实验 6：推箱子游戏

本实验来自于周文洁老师的《微信小程序开发实战》第十三章。在学习了<canvas>组件和小程序界面 API 中绘图相关的用法以后，我们尝试制作简易的推箱子小游戏

学习目标：1、综合应用所学的知识创建完整的推箱子游戏；2、熟练掌握<canvas>和绘图 API。

游戏图片下载链接：https://gaopursuit.oss-cn-beijing.aliyuncs.com/course/mobileDev/boxgame_images.zip

13.1 需求分析

视频讲解

本项目一共需要两个页面，即首页和游戏页面，其中首页用于呈现关卡菜单，点击对应难度的关卡后进入游戏画面。

13.1.1 首页功能需求

首页功能需求如下：

(1) 首页需要包含标题和关卡列表。

(2) 关卡至少要有 4 个关卡选项，每个关卡显示预览图片和第几关。

(3) 点击关卡列表可以打开对应的游戏画面。

13.1.2 游戏页功能需求

游戏页功能需求如下：

(1) 游戏页面需要显示第几关、游戏画面、方向键和“重新开始”按钮。

(2) 点击方向键可以使游戏主角自行移动或推动箱子前进。

(3) 游戏画面由 8×8 的小方块组成，主要包括地板、围墙、箱子、游戏主角和目的地。

(4) 点击“重新开始”按钮可以将箱子和游戏主角回归初始位置并重新开始游戏。

13.2 项目创建

视频讲解

本项目创建选择空白文件夹 boxGame，效果如图 13-1 所示。

单击“新建”按钮完成项目创建，然后准备手动修改页面配置文件。

图 13-1　小程序项目填写效果示意图

13.3　页面配置

13.3.1　创建页面文件

项目创建完毕后，在根目录中会生成文件夹 pages 用于存放页面文件。一般来说首页默认命名为 index，表示小程序运行的第一个页面；其他页面名称可以自定义。本项目有两个页面文件，需要创建 index(首页页面)和 game(游戏页面)。

具体操作如下：

(1) 将 app.json 文件内 pages 属性中的“pages/logs/logs”改成“pages/game/game”。

(2) 按快捷键 Ctrl+S 保存修改后会在 pages 文件夹下自动生成 game 目录。

13.3.2　删除和修改文件

具体操作如下：

(1) 删除 utils 文件夹及其内部所有内容。

(2) 删除 pages 文件夹下的 logs 目录及其内部所有内容。

(3) 删除 index.wxml 和 index.wxss 中的全部代码。

(4) 删除 index.js 中的全部代码，并且输入关键词 page 找到第二个选项按回车键让其自动补全函数(如图 13-2 所示)。

(5) 删除 app.wxss 中的全部代码。

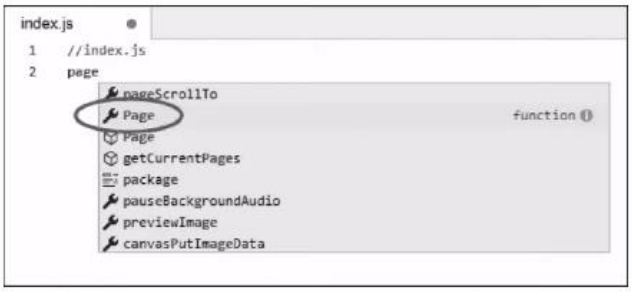


图 13-2　输入关键词创建 Page 函数

(6) 删除 app.js 中的全部代码，并且输入关键词 app 找到第一个选项按回车键让其自动补全函数(如图 13-3 所示)。

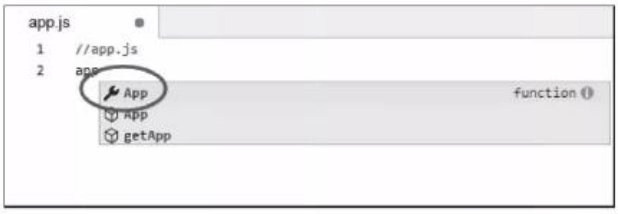


图 13-3　输入关键词创建 App 函数

13.3.3　创建其他文件

接下来创建其他自定义文件，本项目还需要以下两个文件夹。

- images：用于存放图片素材；
- utils：用于存放公共 JS 文件。

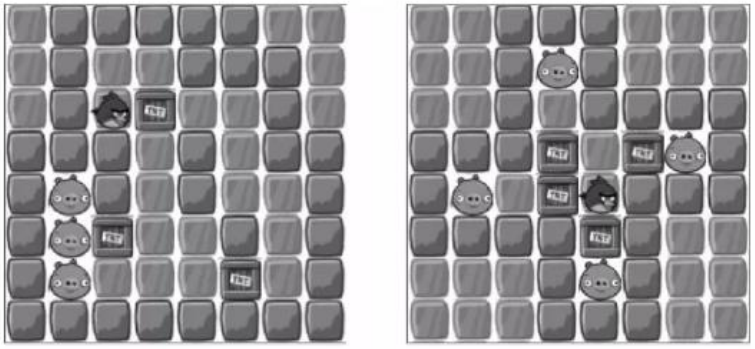
单击目录结构左上角的＋号创建文件夹，并分别命名为 images 和 utils。

1 添加图片文件

本项目将在首页中用到 4 幅关卡图片，图片素材如图 13-4 所示。

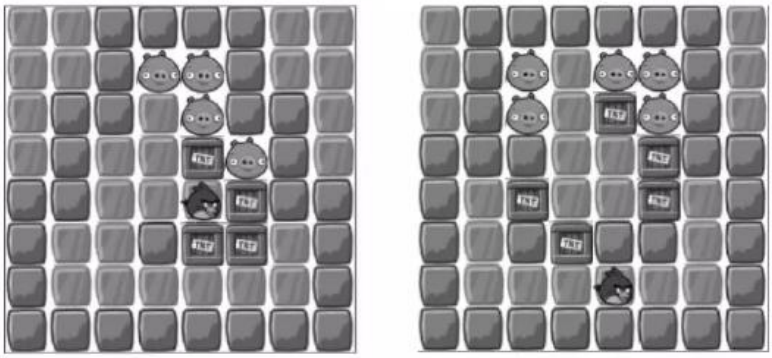
右击目录结构中的 images 文件夹，选择“硬盘打开”，将图片复制、粘贴进去。

此外还需要在游戏页面中用到 5 个游戏图标素材，如图 13-5 所示。



(a) level01.png　　(b) level02.png

图 13-4　关卡图片素材展示



(c) level03.png　　(d) level04.png

图 13-4 （续）

(a) bird.png　(b) box.png　(c) ice.png　(d) pig.png　(e) stone.png

图 13-5 游戏方块图标素材展示

右击目录结构中的 images 文件夹，选择“硬盘打开”，新建二级文件夹 icons，然后将全部图标素材复制、粘贴进去。

2 创建公共 JS 文件

右击 utils 文件夹，选择“新建”→JS，输入 data 后按回车键创建公共文件 data.js，如图 13-6 所示。

全部完成后的目录结构如图 13-7 所示。

图 13-6 新建 JS 文件

图 13-7 全部文件创建完成

此时文件配置就全部完成，13.4 节将正式进行页面布局和样式设计。

13.4.1 导航栏设计

视频讲解

小程序默认导航栏是黑底白字的效果，可以通过在 app.json 中对 window 属性进行重新配置来自定义导航栏效果。更改后的 app.json 文件代码如下：

```
1.  {
2.    "pages": [代码略],
3.    "window": {
4.      "navigationBarBackgroundColor": "#E64340",
5.      "navigationBarTitleText": "推箱子游戏"
6.    }
7.  }
```

上述代码可以更改导航栏背景色为珊瑚红色、字体为白色，效果如图 13-8 所示。

图 13-8 自定义导航栏效果

13.4.2 页面设计

视频讲解

1 公共样式设计

首先在 app.wxss 中设置页面容器和顶端标题的公共样式，代码如下：

```
1.  /* 页面容器样式 */
2.  .container {
3.    height: 100vh;
4.    color: #E64340;
5.    font-weight: bold;
6.    display: flex;
7.    flex-direction: column;
8.    align-items: center;
9.    justify-content: space-evenly;
10. }
11. /* 顶端标题样式 */
12. .title {
13.   font-size: 18pt;
14. }
```

视频讲解

2 首页设计

首页主要包含两部分内容，即标题和关卡列表，页面设计如图 13-9 所示。计划使用如下组件。

- 顶端标题：< view >容器；
- 关卡列表：< view >容器，内部使用数组循环。

WXML(pages/index/index.wxml)代码如下：

```
1.  <view class='container'>
2.    <!-- 标题 -->
3.    <view class='title'>游戏选关</view>
4.
```

```
5.     <!-- 关卡列表 -->
6.       <view class = 'levelBox'>
7.       <view class = 'box'>
8.         <image src = '/images/level01.png'></image>
9.         <text>第 1 关</text>
10.      </view>
11.    </view>
12.
13. </view>
```

相关 WXSS(pages/index/index.wxss)代码片段如下：

```
1.  /* 关卡列表区域 */
2.  .levelBox {
3.    width: 100 % ;
4.  }
5.
6.  /* 单个关卡区域 */
7.  .box {
8.    width: 50 % ;
9.    float: left;
10.   margin: 20rpx 0;
11.   display: flex;
12.   flex - direction: column;
13.   align - items: center;
14. }
15.
16. /* 选关图片 */
17. image {
18.   width: 300rpx;
19.   height: 300rpx;
20. }
```

当前效果如图 13-10 所示。

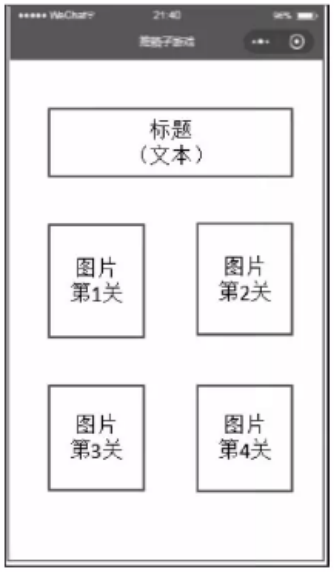


图 13-9　首页设计图

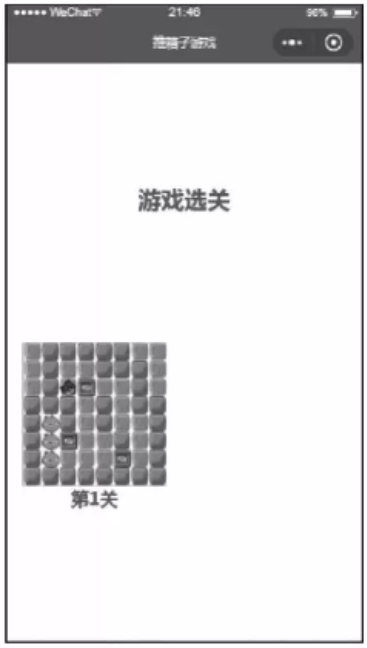


图 13-10　首页效果图

由图可见，此时可以显示标题和一个临时关卡。由于尚未获得关卡数据，所以暂时无法显示完整的关卡列表，仅供作为样式参考。

3 游戏页面设计

视频讲解

游戏页面需要用户点击首页的关卡列表，然后在新窗口中打开该页面。游戏页面包括游戏关卡标题、游戏画面、方向键和“重新开始”按钮，页面设计如图 13-11 所示。

图 13-11　游戏页面设计图

由于暂时没有做点击跳转的逻辑设计，所以可以在开发工具顶端选择“普通编译”下的“添加编译模式”，并携带临时测试参数 level=0，如图 13-12 所示。

此时预览就可以直接显示 game 页面了，设计完毕后再改回“普通编译”模式即可重新显示首页。

计划使用如下组件。

- <view>：整体容器和顶端标题；
- <canvas>：游戏画布；

图 13-12　添加 game 页面的编译模式

- <button>：4 个方向键和 1 个“重新开始”按钮。

WXML(pages/game/game.wxml)代码如下：

```
1.  <view class = 'container'>
2.    <!-- 关卡提示 -->
3.    <view class = 'title'>第 1 关</view>
4.
5.    <!-- 游戏画布 -->
6.    <canvas canvas - id = 'myCanvas'></canvas>
7.
8.    <!-- 方向键 -->
9.    <view class = 'btnBox'>
10.     <button type = 'warn'> ↑ </button>
11.     <view>
12.       <button type = 'warn'>←</button>
13.       <button type = 'warn'> ↓ </button>
14.       <button type = 'warn'>→</button>
15.     </view>
16.   </view>
```

```
17.
18.   <!--“重新开始”按钮 -->
19.   <button type = 'warn'>重新开始</button>
20. </view>
```

WXSS(pages/game/game.wxss)代码如下：

```
1.  /* 游戏画布样式 */
2.  canvas {
3.    border: 1rpx solid;
4.    width: 320px;
5.    height: 320px;
6.  }
7.
8.  /* 方向键按钮整体区域 */
9.  .btnBox {
10.   display: flex;
11.   flex-direction: column;
12.   align-items: center;
13. }
14.
15. /* 方向键按钮第二行 */
16. .btnBox view {
17.   display: flex;
18.   flex-direction: row;
19. }
20.
21. /* 所有方向键按钮 */
22. .btnBox button {
23.   width: 90rpx;
24.   height: 90rpx;
25. }
26.
27. /* 所有按钮样式 */
28. button {
29.   margin: 10rpx;
30. }
```

图 13-13　游戏页面效果图

当前效果如图 13-13 所示。

由图可见，此时可以显示完整样式效果。由于尚未获得游戏数据，所以暂时无法根据用户点击的关卡入口显示对应的游戏内容，仅供作为样式参考。

此时页面布局与样式设计就已完成，13.5 节将介绍如何进行逻辑处理。

13.5　逻辑实现

13.5.1　公共逻辑

视频讲解

在公共 JS 文件(utils/data.js)中配置游戏地图的数据，代码如下：

```
1.  //==========================================================
2.  //地图数据 map1～map4
3.  //地图数据：1 为墙、2 为路、3 为终点、4 为箱子、5 为人物、0 为墙的外围
4.  //==========================================================
```

```
5.  //关卡 1
6.  var map1 = [
7.    [0, 1, 1, 1, 1, 1, 0, 0],
8.    [0, 1, 2, 2, 1, 1, 1, 0],
9.    [0, 1, 5, 4, 2, 2, 1, 0],
10.   [1, 1, 1, 2, 1, 2, 1, 1],
11.   [1, 3, 1, 2, 1, 2, 2, 1],
12.   [1, 3, 4, 2, 2, 1, 2, 1],
13.   [1, 3, 2, 2, 2, 4, 2, 1],
14.   [1, 1, 1, 1, 1, 1, 1, 1]
15. ]
16. //关卡 2
17. var map2 = [
18.   [0, 0, 1, 1, 1, 0, 0, 0],
19.   [0, 0, 1, 3, 1, 0, 0, 0],
20.   [0, 0, 1, 2, 1, 1, 1, 1],
21.   [1, 1, 1, 4, 2, 4, 3, 1],
22.   [1, 3, 2, 4, 5, 1, 1, 1],
23.   [1, 1, 1, 1, 4, 1, 0, 0],
24.   [0, 0, 0, 1, 3, 1, 0, 0],
25.   [0, 0, 0, 1, 1, 1, 0, 0]
26.
27. ]
28. //关卡 3
29. var map3 = [
30.   [0, 0, 1, 1, 1, 1, 0, 0],
31.   [0, 0, 1, 3, 3, 1, 0, 0],
32.   [0, 1, 1, 2, 3, 1, 1, 0],
33.   [0, 1, 2, 2, 4, 3, 1, 0],
34.   [1, 1, 2, 2, 5, 4, 1, 1],
35.   [1, 2, 2, 1, 4, 4, 2, 1],
36.   [1, 2, 2, 2, 2, 2, 2, 1],
37.   [1, 1, 1, 1, 1, 1, 1, 1]
38. ]
39. //关卡 4
40. var map4 = [
41.   [0, 1, 1, 1, 1, 1, 1, 0],
42.   [0, 1, 3, 2, 3, 3, 1, 0],
43.   [0, 1, 3, 2, 4, 3, 1, 0],
44.   [1, 1, 1, 2, 2, 4, 1, 1],
45.   [1, 2, 4, 2, 2, 4, 2, 1],
46.   [1, 2, 1, 4, 1, 1, 2, 1],
47.   [1, 2, 2, 2, 5, 2, 2, 1],
48.   [1, 1, 1, 1, 1, 1, 1, 1]
49. ]
```

这里分别使用 map1～map4 代表 4 个不同关卡的地图数据，以二维数组的形式存放。当前地图均由 8×8 的方格组成，每个位置的数字代表对应的图标素材。

当前地图数据和图片素材仅供参考，开发者也可以自行修改游戏布局和图片。

然后需要在 data.js 中使用 module.exports 语句暴露数据出口，代码片段如下：

```
1.  module.exports = {
2.    maps: [map1, map2, map3, map4]
3.  }
```

现在就完成了公共逻辑处理的部分。

最后需要在 game 页面的 JS 文件顶端引用公共 JS 文件，引用代码如下：

```
var data = require('../../utils/data.js')                    //引用公共 JS 文件
```

需要注意小程序在这里暂时还不支持绝对路径引用，只能使用相对路径。

13.5.2 首页逻辑

首页主要有两个功能需要实现，一是展示关卡列表；二是点击图片能跳转到游戏页面。

1 关卡列表展示

视频讲解

在 JS 文件的 data 中录入关卡图片的数据信息，这里以 4 个关卡为例。

相关 JS(pages/index/index.js)代码片段如下：

```
1.  Page({
2.    /**
3.      * 页面的初始数据
4.      */
5.    data: {
6.      levels: [
7.        'level01.png',
8.        'level02.png',
9.        'level03.png',
10.       'level04.png'
11.     ]
12.   },
13. })
```

接着为关卡对应的< view >组件添加 wx:for 属性循环显示关卡列表数据和图片。

修改后的 WXML(pages/index/index.wxml)代码如下：

```
1.  < view class = 'container'>
2.    <!-- 标题 -->
3.    < view class = 'title'>游戏选关</view>
4.
5.    <!-- 关卡列表 -->
6.    < view class = 'levelBox'>
7.      < view class = 'box' wx:for = '{{levels}}' wx:key =
'levels{{index}}'>
8.        < image src = '/images/{{item}}'></image>
9.        < text >第{{index + 1}}关</text >
10.     </view >
11.   </view >
12.
13. </view >
```

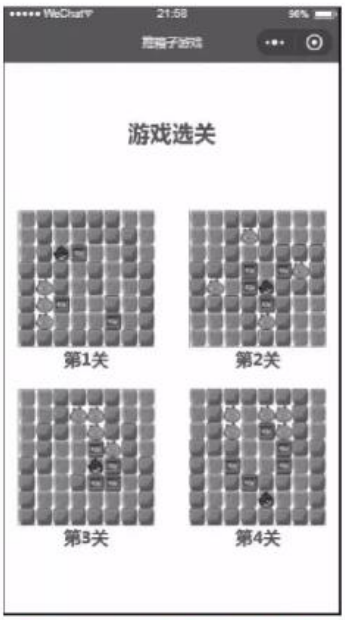


图 13-14 首页关卡列表展示

此时页面效果如图 13-14 所示。

2 点击跳转游戏页面

视频讲解

若希望用户点击关卡图片即可实现跳转，需要首先为关卡列表项目添加点击事件。

相关 WXML(pages/index/index.wxml)代码片段修改如下：

```
1.  <view class = 'container'>
2.    <!-- 标题 -->
3.    <view class = 'title'>游戏选关</view>
4.
5.    <!-- 关卡列表 -->
6.    <view class = 'levelBox'>
7.      <view class = 'box' wx:for = '{{levels}}' wx:key = 'levels{{index}}' bindtap = 'chooseLevel'
data - level = '{{index}}'>
8.        <image src = '/images/{{item}}'></image>
9.        <text>第{{index + 1}}关</text>
10.     </view>
11.   </view>
12.
13. </view>
```

上述代码表示为关卡添加了自定义点击事件函数 chooseLevel，并且使用 data-level 属性携带了关卡图片下标信息。

然后在对应的 index.js 文件中添加 chooseLevel 函数的内容，代码片段如下：

```
1.  Page({
2.    /**
3.      * 自定义函数 -- 游戏选关
4.      */
5.    chooseLevel: function(e) {
6.      let level = e.currentTarget.dataset.level
7.      wx.navigateTo({
8.        url: '../game/game?level = ' + level
9.      })
10.   },
11. })
```

此时已经可以点击跳转到 game 页面，并且成功携带了关卡图片数据，但是仍需在 game 页面进行携带数据的接收处理才可显示正确的游戏画面。

13.5.3 游戏页逻辑

游戏页主要有以下几个功能需要实现：

- 显示当前是第几关；
- 游戏地图的绘制；
- 4 个方向键可以移动游戏主角；
- 点击“重新开始”按钮可以使游戏地图还原成最初状态。

1 显示当前第几关

视频讲解

在首页逻辑中已经实现了页面跳转并携带了关卡对应的图片信息，现在需要在游戏页面接收关卡信息，并显示对应的图片内容。

相关 JS(pages/game/game.js)代码片段如下：

```
1.  Page({
2.    /**
3.      * 页面的初始数据
4.      */
5.    data: {
```

```
6.        level: 1
7.      },
8.      /**
9.       * 生命周期函数--监听页面加载
10.      */
11.     onLoad: function(options) {
12.       //获取关卡
13.       let level = options.level
14.       //更新页面关卡标题
15.       this.setData({
16.         level: parseInt(level) + 1
17.       })
18.     },
19. })
```

修改 WXML(pages/game/game.wxml)代码片段如下：

```
1.  <view class='container'>
2.    <!--关卡提示-->
3.    <view class='title'>第{{level}}关</view>
4.
5.    ...
6.  </view>
```

此时重新从首页点击不同的关卡图片跳转就可以发现经能够正确显示对应的内容了。

运行效果如图 13-15 所示。

图 13-15　首页列表中的选关效果

2 游戏逻辑实现

视频讲解

1）准备工作

首先在 game.js 文件的顶端记录一些游初始数据信息。

对应的 JS(pages/game/game.js)代码段添加如下：

```
1.  //地图图层数据
2.  var map = [
3.    [0, 0, 0, 0, 0, 0, 0, 0],
4.    [0, 0, 0, 0, 0, 0, 0, 0],
5.    [0, 0, 0, 0, 0, 0, 0, 0],
6.    [0, 0, 0, 0, 0, 0, 0, 0],
7.    [0, 0, 0, 0, 0, 0, 0, 0],
8.    [0, 0, 0, 0, 0, 0, 0, 0],
9.    [0, 0, 0, 0, 0, 0, 0, 0],
10.   [0, 0, 0, 0, 0, 0, 0, 0]
11. ]
12. //箱子图层数据
13. var box = [
14.   [0, 0, 0, 0, 0, 0, 0, 0],
15.   [0, 0, 0, 0, 0, 0, 0, 0],
16.   [0, 0, 0, 0, 0, 0, 0, 0],
17.   [0, 0, 0, 0, 0, 0, 0, 0],
18.   [0, 0, 0, 0, 0, 0, 0, 0],
19.   [0, 0, 0, 0, 0, 0, 0, 0],
20.   [0, 0, 0, 0, 0, 0, 0, 0],
21.   [0, 0, 0, 0, 0, 0, 0, 0]
```

```
22. ]
23. //方块的宽度
24. var w = 40
25. //初始化游戏主角(小鸟)的行与列
26. var row = 0
27. var col = 0
28.
29. Page({
30.     …
31. })
```

2）初始化游戏画面

首先需要根据当前是第几关读取对应的游戏地图信息，并更新到游戏初始数据中。

视频讲解

在 game.js 文件中添加 initMap 函数，用于初始化游戏地图数据。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.      * 自定义函数 -- 初始化地图数据
4.      */
5.    initMap: function(level) {
6.      //读取原始的游戏地图数据
7.      let mapData = data.maps[level]
8.      //使用双重 for 循环记录地图数据
9.      for (var i = 0; i < 8; i++) {
10.       for (var j = 0; j < 8; j++) {
11.         box[i][j] = 0
12.         map[i][j] = mapData[i][j]
13.
14.         if (mapData[i][j] == 4) {
15.           box[i][j] = 4
16.           map[i][j] = 2
17.         } else if (mapData[i][j] == 5) {
18.           map[i][j] = 2
19.           //记录小鸟的当前行和列
20.           row = i
21.           col = j
22.         }
23.       }
24.     }
25.   },
26. })
```

上述代码首先从公共函数文件 data.js 中读取对应关卡的游戏地图数据，然后使用双重 for 循环对每一块地图数据进行解析，并更新当前游戏的初始地图数据、箱子数据以及游戏主角(小鸟)的所在位置。

然后在 game.js 中添加自定义函数 drawCanvas，用于将地图信息绘制到画布上。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.      * 自定义函数 -- 绘制地图
4.      */
5.    drawCanvas: function() {
```

```
6.      let ctx = this.ctx
7.      //清空画布
8.      ctx.clearRect(0, 0, 320, 320)
9.      //使用双重 for 循环绘制 8x8 的地图
10.     for (var i = 0; i < 8; i++) {
11.       for (var j = 0; j < 8; j++) {
12.         //默认是道路
13.         let img = 'ice'
14.         if (map[i][j] == 1) {
15.           img = 'stone'
16.         } else if (map[i][j] == 3) {
17.           img = 'pig'
18.         }
19.
20.         //绘制地图
21.         ctx.drawImage('/images/icons/' + img + '.png', j * w, i * w, w, w)
22.
23.         if (box[i][j] == 4) {
24.           //叠加绘制箱子
25.           ctx.drawImage('/images/icons/box.png', j * w, i * w, w, w)
26.         }
27.       }
28.     }
29.
30.     //叠加绘制小鸟
31.     ctx.drawImage('/images/icons/bird.png', col * w, row * w, w, w)
32.
33.     ctx.draw()
34.   },
35. })
```

最后在 game.js 的 onLoad 函数中创建画布上下文，并依次调用自定义函数 initMap 和 drawCanvas。对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.      * 生命周期函数 -- 监听页面加载
3.      */
4.    onLoad: function(options) {
5.      //获取关卡
6.      …代码略…
7.      //更新页面关卡标题
8.      …代码略…
9.      //创建画布上下文
10.     this.ctx = wx.createCanvasContext('myCanvas')
11.     //初始化地图数据
12.     this.initMap(level)
13.     //绘制画布内容
14.     this.drawCanvas()
15.   },
16. })
```

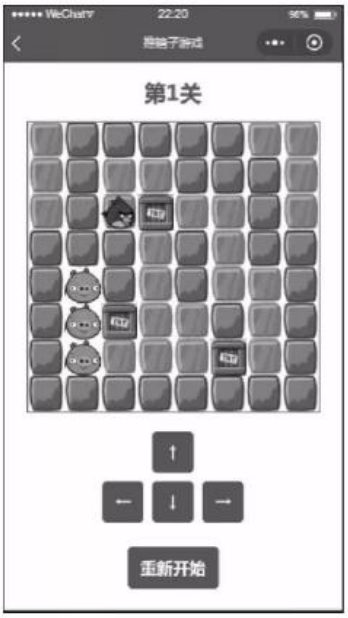


图 13-16　绘制地图效果

当前效果如图 13-16 所示。

3）方向键逻辑实现

修改 game.wxml 页面中的 4 个方向键 <button>，为其绑定点击事件。

视频讲解

WXML(pages/game/game.wxml)代码修改后如下：

```
1.  <view class = 'container'>
2.    <!-- 关卡提示(代码略) -->
3.
4.    <!-- 游戏画布(代码略) -->
5.
6.    <!-- 方向键 -->
7.    <view class = 'btnBox'>
8.      <button type = 'warn' bindtap = 'up'> ↑ </button>
9.      <view>
10.       <button type = 'warn' bindtap = 'left'> ← </button>
11.       <button type = 'warn' bindtap = 'down'> ↓ </button>
12.       <button type = 'warn' bindtap = 'right'> → </button>
13.     </view>
14.   </view>
15.
16.   <!--"重新开始"按钮(代码略) -->
17. </view>
```

在 game.js 文件中添加自定义函数 up、down、left 和 right，分别用于实现游戏主角(小鸟)在上、下、左、右 4 个方向的移动，每次点击在条件允许的情况下移动一格。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.     * 自定义函数 -- 方向键: 上
4.     */
5.    up: function() {
6.      //不在最顶端才考虑上移
7.      if (row > 0) {
8.        //如果上方不是墙或箱子,可以移动小鸟
9.        if (map[row - 1][col] != 1 && box[row - 1][col] != 4) {
10.         //更新当前小鸟的坐标
11.         row = row - 1
12.       }
13.       //如果上方是箱子
14.       else if (box[row - 1][col] == 4) {
15.         //箱子不在最顶端才能考虑推动
16.         if (row - 1 > 0) {
17.           //如果箱子上方不是墙或箱子
18.           if (map[row - 2][col] != 1 && box[row - 2][col] != 4) {
19.             box[row - 2][col] = 4
20.             box[row - 1][col] = 0
21.             //更新当前小鸟的坐标
22.             row = row - 1
23.           }
24.         }
25.       }
26.       //重新绘制地图
27.       this.drawCanvas()
28.     }
29.   },
30.   /**
31.    * 自定义函数 -- 方向键: 下
```

```
32.     */
33.   down: function() {
34.     //不在最底端才考虑下移
35.     if (row < 7) {
36.       //如果下方不是墙或箱子,可以移动小鸟
37.       if (map[row + 1][col] != 1 && box[row + 1][col] != 4) {
38.         //更新当前小鸟的坐标
39.         row = row + 1
40.       }
41.       //如果下方是箱子
42.       else if (box[row + 1][col] == 4) {
43.         //箱子不在最底端才能考虑推动
44.         if (row + 1 < 7) {
45.           //如果箱子下方不是墙或箱子
46.           if (map[row + 2][col] != 1 && box[row + 2][col] != 4) {
47.             box[row + 2][col] = 4
48.             box[row + 1][col] = 0
49.             //更新当前小鸟的坐标
50.             row = row + 1
51.           }
52.         }
53.       }
54.       //重新绘制地图
55.       this.drawCanvas()
56.     }
57.   },
58.   /**
59.    * 自定义函数 -- 方向键: 左
60.    */
61.   left: function() {
62.     //不在最左侧才考虑左移
63.     if (col > 0) {
64.       //如果左侧不是墙或箱子,可以移动小鸟
65.       if (map[row][col - 1] != 1 && box[row][col - 1] != 4) {
66.         //更新当前小鸟的坐标
67.         col = col - 1
68.       }
69.       //如果左侧是箱子
70.       else if (box[row][col - 1] == 4) {
71.         //箱子不在最左侧才能考虑推动
72.         if (col - 1 > 0) {
73.           //如果箱子左侧不是墙或箱子
74.           if (map[row][col - 2] != 1 && box[row][col - 2] != 4) {
75.             box[row][col - 2] = 4
76.             box[row][col - 1] = 0
77.             //更新当前小鸟的坐标
78.             col = col - 1
79.           }
80.         }
81.       }
82.       //重新绘制地图
83.       this.drawCanvas()
84.     }
85.   },
86.   /**
```

3 判断游戏成功

视频讲解

在 game.js 文件中添加自定义函数 isWin，用于判断游戏是否已经成功。对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.      * 自定义函数 -- 判断游戏是否成功
4.      */
5.    isWin: function() {
6.      //使用双重 for 循环遍历整个数组
7.      for (var i = 0; i < 8; i++) {
8.        for (var j = 0; j < 8; j++) {
9.          //如果有箱子没在终点
10.         if (box[i][j] == 4 && map[i][j] != 3) {
11.           //返回 false,表示游戏尚未成功
12.           return false
13.         }
14.       }
15.     }
16.     //返回 true,表示游戏成功
17.     return true
18.   },
19. })
```

上述代码的判断逻辑是只要有一个箱子没有在终点位置就判断游戏尚未成功。

然后在 game.js 中添加自定义函数 checkWin，要求一旦游戏成功就弹出提示对话框。对应的 JS(pages/game/game.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.      * 自定义函数 -- 游戏成功处理
4.      */
5.    checkWin: function() {
6.      if (this.isWin()) {
7.        wx.showModal({
8.          title: '恭喜',
9.          content: '游戏成功!',
10.         showCancel: false
11.       })
12.     }
13.   },
14. })
```

最后在 game.js 的 4 个方向键函数中追加关于游戏成功判断的函数，这里以 up 函数为例，对应的 JS(pages/game/game.js)代码片段修改如下：

```
1.  Page({
2.    /**
3.      * 自定义函数 -- 方向键：上
```

```
4.      */
5.    up: function() {
6.      //不在最顶端才考虑上移
7.      if (row > 0) {
8.        //如果上方不是墙或箱子,可以移动小鸟
9.        …
10.       //如果上方是箱子
11.       …
12.
13.       //重新绘制地图
14.       this.drawCanvas()
15.       //检查游戏是否成功
16.       this.checkWin()
17.     }
18.   },
19. })
```

其他 3 个方向的函数的修改方式和 up 函数完全一致，这里不再一一赘述。游戏成功后的画面如图 13-18 所示。

视频讲解

图 13-18　游戏成功提示画面

4 重新开始游戏

修改 game.wxml 代码，为“重新开始”按钮追加自定义函数的点击事件。

WXML（pages/game/game.wxml）代码片段修改如下：

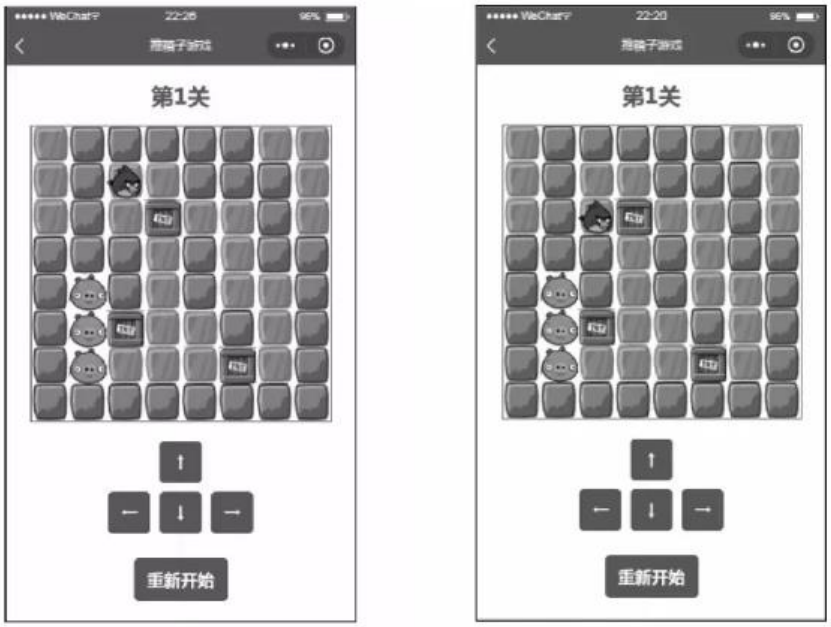
```
1.  <view class = 'container'>
2.    …
3.
4.    <!-- “重新开始”按钮 -->
5.    <button type = 'warn' bindtap = 'restartGame'>重新开始</button>
6.  </view>
```

在 game.js 文件中添加 restartGame 函数，用于重新开始游戏。

对应的 JS(pages/game/game.js)代码片段添加如下：

```
1.  Page({
2.    /**
3.      * 自定义函数 -- 重新开始游戏
4.      */
5.    restartGame: function() {
6.      //初始化地图数据
7.      this.initMap(this.data.level - 1)
8.      //绘制画布内容
9.      this.drawCanvas()
10.   },
11. })
```

此时页面效果如图 13-19 所示。



(a) 向上移动效果　　(b) 重新开始游戏

图 13-19　点击“重新开始”按钮效果

本实验到这里就全部完成了🎉，你完成的怎么样呢？要继续加油啊！😉