为 false 时，不改变方块的形态，重新绘制没改变前方块的形态，调用函数 drawGrids()。在函数 drawButton()内的按钮 button_change 增加单击事件，调用函数 changeGrids()，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;
```

```
public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }

    //随机重新生成一种颜色方块的函数
    public void createGrids() {
        …
    }
    //绘制按钮的函数
    public void drawButton() {
        //设置背景图层
        ShapeElement background = new ShapeElement();
        background.setRgbColor(new RgbColor(120, 198, 197));
        background.setCornerRadius(100);

        Button button_left = new Button(this);                    //初始化按钮
        button_left.setText("←");                                 //设置按钮的文本
        //设置按钮文本的对齐方式
        button_left.setTextAlignment(TextAlignment.CENTER);
        button_left.setTextColor(Color.WHITE);                    //设置文本的颜色
        button_left.setTextSize(100);                             //设置按钮文本的大小
        button_left.setMarginTop(1800);                           //设置按钮的上外边距
        button_left.setMarginLeft(160);                           //设置按钮的左外边距
        button_left.setPadding(10, 0, 10, 0);                     //设置按钮的内边距
        button_left.setBackground(background);                    //设置按钮的背景图层
        //设置按钮的单击事件
        button_left.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                leftShift();
            }
        });
        layout.addComponent(button_left);

        Button button_change = new Button(this);
        button_change.setText("变");
        button_change.setTextAlignment(TextAlignment.CENTER);
        button_change.setTextColor(Color.WHITE);
```

```
button_change.setTextSize(100);
button_change.setMarginLeft(480);
button_change.setMarginTop(-135);
button_change.setPadding(10, 0, 10, 0);
button_change.setBackground(background);
//设置按钮的单击事件
button_change.setClickedListener(new Component.ClickedListener() {
    @Override
    public void onClick(Component component) {
        changGrids();
    }
});
layout.addComponent(button_change);

Button button_right = new Button(this);
button_right.setText("→");
button_right.setTextAlignment(TextAlignment.CENTER);
button_right.setTextColor(Color.WHITE);
button_right.setTextSize(100);
button_right.setMarginLeft(780);
button_right.setMarginTop(-135);
button_right.setPadding(10, 0, 10, 0);
button_right.setBackground(background);
//设置按钮的单击事件
button_right.setClickedListener(new Component.ClickedListener() {
    @Override
    public void onClick(Component component) {
        rightShift();
    }
});
layout.addComponent(button_right);

Button button_start = new Button(this);
button_start.setText("重新开始");
button_start.setTextSize(100);
button_start.setTextAlignment(TextAlignment.CENTER);
button_start.setTextColor(Color.WHITE);
button_start.setMarginTop(5);
button_start.setMarginLeft(180);
button_start.setPadding(10, 10, 10, 10);
button_start.setBackground(background);
layout.addComponent(button_start);
```

```
        Button button_back = new Button(this);
        button_back.setText("返回");
        button_back.setTextSize(100);
        button_back.setTextAlignment(TextAlignment.CENTER);
        button_back.setTextColor(Color.WHITE);
        button_back.setMarginTop(-150);
        button_back.setMarginLeft(680);
        button_back.setPadding(10, 10, 10, 10);
        button_back.setBackground(background);
        //设置按钮的单击事件
        button_back.setClickedListener(new Component.ClickedListener() {
            @Override
            public void onClick(Component component) {
                //跳转到 MainAbilitySlice()语句
                present(new MainAbilitySlice(),new Intent());
            }
        });
        layout.addComponent(button_back);
    }

    //绘制网格的函数
    public void drawGrids() {
        …
    }

…

    //判断方块能否向右移动的函数
    public boolean right() {
        …
    }

    //实现方块改变形态的函数
    public void changeGrids() {
        int[][] Grids = NowGrids; //定义一个二维数组,用来存放改变形态前的方块形态

        for (int row = 0; row < grids_number; row ++ ) {
            //将原来方块的颜色清除
            grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] = 0;
        }
        if (column_number == 2 && Nowcolumn + column_start == 0) {
            Nowcolumn ++ ;
        }

        if (GridsColor == 1) {
            changRedGrids();
```

```
            } else if (GridsColor == 2) {
                changeGreenGrids();
            } else if (GridsColor == 3) {
                changeCyanGrids();
            } else if (GridsColor == 4) {
                changeMagentaGrids();
            } else if (GridsColor == 5) {
                changeBlueGrids();
            } else if (GridsColor == 6) {
                changeWhiteGrids();
            }

            if(change()){
                //如果能够改变形态,则将行的颜色方块添加到 15×10 网格的二维数组 grids 中
                for (int row = 0; row < grids_number; row ++ ) {
                    grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
                }
            }else{
                //如果不能改变形态,则将原来的颜色方块添加到 15×10 网格的二维数组 grids 中
                NowGrids = Grids;
                for (int row = 0; row < grids_number; row ++ ) {
                    grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
                }
            }

            //重新绘制网格
            drawGrids();
        }

        //判断方块能否改变形态的函数
        private boolean change(){
            …
        }
        …
    }
```

进入游戏页面,当每次单击"变"按钮时,正在下落的方块会改变形态并向右移动一格,运行效果如图 5-44 和图 5-45 所示。

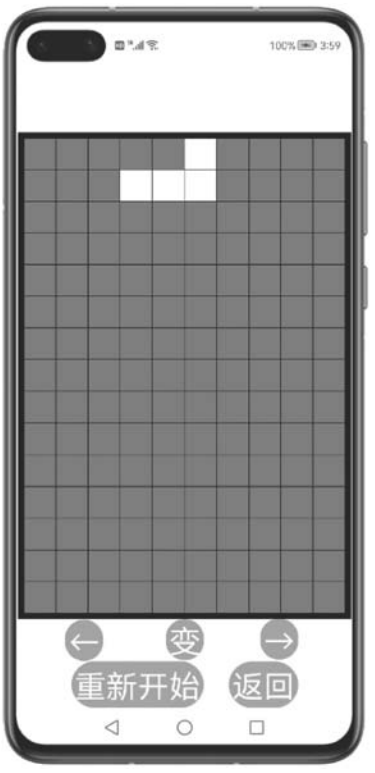

图 5-44　单击“变”按钮前

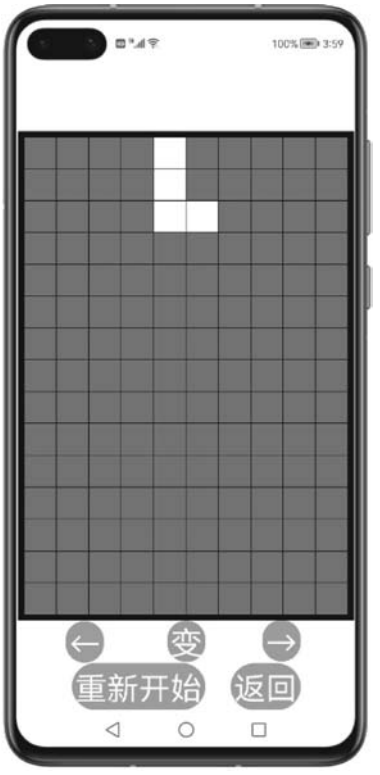

图 5-45　单击“变”按钮后

5.14　在游戏页面实现整行相同色彩方格的消除

本节实现的运行效果：当网格中存在整行相同色彩方格时，该行方格被消除，该行上方所有方格整体向下移动一格。

本节的实现思路：在生成方块之前，从网格的下方往上查找是否存在整行相同色彩方格的行，如果存在则消除该行方格，并且该行上方所有方格整体向下移动。

打开 ThirdAbilitySlice.java 文件。

添加一个名为 eliminateGrids()的函数，以实现整行相同色彩方格的消除。从网格的下方往上查找是否存在整行相同色彩方格的行，以局部变量 k 表示，当 k 为 false 时表示不存在整行相同色彩方格的行。若 k 为 true 时则消除该行方格，并且该行上方所有方格整体向下移动。调用函数 drawGrids()，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;
```

```
import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }

    …

    //判断方块能否改变形态的函数
    private boolean change(){
        …
    }

    //实现消除整行方格的函数
    public void eliminateGrids() {
        boolean k;
        //从下往上循环判断每一行是否已经满了
        for (int row = height - 1; row >= 0; row -- ) {
            k = true;
            //如果该行存在一个或一个以上灰色的方格,则说明该行没有满足条件
            for (int column = 0; column < width; column ++ ) {
                if (grids[row][column] == 0)
                    k = false;
            }
            if (k) {
                //将该行上面的所有行整体向下移动一格
                for (int i = row - 1; i >= 0; i -- ) {
```

```
                    for (int j = 0; j < width; j ++ ) {
                        grids[i + 1][j] = grids[i][j];
                    }
                }
                for (int n = 0; n < width; n ++ ) {
                    grids[0][n] = 0;
                }
                //再次判断该行是否满足消除的条件
                row ++ ;
            }
        }
        //重新绘制网格
        drawGrids();
    }

    //在同一种颜色的不同形态之间切换
    public void changRedGrids() {
        if (NowGrids == RedGrids1) {
            createRedGrids2();
        } else if (NowGrids == RedGrids2) {
            createRedGrids1();
        }
    }
    …
}
```

在函数 createGrids()内随机生成方块前调用函数 eliminateGrids(),代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;
```

```
import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }

    //初始化数据的函数
    public void initialize() {
        …
    }

    //随机重新生成一种颜色方块的函数
    public void createGrids() {
        Nowrow = 0;
        Nowcolumn = 0;

        eliminateGrids();

        double random = Math.random(); //生成[0,1)的随机数
        //根据随机数的大小,调用相关的函数
        if (random >= 0 && random < 0.2) {
            if (random >= 0 && random < 0.1)
                createRedGrids1();
            else
                createRedGrids2();
        } else if (random >= 0.2 && random < 0.4) {
            if (random >= 0.2 && random < 0.3)
                createGreenGrids1();
            else
                createGreenGrids2();
        } else if (random >= 0.4 && random < 0.45) {
            if (random >= 0.4 && random < 0.43)
                createCyanGrids1();
            else
                createCyanGrids2();
        } else if (random >= 0.45 && random < 0.6) {
            if (random >= 0.45 && random < 0.48)
                createMagentaGrids1();
            else if (random >= 0.48 && random < 0.52)
                createMagentaGrids2();
            else if (random >= 0.52 && random < 0.56)
```

```
                createMagentaGrids3();
            else
                createMagentaGrids4();
        } else if (random >= 0.6 && random < 0.75) {
            if (random >= 0.6 && random < 0.63)
                createBlueGrids1();
            else if (random >= 0.63 && random < 0.67)
                createBlueGrids2();
            else if (random >= 0.67 && random < 0.71)
                createBlueGrids3();
            else
                createBlueGrids4();
        } else if (random >= 0.75 && random < 0.9) {
            if (random >= 0.75 && random < 0.78)
                createWhiteGrids1();
            else if (random >= 0.78 && random < 0.82)
                createWhiteGrids2();
            else if (random >= 0.82 && random < 0.86)
                createWhiteGrids3();
            else
                createWhiteGrids4();
        } else {
            createYellowGrids();
        }

        //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
        for (int row = 0; row < grids_number; row ++ ) {
            grids[NowGrids[row][0]][NowGrids[row][1]] = GridsColor;
        }
    }

    //绘制按钮的函数
    public void drawButton() {
        …
    }
    …
}
```

进入游戏页面，当网格中存在整行相同色彩方格时，该行方格被消除，该行上方所有方格整体向下移动，运行效果如图 5-46 和图 5-47 所示。

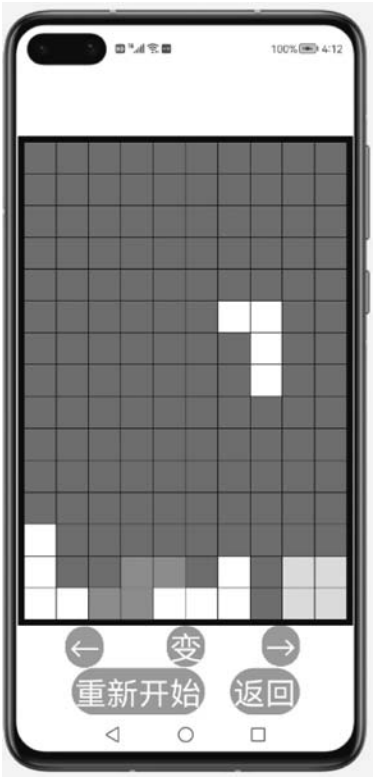

图 5-46　方块消除前

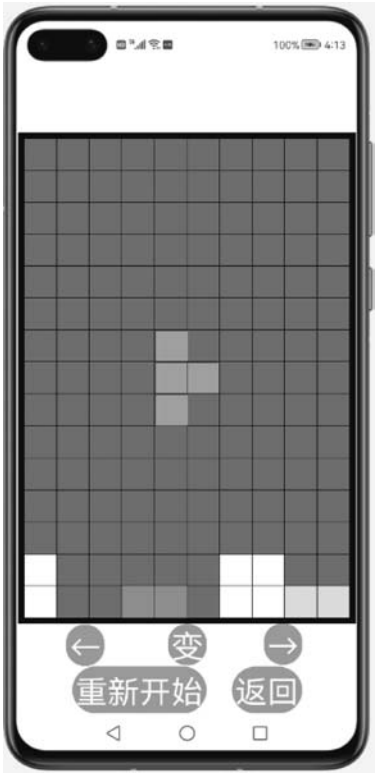

图 5-47　方块消除后

5.15　在游戏页面显示游戏结束的文本

本节实现的运行效果：当网格中无法生成新的方块时，将会在网格的上方显示"游戏结束"的文本。

本节的实现思路：先判断生成方块的位置是否已存在其他方块，若已存在，则表示游戏结束，在网格的上方显示"游戏结束"文本。

打开 ThirdAbilitySlice.java 文件。

添加一个名为 gameover()的函数，以实现判断生成方块的位置是否已存在其他方块。当生成方块的任一方格的数值不为 0 时，说明生成方块的位置存在其他方块，因此返回值为 false，否则返回值为 true，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;
```

```
import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    ...

    @Override
    public void onStart(Intent intent) {
        ...
    }

    ...

    //判断方块能否改变形态的函数
    private boolean change(){
        ...
    }

    //判断游戏是否结束
    public boolean gameover() {
        //当生成方块的任一方格的数值不为 0 时,说明游戏结束
        for (int row = 0; row < grids_number; row ++ ) {
            if (grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] != 0) {
                return false;
            }
        }
        return true;
    }

    //实现消除整行方格的函数
    public void eliminateGrids() {
        ...
    }
    ...
}
```

添加一个名为 drawText()的函数，以实现绘制"游戏结束"的文本。

添加一个文本 text，将文本 setText 设置为游戏结束，将属性 setTextSize(文本的大小)设置为 100，将属性 setTextColor(文本的颜色)设置为 Color. BLUE(蓝色)，将属性 setTextAlignment(文本的对齐方式)设置为 TextAlignment. CENTER(居中)。将属性 setMarginsTopAndBottom(文本的上下外边距)设置为(－2000，0)，将属性 setMarginsLeftAndRight(文本的左右外边距)设置为(350，0)。最后将设置好样式的文本添加到布局 layout 中，设置 UI 布局，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;
import ohos.agp.components.Text;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    ...

    @Override
    public void drawGrids() {
        ...
    }

    ...

    //绘制网格的函数
    public void drawGrids() {
     ...
    }
```

```
    //绘制"游戏结束"文本
    public void drawText(){
        Text text = new Text(this);                          //初始化文本
        text.setText("游戏结束");                             //设置文本
        text.setTextSize(100);                               //设置文本的大小
        text.setTextColor(Color.BLUE);                       //设置文本的颜色
        text.setTextAlignment(TextAlignment.CENTER);         //设置文本的对齐方式
        text.setMarginsTopAndBottom(-2000, 0);               //设置文本的上下外边距
        text.setMarginsLeftAndRight(350, 0);                 //设置文本的左右外边距
        layout.addComponent(text);
        setUIContent(layout);
    }

    //方块自动下落的函数
    public void run() {
        …
    }
    …
}
```

定义一个 boolean 类型的全局变量 Gameover。在函数体 initialize()中将变量 Gameover 初始化为 true。在函数体 createGrids()内,在绘制生成新的方块之前,判断 gameover()函数的返回值,当返回值为 true 时,绘制生成新的方块。当返回值为 false 时,停止时间的流逝,不再实现方块的下落,调用函数 drawText(),以便显示“游戏结束”文本,将变量 Gameover 赋值为 false,以表示游戏已经结束了,代码如下:

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;
import ohos.agp.components.Text;
```

```
import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    private DirectionalLayout layout;                    //自定义定向布局
    private static final int length = 100;               //网格中方格的边长
    private static final int interval = 2;               //网格中方格的间距
    private static final int height = 15;                //网格中竖列方格的数量
    private static final int width = 10;                 //网格中横列方格的数量
    private static final int left = 30;                  //网格的左端距手机边界的距离
    private static final int top = 250;                  //网格的顶端距手机边界的距离
    private static final int margin = 20;                //网格的外围距离
    private int[][] grids;                               //15×10 网格的二维数组
    private int[][] NowGrids;                            //当前方块形态的二维数组
    private int row_number;                              //当前方块的总行数
    private int column_number;                           //当前方块的总列数
    private int column_start;                            //当前方块所在 grids 的列数
    //当前方块的颜色,0 表示灰色,1 代表红色,2 代表绿色,3 代表蓝绿色
    //4 代表品红色,5 代表蓝色,6 代表白色,7 代表黄色
    private int GridsColor;
    //19 种方块所占网格的位置所对应的数值
    private static final int[][] RedGrids1 = {{0, 3},{0, 4},{1, 4},{1, 5}};
    private static final int[][] RedGrids2 = {{0, 5},{1, 5},{1, 4},{2, 4}};
    private static final int[][] GreenGrids1 = {{0, 5},{0, 4},{1, 4},{1, 3}};
    private static final int[][] GreenGrids2 = {{0, 4},{1, 4},{1, 5},{2, 5}};
    private static final int[][] CyanGrids1 = {{0, 4},{1, 4},{2, 4},{3, 4}};
    private static final int[][] CyanGrids2 = {{0, 3},{0, 4},{0, 5},{0, 6}};
    private static final int[][] MagentaGrids1 = {{0,4},{1, 3},{1, 4},{1, 5}};
    private static final int[][] MagentaGrids2 = {{0,4},{1, 4},{1, 5},{2, 4}};
    private static final int[][] MagentaGrids3 = {{0,3},{0, 4},{0, 5},{1, 4}};
    private static final int[][] MagentaGrids4 = {{0,5},{1, 5},{1, 4},{2, 5}};
    private static final int[][] BlueGrids1 = {{0, 3},{1, 3},{1, 4},{1, 5}};
    private static final int[][] BlueGrids2 = {{0, 5},{0, 4},{1, 4},{2, 4}};
    private static final int[][] BlueGrids3 = {{0, 3},{0, 4},{0, 5},{1, 5}};
    private static final int[][] BlueGrids4 = {{0, 5},{1, 5},{2, 5},{2, 4}};
    private static final int[][] WhiteGrids1 = {{0, 5},{1, 5},{1, 4},{1, 3}};
    private static final int[][] WhiteGrids2 = {{0, 4},{1, 4},{2, 4},{2, 5}};
    private static final int[][] WhiteGrids3 = {{0, 5},{0, 4},{0, 3},{1, 3}};
    private static final int[][] WhiteGrids4 = {{0, 4},{0, 5},{1, 5},{2, 5}};
    private static final int[][] YellowGrids = {{0, 4},{0, 5},{1, 5},{1, 4}};
    private static final int grids_number = 4;  //方块的方格数量
    private int Nowrow;                                  //方块下落移动的行数
    private int Nowcolumn;                               //方块左右移动的列数
    private Timer timer;                                 //时间变量
    private boolean Gameover;                            //表示游戏是否结束
```

```
@Override
public void onStart(Intent intent) {
    ...
}

//初始化数据的函数
public void initialize() {
    layout = new DirectionalLayout(this); //对定向布局 layout 初始化
    Gameover = true;
    //将二维数组 grids 初始化为 0
    grids = new int[height][width];
    for (int row = 0; row < height; row ++ )
        for (int column = 0; column < width; column ++ )
            grids[row][column] = 0;

    createGrids();
    drawButton();
    drawGrids();
}

//随机重新生成一种颜色方块的函数
public void createGrids() {
    Nowrow = 0;
    Nowcolumn = 0;

    eliminateGrids();

    double random = Math.random(); //生成[0,1)的随机数
    //根据随机数的大小,调用相关的函数
    if (random >= 0 && random < 0.2) {
        if (random >= 0 && random < 0.1)
            createRedGrids1();
        else
            createRedGrids2();
    } else if (random >= 0.2 && random < 0.4) {
        if (random >= 0.2 && random < 0.3)
            createGreenGrids1();
        else
            createGreenGrids2();
    } else if (random >= 0.4 && random < 0.45) {
        if (random >= 0.4 && random < 0.43)
            createCyanGrids1();
        else
            createCyanGrids2();
    } else if (random >= 0.45 && random < 0.6) {
        if (random >= 0.45 && random < 0.48)
```

```
                createMagentaGrids1();
            else if (random >= 0.48 && random < 0.52)
                createMagentaGrids2();
            else if (random >= 0.52 && random < 0.56)
                createMagentaGrids3();
            else
                createMagentaGrids4();
        } else if (random >= 0.6 && random < 0.75) {
            if (random >= 0.6 && random < 0.63)
                createBlueGrids1();
            else if (random >= 0.63 && random < 0.67)
                createBlueGrids2();
            else if (random >= 0.67 && random < 0.71)
                createBlueGrids3();
            else
                createBlueGrids4();
        } else if (random >= 0.75 && random < 0.9) {
            if (random >= 0.75 && random < 0.78)
                createWhiteGrids1();
            else if (random >= 0.78 && random < 0.82)
                createWhiteGrids2();
            else if (random >= 0.82 && random < 0.86)
                createWhiteGrids3();
            else
                createWhiteGrids4();
        } else {
            createYellowGrids();
        }

        for (int row = 0; row < grids_number; row ++ ) {
            grids[NowGrids[row][0]][NowGrids[row][1]] = GridsColor;
        }
        //判断游戏是否已结束
        if (gameover()) {
            //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
            for (int row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
            }
        } else {
            timer.cancel();             //游戏结束时,停止时间任务
            drawText();                 //绘制"游戏结束"文本
            Gameover = false;
        }
    }
```

```
    //绘制按钮的函数
    public void drawButton() {
        …
    }
    …
}
```

为了使当游戏结束时,“←”“变”“→”按钮不再响应单击事件,即游戏结束时单击这 3 个按钮网格不再发生变化,需要对这 3 个按钮的单击事件添加游戏是否结束的判断。当游戏结束时,按钮不再响应单击事件。在函数体 drawButton()内,对 button_left、button_change 和 button_right 的单击事件判断 Gameover 的值。当 Gameover 的值为 true 时,执行单击事件,代码如下:

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;
import ohos.agp.components.Text;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }
    //初始化数据的函数
    public void initialize() {
        …
    }
```

```
//随机重新生成一种颜色方块的函数
public void createGrids() {
    …
}

//绘制按钮的函数
public void drawButton() {
    //设置背景图层
    ShapeElement background = new ShapeElement();
    background.setRgbColor(new RgbColor(120, 198, 197));
    background.setCornerRadius(100);

    Button button_left = new Button(this);                    //初始化按钮
    button_left.setText("←");                                 //设置按钮的文本
    //设置按钮文本的对齐方式
    button_left.setTextAlignment(TextAlignment.CENTER);
    button_left.setTextColor(Color.WHITE);                    //设置文本的颜色
    button_left.setTextSize(100);                             //设置按钮文本的大小
    button_left.setMarginTop(1800);                           //设置按钮的上外边距
    button_left.setMarginLeft(160);                           //设置按钮的左外边距
    button_left.setPadding(10, 0, 10, 0);                     //设置按钮的内边距
    button_left.setBackground(background);                    //设置按钮的背景图层
    //设置按钮的单击事件
    button_left.setClickedListener(new Component.ClickedListener() {
        @Override
        public void onClick(Component component) {
            leftShift();
            if(Gameover){
                leftShift();
            }
        }
    });
    layout.addComponent(button_left);

    Button button_change = new Button(this);
    button_change.setText("变");
    button_change.setTextAlignment(TextAlignment.CENTER);
    button_change.setTextColor(Color.WHITE);
    button_change.setTextSize(100);
    button_change.setMarginLeft(480);
    button_change.setMarginTop(-135);
    button_change.setPadding(10, 0, 10, 0);
    button_change.setBackground(background);
    //设置按钮的单击事件
    button_change.setClickedListener(new Component.ClickedListener() {
        @Override
```

```
        public void onClick(Component component) {
            changGrids();
            if(Gameover){
               changGrids();
            }
        }
    });
    layout.addComponent(button_change);

    Button button_right = new Button(this);
    button_right.setText("→");
    button_right.setTextAlignment(TextAlignment.CENTER);
    button_right.setTextColor(Color.WHITE);
    button_right.setTextSize(100);
    button_right.setMarginLeft(780);
    button_right.setMarginTop(-135);
    button_right.setPadding(10, 0, 10, 0);
    button_right.setBackground(background);
    //设置按钮的单击事件
    button_right.setClickedListener(new Component.ClickedListener() {
        @Override
        public void onClick(Component component) {
            rightShift();
            if(Gameover){
               rightShift();
            }
        }
    });
    layout.addComponent(button_right);

    Button button_start = new Button(this);
    button_start.setText("重新开始");
    button_start.setTextSize(100);
    button_start.setTextAlignment(TextAlignment.CENTER);
    button_start.setTextColor(Color.WHITE);
    button_start.setMarginTop(5);
    button_start.setMarginLeft(180);
    button_start.setPadding(10, 10, 10, 10);
    button_start.setBackground(background);
    layout.addComponent(button_start);

    Button button_back = new Button(this);
    button_back.setText("返回");
    button_back.setTextSize(100);
    button_back.setTextAlignment(TextAlignment.CENTER);
    button_back.setTextColor(Color.WHITE);
    button_back.setMarginTop(-150);
    button_back.setMarginLeft(680);
    button_back.setPadding(10, 10, 10, 10);
```

```
            button_back.setBackground(background);
            //设置按钮的单击事件
            button_back.setClickedListener(new Component.ClickedListener() {
                @Override
                public void onClick(Component component) {
                    //跳转到 MainAbilitySlice()语句
                    present(new MainAbilitySlice(),new Intent());
                }
            });
            layout.addComponent(button_back);
        }

        //绘制网格的函数
        public void drawGrids() {
            …
        }
        …
    }
```

进入游戏页面，当网格中无法生成新的方块时，将会在网格的上方显示"游戏结束"文本。这时再单击"←""变""→"按钮时，网格不再发生变化，运行效果如图 5-48 所示。

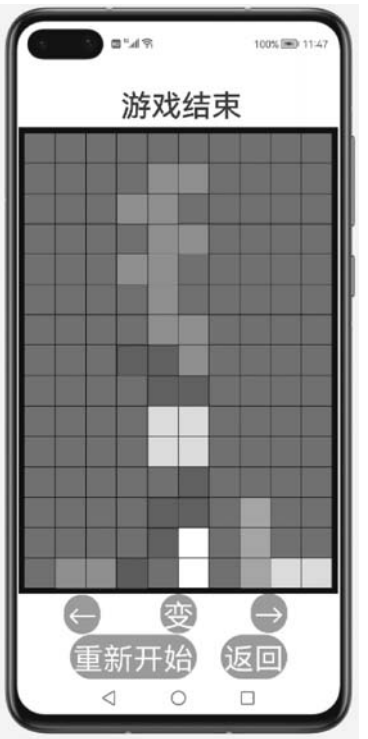

图 5-48　游戏结束页面

5.16 在游戏页面实现游戏重新开始功能

本节实现的运行效果：当单击“重新开始”按钮时，网格中的所有方块便会被清空，游戏将会重新开始。

本节的实现思路：添加重新开始的单击事件，对所有变量进行初始化。

打开 ThirdAbilitySlice.java 文件。

在函数体 drawButton()内，对 button_start 添加单击事件，调用函数 initialize()对变量进行初始化，停止时间的流逝，再调用函数 run()。这里先停止时间流逝再调用函数 run()是因为该单击事件可能是在游戏未结束时响应，如果没有先停止时间流逝再调用函数 run()，则方块的下落速度将会翻倍，代码如下：

```
//第 5 章 ThirdAbilitySlice.java
package com.test.game.slice;

import ohos.aafwk.ability.AbilitySlice;
import ohos.agp.components.DirectionalLayout;
import ohos.aafwk.content.Intent;
import ohos.agp.components.Component;
import ohos.agp.components.ComponentContainer;
import ohos.agp.render.Canvas;
import ohos.agp.render.Paint;
import ohos.agp.utils.Color;
import ohos.agp.utils.RectFloat;
import ohos.agp.utils.TextAlignment;
import ohos.agp.components.Button;
import ohos.agp.components.element.ShapeElement;
import ohos.agp.colors.RgbColor;
import ohos.agp.components.Text;

import java.util.Timer;
import java.util.TimerTask;

public class ThirdAbilitySlice extends AbilitySlice {
    …

    @Override
    public void onStart(Intent intent) {
        …
    }

    //初始化数据的函数
    public void initialize() {
        …
    }
```

```
//随机重新生成一种颜色方块的函数
public void createGrids() {
    …
}

//绘制按钮的函数
public void drawButton() {
    //设置背景图层
    ShapeElement background = new ShapeElement();
    background.setRgbColor(new RgbColor(120, 198, 197));
    background.setCornerRadius(100);

    Button button_left = new Button(this);                    //初始化按钮
    button_left.setText("←");                                 //设置按钮的文本
    //设置按钮文本的对齐方式
    button_left.setTextAlignment(TextAlignment.CENTER);
    button_left.setTextColor(Color.WHITE);                    //设置文本的颜色
    button_left.setTextSize(100);                             //设置按钮文本的大小
    button_left.setMarginTop(1800);                           //设置按钮的上外边距
    button_left.setMarginLeft(160);                           //设置按钮的左外边距
    button_left.setPadding(10, 0, 10, 0);                     //设置按钮的内边距
    button_left.setBackground(background);                    //设置按钮的背景图层
    //设置按钮的单击事件
    button_left.setClickedListener(new Component.ClickedListener() {
        @Override
        public void onClick(Component component) {
            if(Gameover){
               leftShift();
            }
        }
    });
    layout.addComponent(button_left);

    Button button_change = new Button(this);
    button_change.setText("变");
    button_change.setTextAlignment(TextAlignment.CENTER);
    button_change.setTextColor(Color.WHITE);
    button_change.setTextSize(100);
    button_change.setMarginLeft(480);
    button_change.setMarginTop(-135);
    button_change.setPadding(10, 0, 10, 0);
    button_change.setBackground(background);
    //设置按钮的单击事件
    button_change.setClickedListener(new Component.ClickedListener() {
        @Override
        public void onClick(Component component) {
            if(Gameover){
               changGrids();
            }
```

```
        }
    });
    layout.addComponent(button_change);

    Button button_right = new Button(this);
    button_right.setText("→");
    button_right.setTextAlignment(TextAlignment.CENTER);
    button_right.setTextColor(Color.WHITE);
    button_right.setTextSize(100);
    button_right.setMarginLeft(780);
    button_right.setMarginTop(-135);
    button_right.setPadding(10, 0, 10, 0);
    button_right.setBackground(background);
    //设置按钮的单击事件
    button_right.setClickedListener(new Component.ClickedListener() {
        @Override
        public void onClick(Component component) {
            if(Gameover){
               rightShift();
            }
        }
    });
    layout.addComponent(button_right);

    Button button_start = new Button(this);
    button_start.setText("重新开始");
    button_start.setTextSize(100);
    button_start.setTextAlignment(TextAlignment.CENTER);
    button_start.setTextColor(Color.WHITE);
    button_start.setMarginTop(5);
    button_start.setMarginLeft(180);
    button_start.setPadding(10, 10, 10, 10);
    button_start.setBackground(background);
    //设置按钮的单击事件
    button_start.setClickedListener(new Component.ClickedListener() {
        @Override
        public void onClick(Component component) {
            initialize();          //对数据进行初始化
            timer.cancel();        //停止时间任务
            run();                 //执行时间任务
        }
    });
    layout.addComponent(button_start);

    Button button_back = new Button(this);
    button_back.setText("返回");
    button_back.setTextSize(100);
    button_back.setTextAlignment(TextAlignment.CENTER);
    button_back.setTextColor(Color.WHITE);
    button_back.setMarginTop(-150);
```

```
            button_back.setMarginLeft(680);
            button_back.setPadding(10, 10, 10, 10);
            button_back.setBackground(background);
            //设置按钮的单击事件
            button_back.setClickedListener(new Component.ClickedListener() {
                @Override
                public void onClick(Component component) {
                    //跳转到 MainAbilitySlice()语句
                    present(new MainAbilitySlice(),new Intent());
                }
            });
            layout.addComponent(button_back);
        }
        public void drawGrids() {
            …
    }
    …
```

进入游戏页面，无论游戏是否已经结束，当单击"重新开始"按钮时，网格中的所有方块便会被清空，游戏将会重新开始，运行效果如图 5-49 和图 5-50 所示。

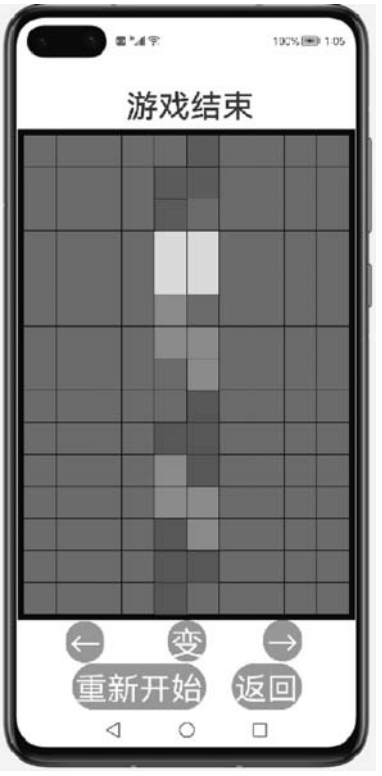

图 5-49　重新开始前

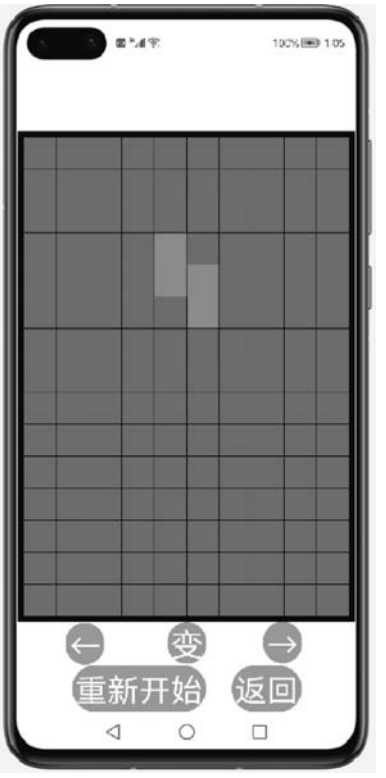

图 5-50　重新开始后

至此，在智能手机上用 Java 实现了“俄罗斯方块”App 的全部功能！

5.17 JavaScript 与 Java 的对比

在第 4 章完成了用 JavaScript 开发并且运行在智能手机上的经典游戏 App——“数字华容道”，在本章的前 16 节完成了用 Java 开发并且运行在智能手机上的经典游戏 App——“俄罗斯方块”。

接下来给出用 JavaScript 开发经典游戏 App——“俄罗斯方块”的代码，并且对 JavaScript 与 Java 这两种编程语言开发进行简单的对比。

创建一个名为 Game_JS 的 Hello World 项目，其余代码文件如图 5-51 所示。

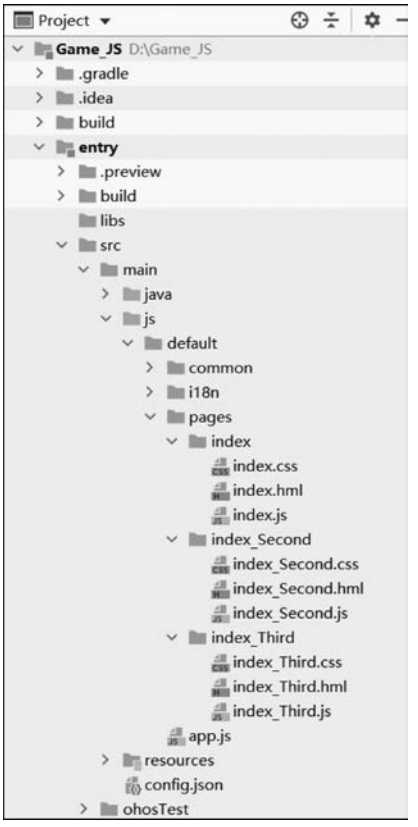

图 5-51 JavaScript 代码文件

JavaScript 代码文件中的代码如下：

```
第 5 章 config.json
…
```

```
        "launchType": "standard",
        "metaData": {
          "customizeData": [
            {
              "name": "hwc-theme",
              "value": "androidhwext:style/Theme.Emui.Light.NoTitleBar",
              "extra": ""
            }
          ]
        }
...
```

```
<!-- 第 5 章 index.hml -->
<div class="container">
    <!-- 添加文本为"开始"的按钮组件 -->
    <input type="button" value="开始" class="btn_game"
           onclick="clickAction_game" />
    <!-- 添加文本为"关于"的按钮组件 -->
    <input type="button" value="关于" class="btn_author"
           onclick="clickAction_author" />
</div>
```

```
/* 第 5 章 index.css */
.container {
    flex-direction: column;
    justify-content: center;
    align-items: center;
    background-color: #EFE5D3;
}

/* 文本为"开始"的按钮样式 */
.btn_game {
    height: 50px;
    width: 100%;
    font-size: 25px;
    text-color: #FFFFFF;
    text-align: center;
    radius: 100px;
    background-color: #78C6C5;
}

/* 文本为"关于"的按钮样式 */
.btn_author {
    height: 50px;
    width: 100%;
    margin-top: 30px;
```

```
    font-size:25px;
    text-color: #FFFFFF;
    text-align: center;
    radius: 100px;
    background-color:#78C6C5;
}

@media screen and (device-type: tablet) and (orientation: landscape) {
    .title {
        font-size: 100px;
    }
}

@media screen and (device-type: wearable) {
    .title {
        font-size: 28px;
        color: #FFFFFF;
    }
}

@media screen and (device-type: tv) {
    .container {
        background-image: url("../../common/images/Wallpaper.png");
        background-size: cover;
        background-repeat: no-repeat;
        background-position: center;
    }

    .title {
        font-size: 100px;
        color: #FFFFFF;
    }
}

@media screen and (device-type: phone) and (orientation: landscape) {
    .title {
        font-size: 60px;
    }
}
```

```
//第 5 章 index.js
import router from '@system.router';

export default {
    data: {
```

```
    },
    //文本为"开始"的按钮的单击事件
    clickAction_game(){
        //页面跳转语句
        router.replace({
            uri:'pages/index_Third/index_Third'
        })
    },
    //文本为"关于"的按钮的单击事件
    clickAction_author(){
        //页面跳转语句
        router.replace({
            uri:'pages/index_Second/index_Second'
        })
    }
}
```

```
<!-- 第 5 章 index_Second.hml -->
<div class = "container">
    <!-- 文本为"程序:俄罗斯方块"的文本组件 -->
    <text class = "title">
        程序:俄罗斯方块
    </text>
    <!-- 文本为"作者:张诏添"的文本组件 -->
    <text class = "title">
        作者:张诏添
    </text>
    <!-- 文本为"版本:v1.1.0"的文本组件 -->
    <text class = "title">
        版本:v1.1.0
    </text>
    <!-- 添加文本为"返回"的按钮组件 -->
    <input type = "button" value = "返回" class = "btn_back"
           onclick = "clickAction_back" />
</div>
```

```
/* 第 5 章 index_Second.css */
.container {
    flex-direction: column;
    background-color: #EFE5D3;
}

/* 文本的样式 */
.title {
    font-size: 25px;
```

```
    text-color: #000000;
    margin-top: 20px;
    margin-left: 5px;
}

/*文本为"返回"的按钮样式*/
.btn_back {
    height: 50px;
    width: 100%;
    margin-top: 30px;
    font-size:25px;
    text-color: #FFFFFF;
    text-align: center;
    radius: 100px;
    background-color: #78C6C5;
}
```

```
//第5章 index_Second.js
import router from '@system.router';

export default {
    data: {

    },
    //文本为"关于"的按钮的单击事件
    clickAction_back(){
        //页面跳转语句
        router.replace({
            uri:'pages/index/index'
        })
    }
}
```

```
<!-- 第5章 index_Third.hml -->
<div class="container">
    <!-- 栈组件 -->
    <stack>
        <!-- 画布组件 -->
        <canvas class="canvas" ref="canvas"></canvas>
        <!-- 文本为"游戏结束"的文本组件 -->
        <text class="game_over" show="{{isShow}}">
            游戏结束
        </text>
    </stack>
    <!-- 添加文本为"←"的按钮组件 -->
    <input type="button" value="←" class="btn_left"
```

```
            onclick = "clickAction_left" />
    <!-- 添加文本为"变"的按钮组件 -->
    < input type = "button" value = "变" class = "btn_change"
            onclick = "clickAction_change" />
    <!-- 添加文本为"→"的按钮组件 -->
    < input type = "button" value = "→" class = "btn_right"
            onclick = "clickAction_right" />
    <!-- 添加文本为"重新开始"的按钮组件 -->
    < input type = "button" value = "重新开始" class = "btn_start"
            onclick = "clickAction_start" />
    <!-- 添加文本为"返回"的按钮组件 -->
    < input type = "button" value = "返回" class = "btn_back"
            onclick = "clickAction_back" />
</div >
```

```
/ * 第 5 章 index_Third.css * /
.container {
    flex - direction: column;
}

/ * 画布组件的样式 * /
.canvas {
    width: 350;
    height: 518px;
    margin - top: 75px;
    margin - left: 6px;
    background - color: black;
}

/ * 文本"游戏结束"样式 * /
.game_over {
    font - size: 30px;
    text - color: #0000FF;
    text - align: center;
    margin - top: 18px;
    margin - left: 120px;
}

/ * 文本为"←"的按钮样式 * /
.btn_left{
    height: 50px;
    width: 50px;
    margin - top: 1px;
    font - size:40px;
    text - color: #FFFFFF;
    text - align: center;
```

```
    radius: 100px;
    background-color:#78C6C5;
    margin-left: 55px;
}

/*文本为"变"的按钮样式*/
.btn_change{
    height: 50px;
    width: 50px;
    margin-top: 1px;
    font-size:35px;
    text-color: #FFFFFF;
    text-align: center;
    radius: 100px;
    background-color:#78C6C5;
    margin-left: 155px;
    margin-top: -50px;
}

/*文本为"→"的按钮样式*/
.btn_right{
    height: 50px;
    width: 50px;
    margin-top: 1px;
    font-size:40px;
    text-color: #FFFFFF;
    text-align: center;
    radius: 100px;
    background-color:#78C6C5;
    margin-left: 255px;
    margin-top: -50px;
}

/*文本为"重新开始"的按钮样式*/
.btn_start{
    height: 50px;
    width: 150px;
    margin-top: 1px;
    font-size:35px;
    text-color: #FFFFFF;
    text-align: center;
    radius: 100px;
    background-color:#78C6C5;
    margin-left: 60px;
}

/*文本为"返回"的按钮样式*/
.btn_back{
    height: 50px;
```

```
    width: 80px;
    margin-top: 1px;
    font-size:35px;
    text-color: #FFFFFF;
    text-align: center;
    radius: 100px;
    background-color:#78C6C5;
    margin-left: 230px;
    margin-top: -50px;
}
```

```
//第 5 章 index_Third.js
import router from '@system.router';

const length = 32;              //网格中方格的边长
const interval = 2;             //网格中方格的间距
const height = 15;              //网格中竖列方格的数量
const width = 10;               //网格中横列方格的数量
const left = 6;                 //网格的左端距手机边界的距离
const top = 5;                  //网格的顶端距手机边界的距离
//19 种方块所占网格的位置所对应的数值
const RedGrids1 = [[0, 3], [0, 4], [1, 4], [1, 5]];
const RedGrids2 = [[0, 5], [1, 5], [1, 4], [2, 4]];
const GreenGrids1 = [[0, 5], [0, 4], [1, 4], [1, 3]];
const GreenGrids2 = [[0, 4], [1, 4], [1, 5], [2, 5]];
const CyanGrids1 = [[0, 4], [1, 4], [2, 4], [3, 4]];
const CyanGrids2 = [[0, 3], [0, 4], [0, 5], [0, 6]];
const MagentaGrids1 = [[0, 4], [1, 3], [1, 4], [1, 5]];
const MagentaGrids2 = [[0, 4], [1, 4], [1, 5], [2, 4]];
const MagentaGrids3 = [[0, 3], [0, 4], [0, 5], [1, 4]];
const MagentaGrids4 = [[0, 5], [1, 5], [1, 4], [2, 5]];
const BlueGrids1 = [[0, 3], [1, 3], [1, 4], [1, 5]];
const BlueGrids2 = [[0, 5], [0, 4], [1, 4], [2, 4]];
const BlueGrids3 = [[0, 3], [0, 4], [0, 5], [1, 5]];
const BlueGrids4 = [[0, 5], [1, 5], [2, 5], [2, 4]];
const WhiteGrids1 = [[0, 5], [1, 5], [1, 4], [1, 3]];
const WhiteGrids2 = [[0, 4], [1, 4], [2, 4], [2, 5]];
const WhiteGrids3 = [[0, 5], [0, 4], [0, 3], [1, 3]];
const WhiteGrids4 = [[0, 4], [0, 5], [1, 5], [2, 5]];
const YellowGrids = [[0, 4], [0, 5], [1, 5], [1, 4]];
const grids_number = 4;           //方块的方格数量

var grids;                        //15 × 10 网格的二维数组
var NowGrids;                     //当前方块形态的二维数组
var row_number;                   //当前方块的总行数
var column_number;                //当前方块的总列数
var column_start;                 //当前方块所在 grids 的列数
//当前方块的颜色,0 表示灰色,1 代表红色,2 代表绿色,3 代表蓝绿色
//4 代表品红色,5 代表蓝色,6 代表白色,7 代表黄色
```

```
var GridsColor;
var Nowrow;                                //方块下落移动的行数
var Nowcolumn;                             //方块左右移动的列数
var Gameover;                              //表示游戏是否结束
var timer = null;                          //时间变量

export default {
    data: {
        isShow: false                      //先不显示游戏结束的文本
    },
    //生命周期函数
    onInit(){
        this.initialize();
        this.createGrids();
    },
    //生命周期函数
    onShow(){
        this.drawGrids();

        timer = setInterval(this.run, 750);   //启动时间任务
    },
    //初始化数据的函数
    initialize(){
        Gameover = true;
        grids = [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
    },
    //随机重新生成一种颜色方块的函数
    createGrids() {
        Nowrow = 0;
        Nowcolumn = 0;
```

```
let random = Math.random();        //生成[0,1)的随机数
//根据随机数的大小,调用相关的函数
if (random >= 0 && random < 0.2) {
    if (random >= 0 && random < 0.1){
        this.createRedGrids1();
    } else{
        this.createRedGrids2();
    }
} else if (random >= 0.2 && random < 0.4) {
    if (random >= 0.2 && random < 0.3){
        this.createGreenGrids1();
    }
    else{
        this.createGreenGrids2();
    }
} else if (random >= 0.4 && random < 0.45) {
    if (random >= 0.4 && random < 0.43)
        this.createCyanGrids1();
    else
        this.createCyanGrids2();
} else if (random >= 0.45 && random < 0.6) {
    if (random >= 0.45 && random < 0.48)
        this.createMagentaGrids1();
    else if (random >= 0.48 && random < 0.52)
        this.createMagentaGrids2();
    else if (random >= 0.52 && random < 0.56)
        this.createMagentaGrids3();
    else
        this.createMagentaGrids4();
} else if (random >= 0.6 && random < 0.75) {
    if (random >= 0.6 && random < 0.63)
        this.createBlueGrids1();
    else if (random >= 0.63 && random < 0.67)
        this.createBlueGrids2();
    else if (random >= 0.67 && random < 0.71)
        this.createBlueGrids3();
    else
        this.createBlueGrids4();
} else if (random >= 0.75 && random < 0.9) {
    if (random >= 0.75 && random < 0.78)
        this.createWhiteGrids1();
    else if (random >= 0.78 && random < 0.82)
        this.createWhiteGrids2();
    else if (random >= 0.82 && random < 0.86)
        this.createWhiteGrids3();
```

```
            else
                this.createWhiteGrids4();
        } else {
            this.createYellowGrids();
        }

        //判断游戏是否结束
        if (this.gameover()) {
            //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
            for (let row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
            }
        } else {
            clearInterval(timer);          //当游戏结束时,停止时间任务
            Gameover = false;              //绘制"游戏结束"文本
            this.isShow = true;            //显示游戏结束的文本
        }
    },
    //绘制网格的函数
    drawGrids(){
        var context = this.$refs.canvas.getContext('2d');

        //对数值进行判断,并将画笔设置为相应的颜色
        for (let row = 0; row < height; row ++ ){
            for (let column = 0; column < width; column ++ ){
                if (grids[row][column] == 0){
                    context.fillStyle = "#888888";
                } else if (grids[row][column] == 1){
                    context.fillStyle = "#FF0000";
                } else if (grids[row][column] == 2){
                    context.fillStyle = "#00FF00";
                } else if (grids[row][column] == 3){
                    context.fillStyle = "#FF00FF";
                } else if (grids[row][column] == 4){
                    context.fillStyle = "#00FFFF";
                } else if (grids[row][column] == 5){
                    context.fillStyle = "#0000FF";
                } else if (grids[row][column] == 6){
                    context.fillStyle = "#FFFFFF";
                } else if (grids[row][column] == 7){
                    context.fillStyle = "#FFFF00";
                }
                //绘制矩形
                context.fillRect(left + column * (length + interval),top + row * (length +
interval),length,length);
```

```
            }
        }
    },
    //方块自动下落的函数
    run(){
        //如果能够下落,则下落一行
        if (this.down()) {
            //将原来方块的颜色清除
            for (let row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] = 0;
            }
            Nowrow ++ ;
            //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
            for (let row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
            }
        } else {
            //如果不能下落,则判断能否消除和重新随机生成一种颜色方块
            this.eliminateGrids();
            this.createGrids();
        }
        //重新绘制网格
        this.drawGrids();
    },
    //判断方块能否下落的函数
    down(){
        let k;
        //表示方块已经接触到网格的底端
        if (Nowrow + row_number == height) {
            return false;
        }

        //判断方块的下一行是否存在其他方块
        for (let row = 0; row < grids_number; row ++ ) {
            k = true;
            for (let i = 0; i < grids_number; i ++ ) {
                 if (NowGrids[row][0] + 1 == NowGrids[i][0] && NowGrids[row][1] ==
NowGrids[i][1]) {
                    k = false;
                }
            }
            if (k) {
                if (grids[NowGrids[row][0] + Nowrow + 1][NowGrids[row][1] + Nowcolumn] != 0){
                    return false;
```

```
                }
            }
        }
        return true;
    },
    //实现方块向左移动的函数
    leftShift() {
        if (this.left()) {
            //将原来方块的颜色清除
            for (let row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] = 0;
            }
            Nowcolumn -- ;
            //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
            for (let row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
            }
        }
        //重新绘制网格
        this.drawGrids();
    },
    //判断方块能否向左移动的函数
    left() {
        let k;
        //表示方块已经接触到网格的左端
        if (Nowcolumn + column_start == 0) {
            return false;
        }

        //表示方块的左一列是否存在其他方块
        for (let column = 0; column < grids_number; column ++ ) {
            k = true;
            for (let j = 0; j < grids_number; j ++ ) {
                if (NowGrids[column][0] == NowGrids[j][0]
                && NowGrids[column][1] - 1 == NowGrids[j][1]) {
                    k = false;
                }
            }
            if (k) {
                if (grids[NowGrids[column][0] + Nowrow][NowGrids[column][1] + Nowcolumn -
1] != 0)
                    return false;
            }
        }
```

```
            return true;
        },
        //实现方块向右移动的函数
        rightShift() {
            if (this.right()) {
                //将原来方块的颜色清除
                for (let row = 0; row < grids_number; row ++ ) {
                    grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] = 0;
                }
                Nowcolumn ++ ;
                //将颜色方块添加到 15 × 10 网格的二维数组 grids 中
                for (let row = 0; row < grids_number; row ++ ) {
                    grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
                }
            }
            //重新绘制网格
            this.drawGrids();
        },
        //判断方块能否向右移动的函数
        right() {
            let k;
            //表示方块已经接触到网格的右端
            if (Nowcolumn + column_number + column_start == width) {
                return false;
            }

            //表示方块的右一列是否存在其他方块
            for (let column = 0; column < grids_number; column ++ ) {
                k = true;
                for (let j = 0; j < grids_number; j ++ ) {
                     if (NowGrids[column][0] == NowGrids[j][0]
                     && NowGrids[column][1] + 1 == NowGrids[j][1]) {
                        k = false;
                    }
                }
                if (k) {
                    if (grids[NowGrids[column][0] + Nowrow][NowGrids[column][1] + Nowcolumn +
1] != 0)
                        return false;
                }
            }

            return true;
        },
        //文本为"←"的按钮的单击事件
```

```
    clickAction_left(){
        if (Gameover){
            this.leftShift();
        }
    },
    //文本为"→"的按钮的单击事件
    clickAction_right(){
        if (Gameover){
            this.rightShift();
        }
    },
    //文本为"变"的按钮的单击事件
    clickAction_change(){
        if (Gameover){
            this.changGrids();
        }
    },
    //文本为"重新开始"的按钮的单击事件
    clickAction_start(){
        this.initialize();
        this.createGrids();
        this.drawGrids();
        this.isShow = false;
        clearInterval(timer);
        this.onShow();
    },
    //文本为"返回"的按钮的单击事件
    clickAction_back(){
        //页面跳转语句
        router.replace({
            uri:'pages/index/index'
        })
    },
    //实现方块改变形态的函数
    changGrids() {
        let Grids = NowGrids; //定义一个二维数组,用来存放改变形态前的方块形态

        for (let row = 0; row < grids_number; row ++ ) {
            //将原来方块的颜色清除
            grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] = 0;
        }
        if (column_number == 2 && Nowcolumn + column_start == 0) {
            Nowcolumn ++ ;
        }

        if (GridsColor == 1) {
```

```
            this.changRedGrids();
        } else if (GridsColor == 2) {
            this.changeGreenGrids();
        } else if (GridsColor == 3) {
            this.changeCyanGrids();
        } else if (GridsColor == 4) {
            this.changeMagentaGrids();
        } else if (GridsColor == 5) {
            this.changeBlueGrids();
        } else if (GridsColor == 6) {
            this.changeWhiteGrids();
        }
        if(this.change()){
            //如果能够改变形态,则将行的颜色方块添加到 15×10 网格的二维数组 grids 中
            for (let row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
            }
        }else{
            //如果不能改变形态,则将原来颜色方块添加到 15×10 网格的二维数组 grids 中
            NowGrids = Grids;
            for (let row = 0; row < grids_number; row ++ ) {
                grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] =
GridsColor;
            }
        }

        //重新绘制网格
        this.drawGrids();
    },
    //判断方块能否改变形态的函数
    change(){
        for (let row = 0; row < grids_number; row ++ ) {
            if (grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] != 0) {
                return false;
            }

            if (NowGrids[row][0] + Nowrow < 0 || NowGrids[row][0] + Nowrow >= height ||
NowGrids[row][1] + Nowcolumn < 0 || NowGrids[row][1] + Nowcolumn >= width) {
                return false;
            }
        }

        return true;
    },
    //判断游戏是否结束
```

```
    gameover() {
        //当生成方块的任一方格的数值不为 0 时,说明游戏结束
        for (let row = 0; row < grids_number; row ++ ) {
            if (grids[NowGrids[row][0] + Nowrow][NowGrids[row][1] + Nowcolumn] != 0) {
                return false;
            }
        }
        return true;
    },
    //实现消除整行方格的函数
    eliminateGrids() {
        let k;
        //从下往上循环判断每一行是否已经满了
        for (let row = height - 1; row >= 0; row -- ) {
            k = true;
            //如果该行存在一个或一个以上灰色的方格,则说明该行没有满足条件
            for (let column = 0; column < width; column ++ ) {
                if (grids[row][column] == 0)
                    k = false;
            }
            if (k) {
                //将该行上面的所有行整体向下移动一格
                for (let i = row - 1; i >= 0; i -- ) {
                    for (let j = 0; j < width; j ++ ) {
                        grids[i + 1][j] = grids[i][j];
                    }
                }
                for (let n = 0; n < width; n ++ ) {
                    grids[0][n] = 0;
                }
                //再次判断该行是否满足消除的条件
                row ++ ;
            }
        }
        //重新绘制网格
        this.drawGrids();
    },
    //在同一种颜色的不同形态之间切换
    changRedGrids() {
        if (NowGrids == RedGrids1) {
            this.createRedGrids2();
        } else if (NowGrids == RedGrids2) {
            this.createRedGrids1();
        }
    },
    changeGreenGrids() {
```

```
        if (NowGrids == GreenGrids1) {
            this.createGreenGrids2();
        } else if (NowGrids == GreenGrids2) {
            this.createGreenGrids1();
        }
    },
    changeCyanGrids() {
        if (NowGrids == CyanGrids1) {
            this.createCyanGrids2();
        } else if (NowGrids == CyanGrids2) {
            this.createCyanGrids1();
        }
    },
    changeMagentaGrids() {
        if (NowGrids == MagentaGrids1) {
            this.createMagentaGrids2();
        } else if (NowGrids == MagentaGrids2) {
            this.createMagentaGrids3();
        } else if (NowGrids == MagentaGrids3) {
            this.createMagentaGrids4();
        } else if (NowGrids == MagentaGrids4) {
            this.createMagentaGrids1();
        }
    },
    changeBlueGrids() {
        if (NowGrids == BlueGrids1) {
            this.createBlueGrids2();
        } else if (NowGrids == BlueGrids2) {
            this.createBlueGrids3();
        } else if (NowGrids == BlueGrids3) {
            this.createBlueGrids4();
        } else if (NowGrids == BlueGrids4) {
            this.createBlueGrids1();
        }
    },
    changeWhiteGrids() {
        if (NowGrids == WhiteGrids1) {
            this.createWhiteGrids2();
        } else if (NowGrids == WhiteGrids2) {
            this.createWhiteGrids3();
        } else if (NowGrids == WhiteGrids3) {
            this.createWhiteGrids4();
        } else if (NowGrids == WhiteGrids4) {
            this.createWhiteGrids1();
        }
    },
```

```
        //对对应颜色方块的不同形态赋予 NowGrids、row_number、column_number、
        //GridsColor、column_start 的值
        createRedGrids1() {
            NowGrids = RedGrids1;
            row_number = 2;
            column_number = 3;
            GridsColor = 1;
            column_start = 3;
        },
        createRedGrids2() {
            NowGrids = RedGrids2;
            row_number = 3;
            column_number = 2;
            GridsColor = 1;
            column_start = 4;
        },
        createGreenGrids1() {
            NowGrids = GreenGrids1;
            row_number = 2;
            column_number = 3;
            GridsColor = 2;
            column_start = 3;
        },
        createGreenGrids2() {
            NowGrids = GreenGrids2;
            row_number = 3;
            column_number = 2;
            GridsColor = 2;
            column_start = 4;
        },
        createCyanGrids1() {
            NowGrids = CyanGrids1;
            row_number = 4;
            column_number = 1;
            GridsColor = 3;
            column_start = 4;
        },
        createCyanGrids2() {
            NowGrids = CyanGrids2;
            row_number = 1;
            column_number = 4;
            GridsColor = 3;
            column_start = 3;
        },
        createMagentaGrids1() {
```

```
        NowGrids = MagentaGrids1;
        row_number = 2;
        column_number = 3;
        GridsColor = 4;
        column_start = 3;
    },
    createMagentaGrids2() {
        NowGrids = MagentaGrids2;
        row_number = 3;
        column_number = 2;
        GridsColor = 4;
        column_start = 4;
    },
    createMagentaGrids3() {
        NowGrids = MagentaGrids3;
        row_number = 2;
        column_number = 3;
        GridsColor = 4;
        column_start = 3;
    },
    createMagentaGrids4() {
        NowGrids = MagentaGrids4;
        row_number = 3;
        column_number = 2;
        GridsColor = 4;
        column_start = 4;
    },
    createBlueGrids1() {
        NowGrids = BlueGrids1;
        row_number = 2;
        column_number = 3;
        GridsColor = 5;
        column_start = 3;
    },
    createBlueGrids2() {
        NowGrids = BlueGrids2;
        row_number = 3;
        column_number = 2;
        GridsColor = 5;
        column_start = 4;
    },
    createBlueGrids3() {
        NowGrids = BlueGrids3;
        row_number = 2;
        column_number = 3;
```

```
            GridsColor = 5;
            column_start = 3;
        },
        createBlueGrids4() {
            NowGrids = BlueGrids4;
            row_number = 3;
            column_number = 2;
            GridsColor = 5;
            column_start = 4;
        },
        createWhiteGrids1() {
            NowGrids = WhiteGrids1;
            row_number = 2;
            column_number = 3;
            GridsColor = 6;
            column_start = 3;
        },
        createWhiteGrids2() {
            NowGrids = WhiteGrids2;
            row_number = 3;
            column_number = 2;
            GridsColor = 6;
            column_start = 4;
        },
        createWhiteGrids3() {
            NowGrids = WhiteGrids3;
            row_number = 2;
            column_number = 3;
            GridsColor = 6;
            column_start = 3;
        },
        createWhiteGrids4() {
            NowGrids = WhiteGrids4;
            row_number = 3;
            column_number = 2;
            GridsColor = 6;
            column_start = 4;
        },
        createYellowGrids() {
            NowGrids = YellowGrids;
            row_number = 2;
            column_number = 2;
            GridsColor = 7;
            column_start = 4;
        }
    }
```

实现效果和前16节的实现效果是一致的。

现在对这两个程序Game和Game_JS进行对比。

对于编写布局方式，用Java开发的项目有以下两种布局方式。

(1) 在代码中创建布局：用代码创建Component和ComponentContainer对象，为这些对象设置合适的布局参数和属性值，并将Component添加到ComponentContainer中，从而创建出完整界面。例如程序Game中的ThirdAbilitySlice.java文件的布局方式。

(2) 在XML文件中声明UI布局：按层级结构来描述Component和ComponentContainer的关系，给组件节点设定合适的布局参数和属性值，在代码中可直接加载，以便生成此布局，例如程序Game中的MainAbilitySlice.java和SecondAbilitySlice.java文件的布局方式。需要说明的是，以这两种方式创建出的布局没有本质差别，在XML文件中声明布局时，在加载后同样可在代码中对该布局进行修改。

用JavaScript开发的项目只有一种布局方式，布局文件主要为hml文件和css文件。hml文件是页面的结构，它用于描述页面中包含哪些组件；scc文件是页面的样式，它用于描述页面中的组件是什么样的。

用Java开发的项目在XML文件中声明UI布局和用JavaScript开发的项目的布局方式很雷同。

例如对于主页面布局，对比代码文本ability_main.xml、代码文件index.hml、代码文件index.css，发现都可以添加两个按钮Button，并且可以对按钮Button的各种属性值进行设置，不同的只是部分属性名称不一致，但其对应的描述是一致的。如在这两种布局方式中，组件宽度和组件高度的名称同样为width和height，而文本大小和文本的对齐方式，在Java中的名称为text_size和text_alignment，在JavaScript中的名称为font-size和text-align。

对于组件间的交互方式，用Java开发的项目主要编写在AbilitySlice.java文件中，而用JavaScript开发的项目主要编写在js文件中，两者也有一些相同点和不同点。

相同之处在于：两者均有对应的页面生命周期事件，组件间的交互以函数间相互调用的方式为主，大部分语句是相同的，例如两者均有if语句和for语句，而且用法都是相同的。

不同之处在于：在Java中的函数必须定义类型void、String、int等，而在JavaScript中的函数是不需要定义类型的，函数的定义直接为"函数名(形式参数){函数体}"，而函数的调用为"this.函数名(实际参数)"。另外一点不同的地方是少部分语句，例如在Java中的变量的类型有很多种，有整型int、双精度类型double、字符串类型String等，常量则为在类型前添加final；在JavaScript中的变量类型为两种：全局变量var和局部变量let，常量则为const。

总地来讲，对于用Java和JavaScript开发HarmonyOS的项目有很大相通之处，当掌握了其中一种语言后，学习另外一种语言也会变得十分容易。对于部分项目来讲，当用一种编程语言开发出来后，基本可以采用复制、粘贴的方式再加修改部分语句后，即可实现用另外一种编程语言开发同一个项目了。正所谓"程序＝算法＋数据结构"，希望各位读者在以后的开发学习中不要过于纠结编程语言的选择。