2025年夏季《移动软件开发》实验报告

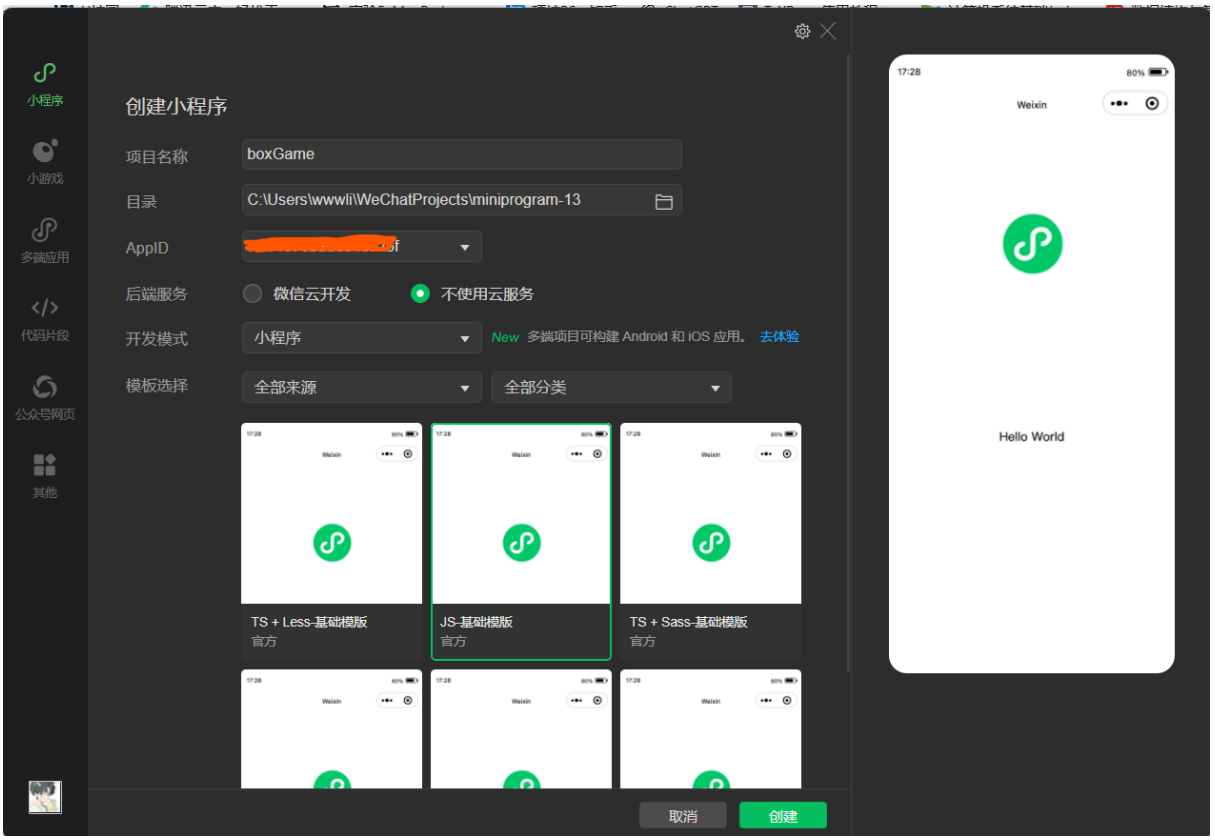
一、实验目标

1、学习使用快速启动模板创建小程序的方法；2、学习不使用模板手动创建小程序的方法。

二、实验步骤

2.1 项目创建

创建空白项目boxGame，选取模板为JS-基础模板。



2.2 页面配置

将app.json中pages属性中的"pages/logs/logs"删除，并且删除上一行的逗号。

删除utils文件夹及其内部所有内容。

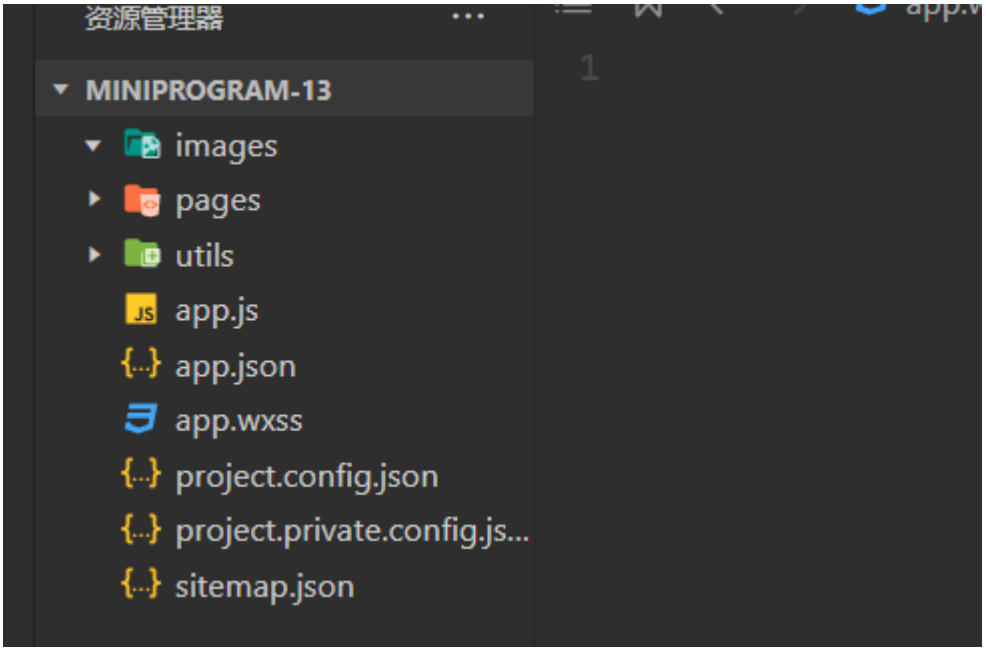
删除pages文件夹下的logs目录及其内部所有内容

删除index.wxml和index.wxss中的全部代码

删除index.js中的全部代码，并且输入关键词"page"，并让其自动补全函数。

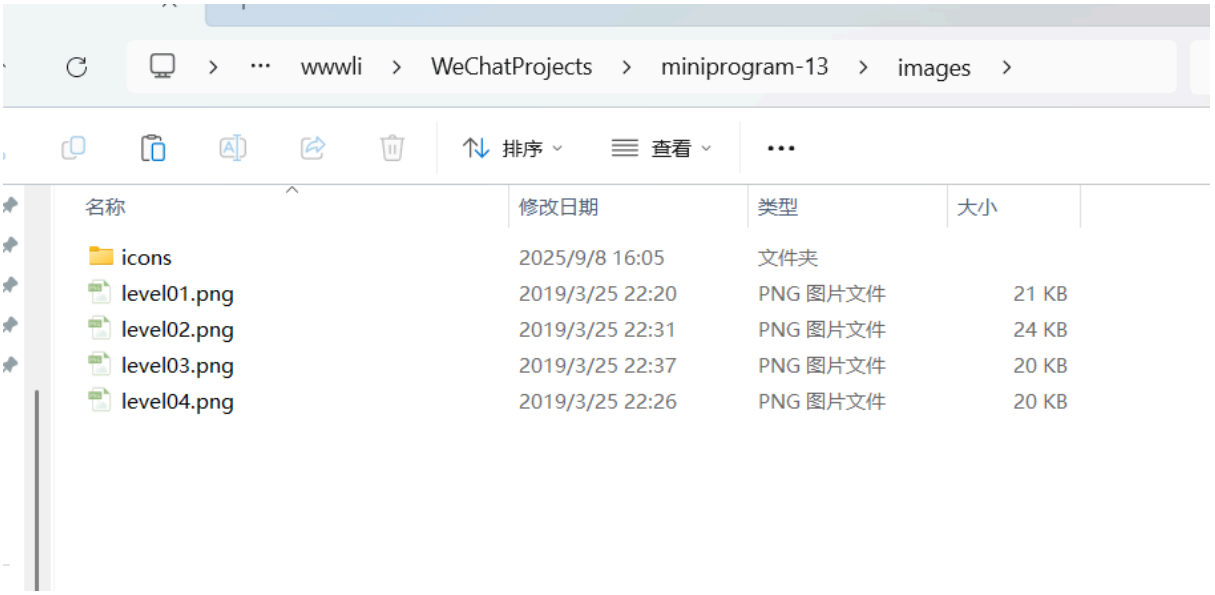
2.3 创建其他文件

新建images文件夹，用于存放图片素材，再创立utils文件夹，用于存放公共JS文件



2.4 添加图片

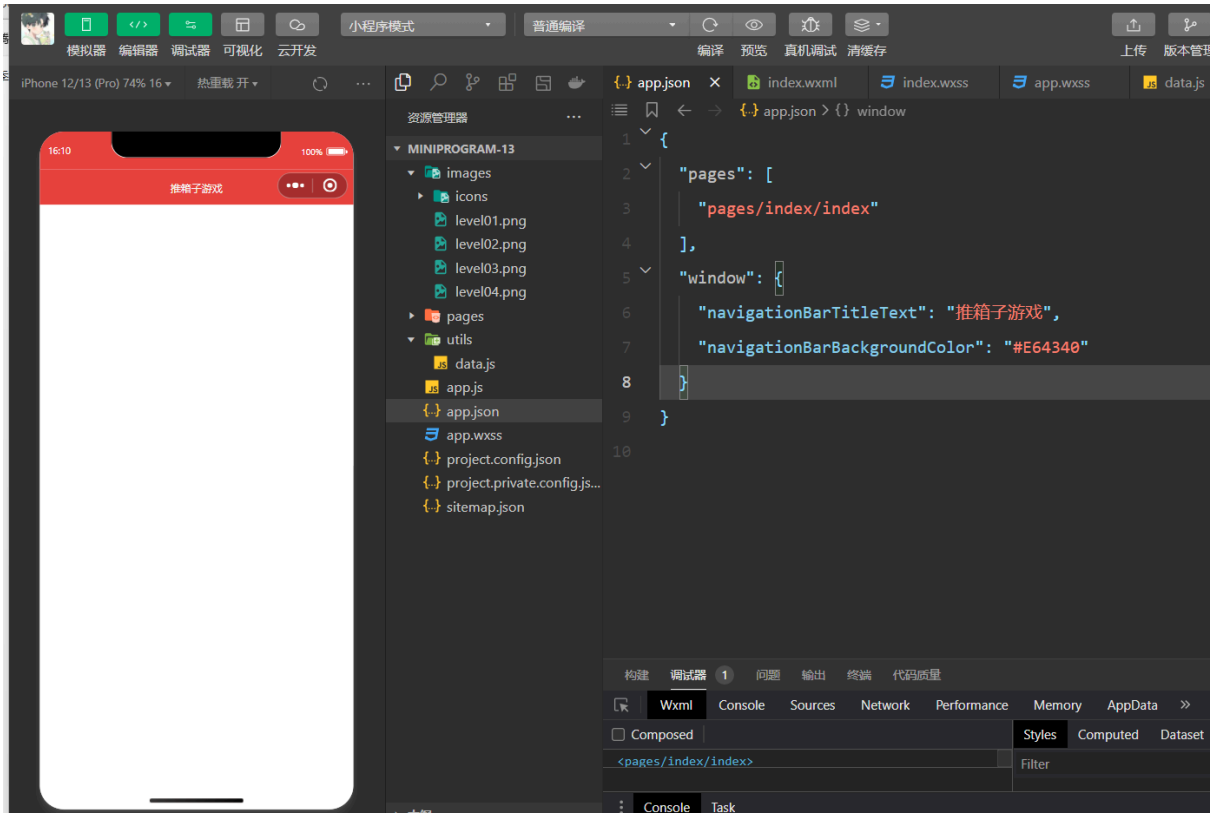
将下载的图片素材添加到images文件夹中



2.5 导航栏设计

在app.json文件中，对windows属性重新配置来定义导航栏效果，设计背景色为珊瑚红色，字体为白色，代码和实验结果如下：

```
{
  "pages": [
    "pages/index/index"
  ],
  "window": {
    "navigationBarTitleText": "推箱子游戏",
    "navigationBarBackgroundColor": "#E64340"
  }
}
```



2.6 公共样式设计

在app.wxss文件中添加下述代码，从而使得界面更加美观。

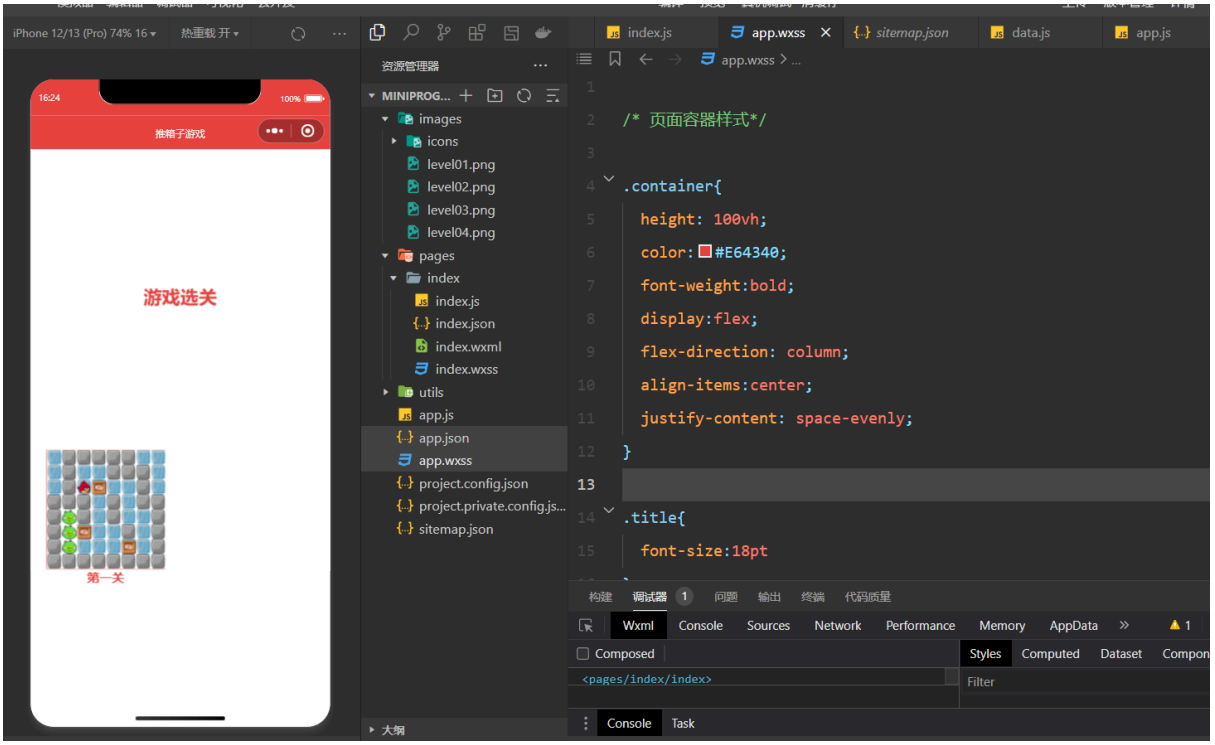
```
/* 页面容器样式*/

.container{
  height: 100vh;
  color:#E64340;
  font-weight:bold;
  display:flex;
  flex-direction: column;
  align-items:center;
  justify-items: space-evenly;
}

.title{
  font-size:18pt;
}
```

2.7 首页设计

我们在pages文件夹中的index.wxss和index.wxml文件中编写如下代码，从而得到一个初步的小程序。

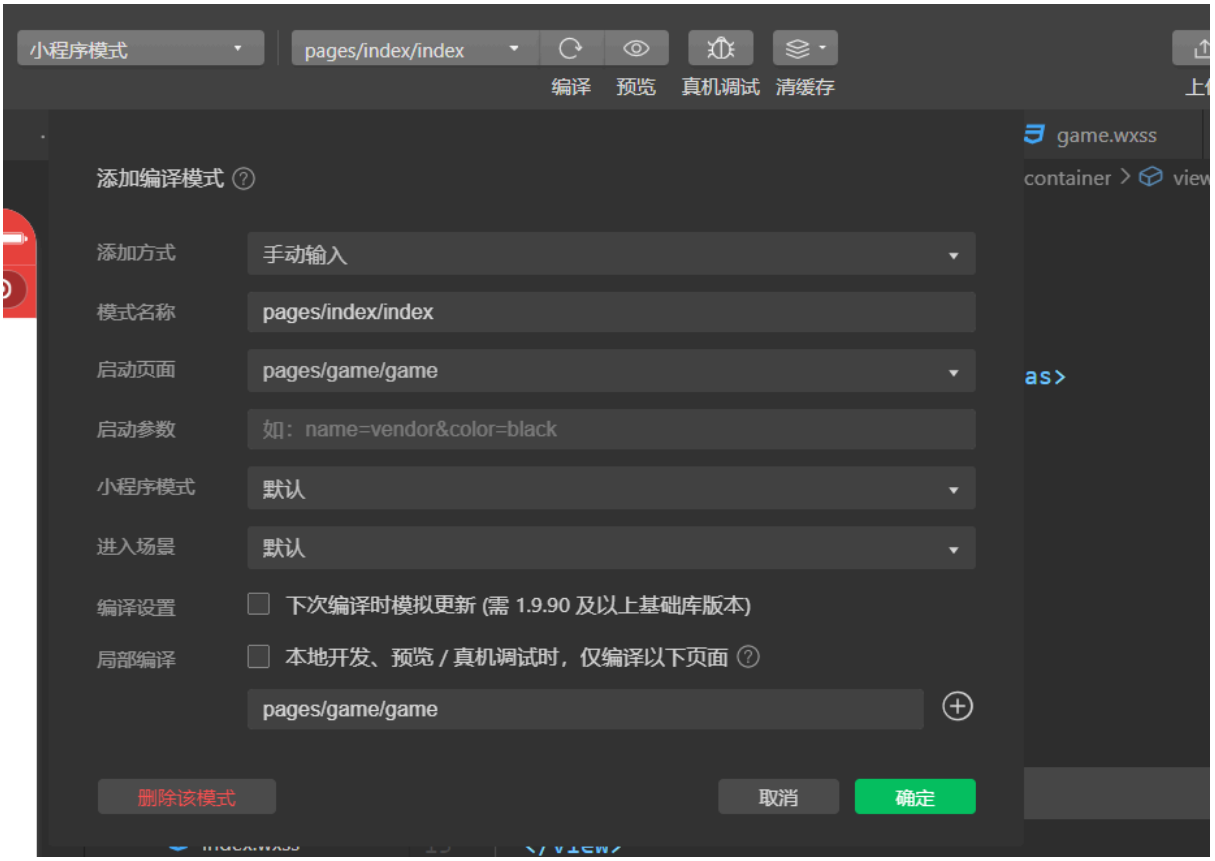
```
//index.wxss
.levelBox{
  width:100%;
}

.box{
  width:50%;
  float:left;
  margin:20rpx 0;
  display:flex;
  flex-direction: column;
  align-items: center;
}

image{
  width:300rpx;
  height:300rpx;
}
```

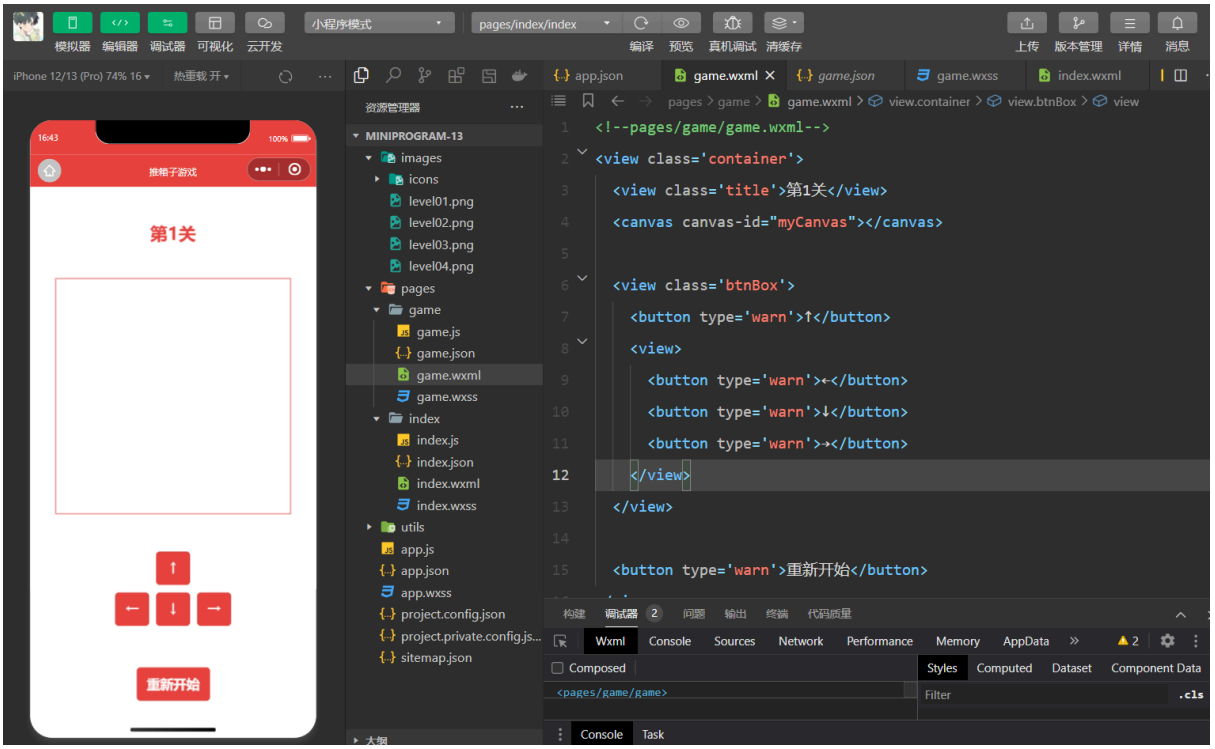
```
//index.wxml
<view class='container'>
  <view class='title'>游戏选关</view>
  <view class='levelBox'>
    <view class='box'>
      <image src='/images/level01.png'></image>
      <text>第一关</text>
    </view>
  </view>
</view>
```



2.8 游戏界面设计

按照参考教程先在pages文件夹中添加game页面，然后修改game.wxss和game.wxml文件，并且在开发工具顶端选择“普通编译”下的“添加编译模式”，添加临时测试参数level=0,操作过程和结果如下图所示：


小程序模式
pages/index/index
编译
预览
真机调试
清缓存
添加编译模式
添加方式
手动输入
模式名称
pages/index/index
启动页面
pages/game/game
启动参数
如：name=vendor&color=black
小程序模式
默认
进入场景
默认
编译设置
下次编译时模拟更新 (需 1.9.90 及以上基础库版本)
局部编译
本地开发、预览 / 真机调试时，仅编译以下页面
pages/game/game
删除该模式
取消
确定


第1关
重新开始
MINIPROGRAM-13
images
icons
level01.png
level02.png
level03.png
level04.png
pages
game
game.js
game.json
game.wxml
game.wxss
index
index.js
index.json
index.wxml
index.wxss
utils
app.js
app.json
app.wxss
project.config.json
project.private.config.js...
sitemap.json
<!--pages/game/game.wxml-->
<view class='container'>
<view class='title'>第1关</view>
<canvas canvas-id="myCanvas"></canvas>
<view class='btnBox'>
<button type='warn'>↑</button>
<view>
<button type='warn'>←</button>
<button type='warn'>↓</button>
<button type='warn'>→</button>
</view>
</view>
<button type='warn'>重新开始</button>
Wxml
Console
Sources
Network
Performance
Memory
AppData
Styles
Computed
Dataset
Component Data

2.9 公共逻辑实现

在公共JS文件（utils/data.js）中配置游戏地图的数据，分别使用map1~map4代表4个不同关卡的地图数据，以二维数据形式存放。然后再在data.js暴露出数据出口，代码如下：

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

接着在game.js文件中添加对data.js文件的相对路径的引用，代码如下：

```
var data=require('../../utils/data.js')
```

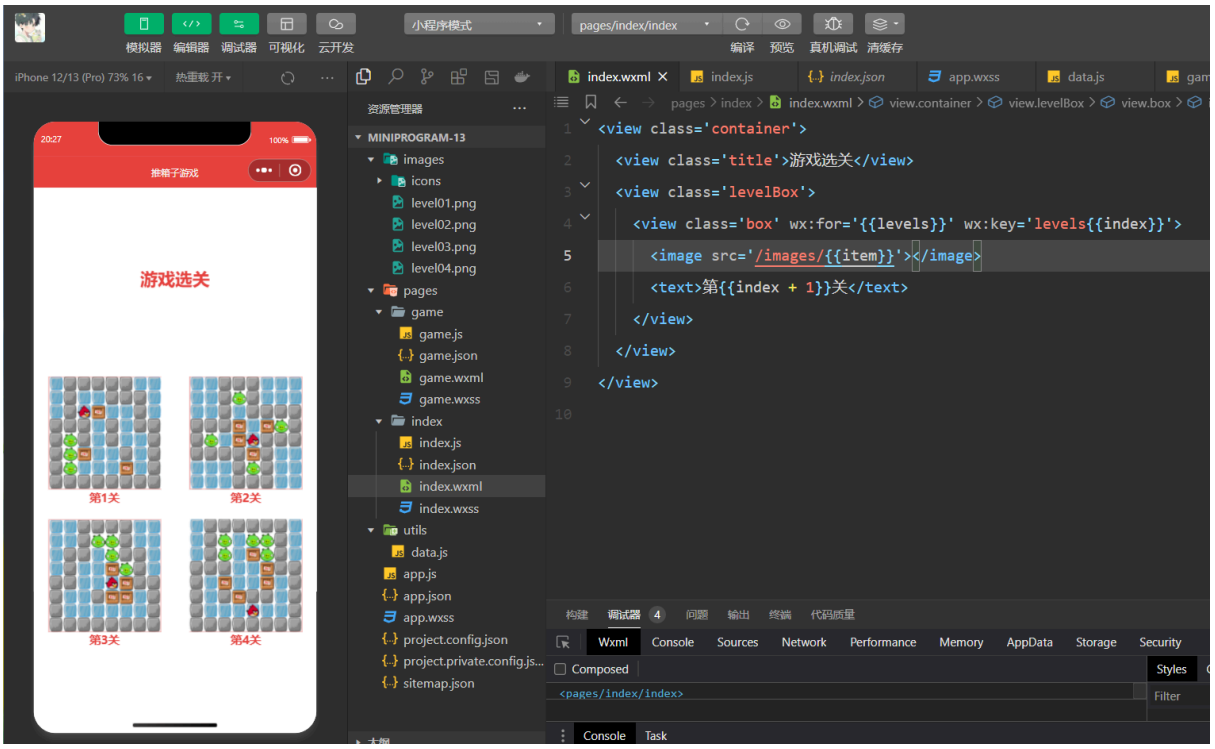
2.10 关卡列表实现

在index.js文件中加载关卡图片的代码，接着在index.wxml中关卡对应的<view>组件中添加wx:for属性循环显示关卡列表数据和图片。代码和运行结果如下：

```
//index.js
data:{
    levels:[
      'level01.png',
      'level02.png',
      'level03.png',
      'level04.png'
    ]
  },
```

```
//index.wxml
<view class='container'>
  <view class='title'>游戏选关</view>
  <view class='levelBox'>
    <view class='box' wx:for='{{levels}}' wx:key='levels{{index}}'>
      <image src='/images/{{item}}'></image>
      <text>第{{index + 1}}关</text>
    </view>
```

```
  </view>
</view>
```



2.11 跳转游戏界面

给关卡添加了自定义点击事件函数chooseLevel，并且使用data-level属性携带关卡图片下标信息。在index.js文件中添加chooseLevel函数，通过相关代码就可以实现点击图片跳转到game界面。由于跳转界面需要点击才能进行，所以结果就不进行展示了。代码如下：

```
//index.wxml
<view class='container'>
  <view class='title'>游戏选关</view>
  <view class='levelBox'>
    <view class='box' wx:for='{{levels}}' wx:key='levels{{index}}' bindtap
='chooseLevel' data-level='{{index}}'>
      <image src='/images/{{item}}'></image>
      <text>第{{index + 1}}关</text>
    </view>
```

```
  </view>
</view>
```

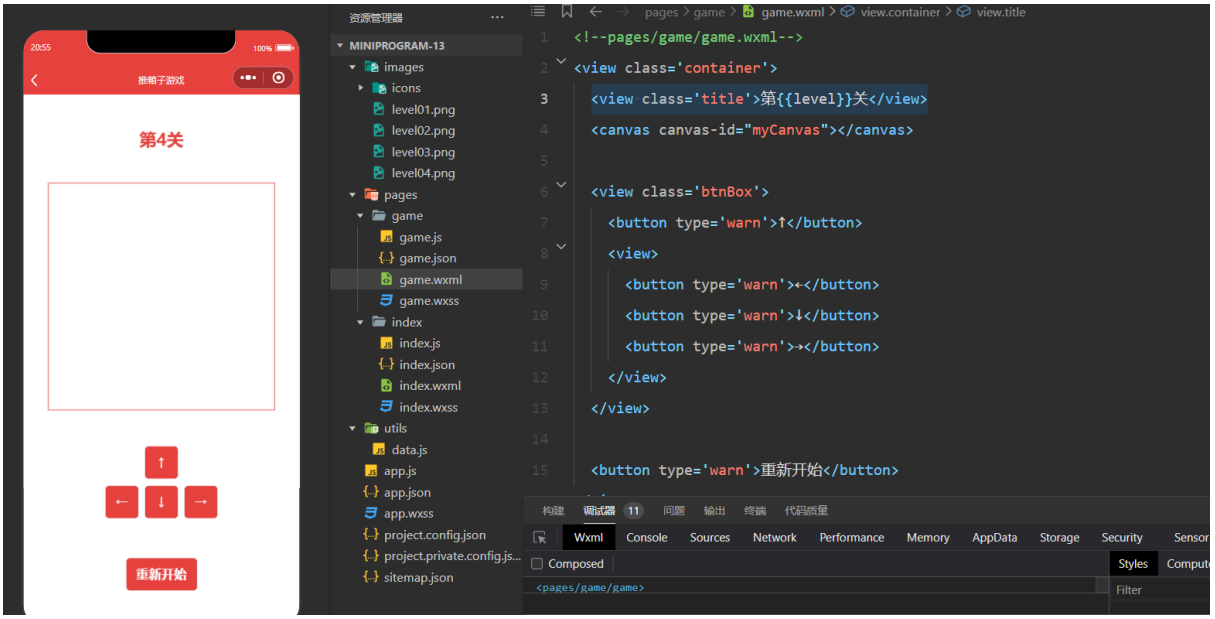
```
//index.js
chooseLevel:function(e){
  let level=e.currentTarget.dataset.level
  wx.navigateTo({
    url:'../game/game?level='+level
  })
},
```

2.12 显示关卡逻辑

在之前首页逻辑中，我们已经实现了页面跳转并且携带关卡对应的图片信息，现在只需要在游戏界面接受关卡信息即可。代码和运行结果如下：

```
//game.js
onLoad(options) {
  let level=options.level
  this.setData({
    level:parseInt(level)+1
  })
},
```

```
//game.wxml
<view class='title'>第{{level}}关</view>
```



2.13 游戏逻辑实现

首先在gane.js文件的顶端对一些游戏初始数据信息进行记录，例如地图图层数据和箱子图层数据；接着初始化游戏界面，将当前关卡对应的游戏地图数据更新到游戏初始数据中。在game.js文件中添加initMap函数用于初始化游戏地图数据。

```
//自定义函数--初始化地图数据
  initMap: function (level) {
    //读取原始的游戏地图数据
    let mapData = data.maps[level]
    //使用双重for循环遍历地图数据
    for (var i = 0; i < 8; i++) {
      for (var j = 0; j < 8; j++) {
        map[i][j] = mapData[i][j]
        box[i][j] = 0
        if (mapData[i][j] == 4) {
          box[i][j] = 4
          map[i][j] = 2
        } else if (mapData[i][j] == 5){
          map[i][j] = 2
          //记录小鸟的当前行和列
          row = i
          col = j
```

```
      }
    }
  }
},
```

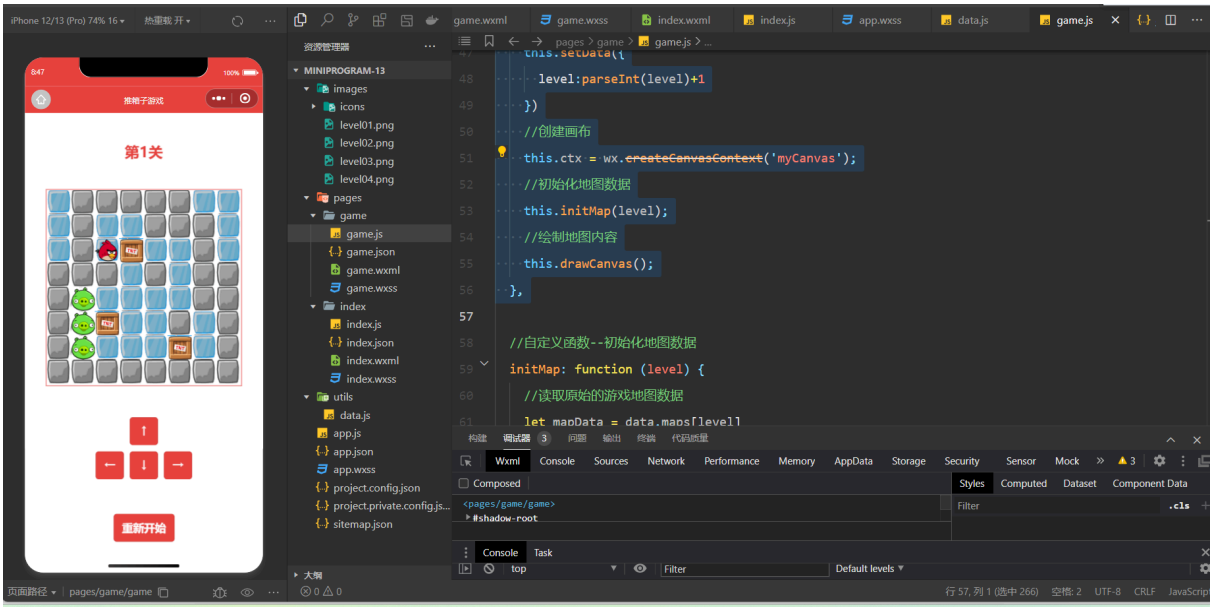
接着，我们添加自定义函数drawCanvas，将地图信息绘制到画布上。

```
drawCanvas: function () {
  let ctx = this.ctx
  // 清空画布
  ctx.clearRect(0, 0, 320, 320)
  // 使用双重 for 循环绘制 8x8 的地图
  for (var i = 0; i < 8; i++) {
    for (var j = 0; j < 8; j++) {
      // 默认是道路
      let img = 'ice'
      if (map[i][j] == 1) {
        img = 'stone'
      } else if (map[i][j] == 3) {
        img = 'pig'
      }
      // 绘制地图
      ctx.drawImage('/images/icons/' + img + '.png', j * w, i * w, w, w)
      if (box[i][j] == 4) {
        // 叠加绘制箱子
        ctx.drawImage('/images/icons/box.png', j * w, i * w, w, w)
      }
    }
  }
  // 叠加绘制小鸟
  ctx.drawImage('/images/icons/bird.png', col * w, row * w, w, w)
  ctx.draw()
},
```

最后在game.js文件中的onLoad函数中依次调用initMap和drawCanvas函数。代码如下：

```
onLoad:function (options) {
    let level=options.level
    this.setData({
      level:parseInt(level)+1
    })
    //创建画布
    this.ctx = wx.createCanvasContext('myCanvas');
    //初始化地图数据
    this.initMap(level);
    //绘制地图内容
    this.drawCanvas();
  },
```

经过上述步骤后，每个关卡都可以显示其地图，所得到的结果如下：



2.14 方向键逻辑实现

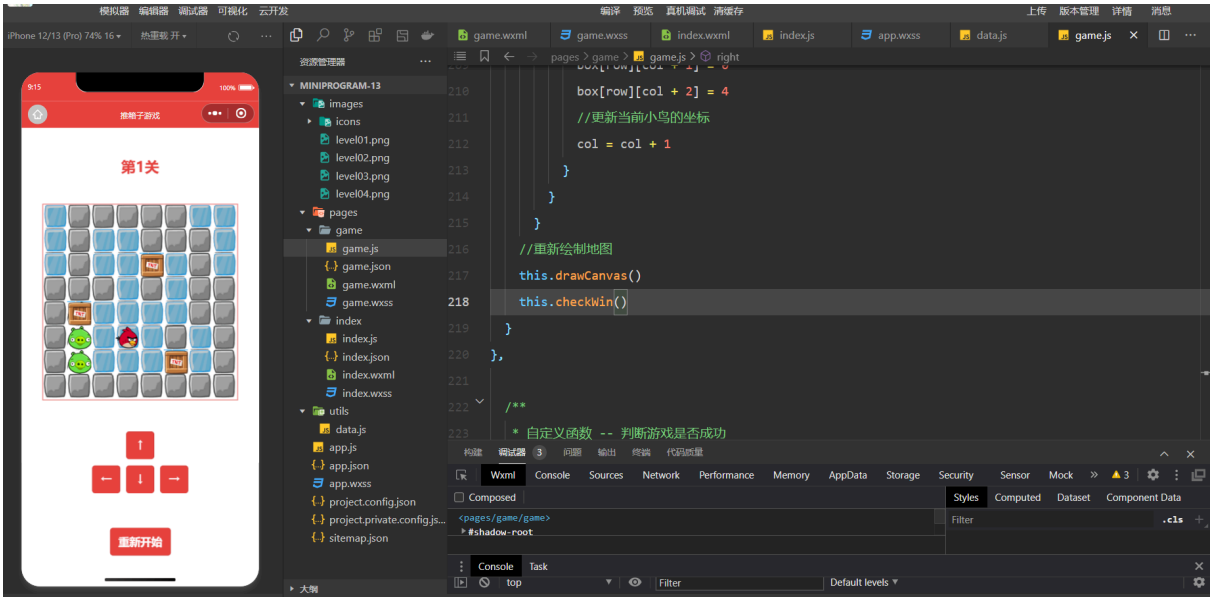
在game.js文件中添加up、down、left和right函数，实现小鸟的上、下、左、右移动，每次点击在无障碍的条件下移动一格。代码如下：

```
/**
  * 自定义函数--方向键：上
  */
```

```
up: function () {
  //不在最顶端才考虑上移
  if (row > 0) {
    //如果上方不是墙或箱子，可以移动小鸟
    if (map[row - 1][col] != 1 && box[row - 1][col] != 4) {
      //更新当前小鸟的坐标
      row = row - 1
    }
    //如果上方是箱子
    else if (box[row - 1][col] == 4) {
      //箱子不在最顶端才能考虑推动
      if (row - 1 > 0) {
        //如果箱子上方不是墙或箱子
        if (map[row - 2][col] != 1 && box[row - 2][col] != 4) {
          box[row - 1][col] = 0
          box[row - 2][col] = 4
          //更新当前小鸟的坐标
          row = row - 1
        }
      }
    }
    //重新绘制地图
    this.drawCanvas()
  }
},
down: function() {
  //不在最 bottom 端才考虑下移
  if (row < 7) {
    //如果下方不是墙或箱子，可以移动小鸟
    if (map[row + 1][col] != 1 && box[row + 1][col] != 4) {
      //更新当前小鸟的坐标
      row = row + 1
    }
    //如果下方是箱子
    else if (box[row + 1][col] == 4) {
      //箱子不在最 bottom 端才能考虑推动
      if (row + 1 < 7) {
        //如果箱子下方不是墙或箱子
```

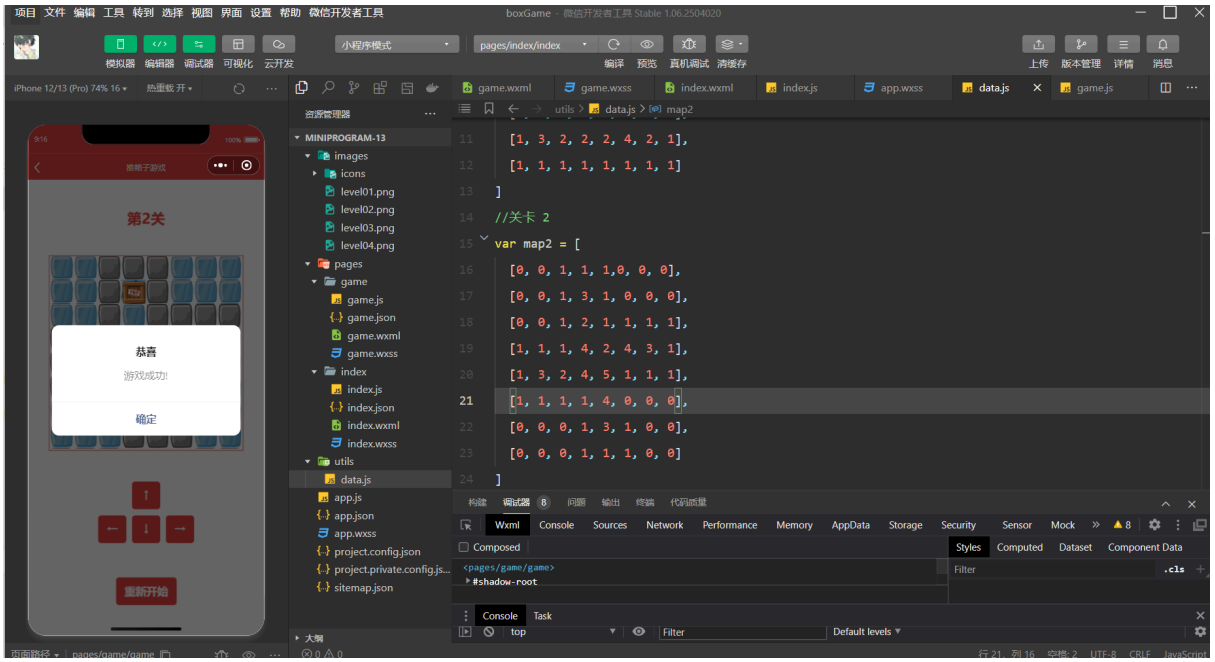
```
        if (map[row + 2][col] != 1 && box[row + 2][col] != 4) {
          box[row + 2][col] = 4
          box[row + 1][col] = 0
          //更新当前小鸟的坐标
          row = row + 1
        }
      }
    }
   //重新绘制地图
   this.drawCanvas()
   }
 },
 /**
  * 自定义函数--方向键：左
  */
 left: function() {
   //不在最左侧才考虑左移
   if (col > 0) {
     //如果左侧不是墙或箱子，可以移动小鸟
     if (map[row][col - 1] != 1 && box[row][col - 1] != 4) {
       //更新当前小鸟的坐标
       col = col - 1
     }
     //如果左侧是箱子
     else if (box[row][col - 1] == 4) {
       //箱子不在最左侧才能考虑推动
       if (col - 1 > 0) {
         //如果箱子左侧不是墙或箱子
         if (map[row][col - 2] != 1 && box[row][col - 2] != 4) {
           box[row][col - 1] = 0
           box[row][col - 2] = 4
           //更新当前小鸟的坐标
           col = col - 1
         }
       }
     }
   //重新绘制地图
   this.drawCanvas()
```

```
    }
  },
  right: function() {
    //不在最右侧才考虑右移
    if (col < 7) {
      //如果右侧不是墙或箱子，可以移动小鸟
      if (map[row][col + 1] != 1 && box[row][col + 1] != 4) {
        //更新当前小鸟的坐标
        col = col + 1
      }
      //如果右侧是箱子
      else if (box[row][col + 1] == 4) {
        //箱子不在最右侧才能考虑推动
        if (col + 1 < 7) {
          //如果箱子右侧不是墙或箱子
          if (map[row][col + 2] != 1 && box[row][col + 2] != 4) {
            box[row][col + 1] = 0
            box[row][col + 2] = 4
            //更新当前小鸟的坐标
            col = col + 1
          }
        }
      }
    //重新绘制地图
    this.drawCanvas()
  }
},
```



2.15 判断游戏是否成功

在game.js文件中添加isWin和checkWin函数，用于判断游戏是否成功，如果成功会弹出游戏成功的对话框，如下图所示。



2.16 重新开始游戏

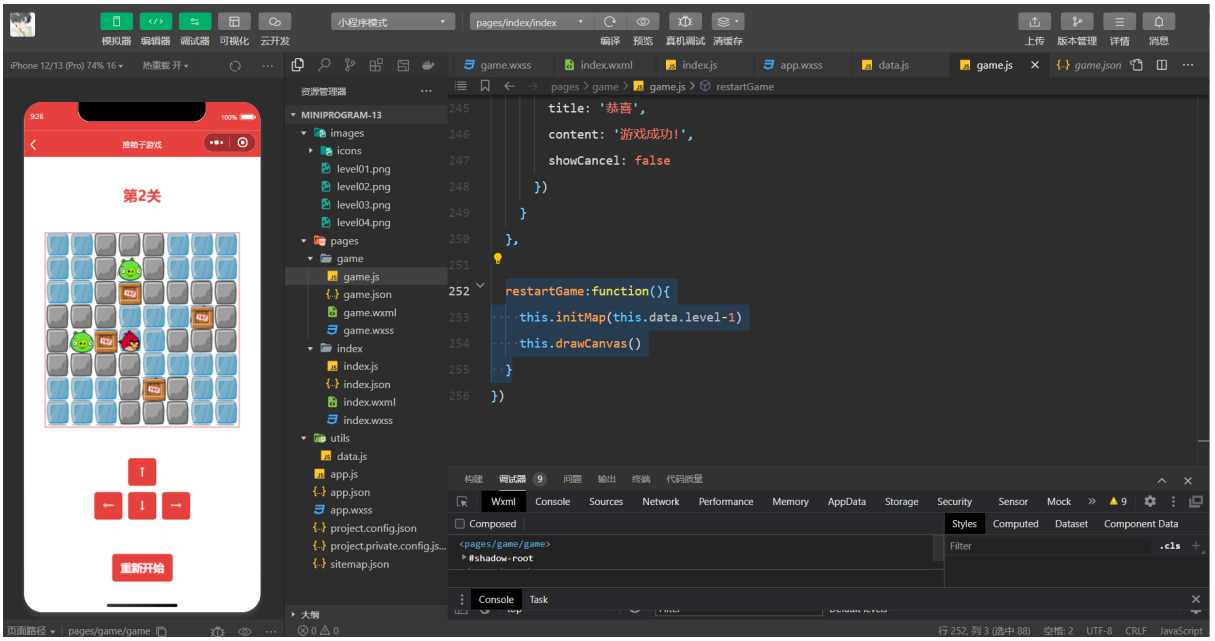
为game.wxml文件中的”重新开始“按钮添加自定义函数的点击事件，在game.js添加restartGame函数，用于重新开始游戏。

```
//game.wxml
<button type='warn' bindtap='restartGame'>重新开始</button>
```

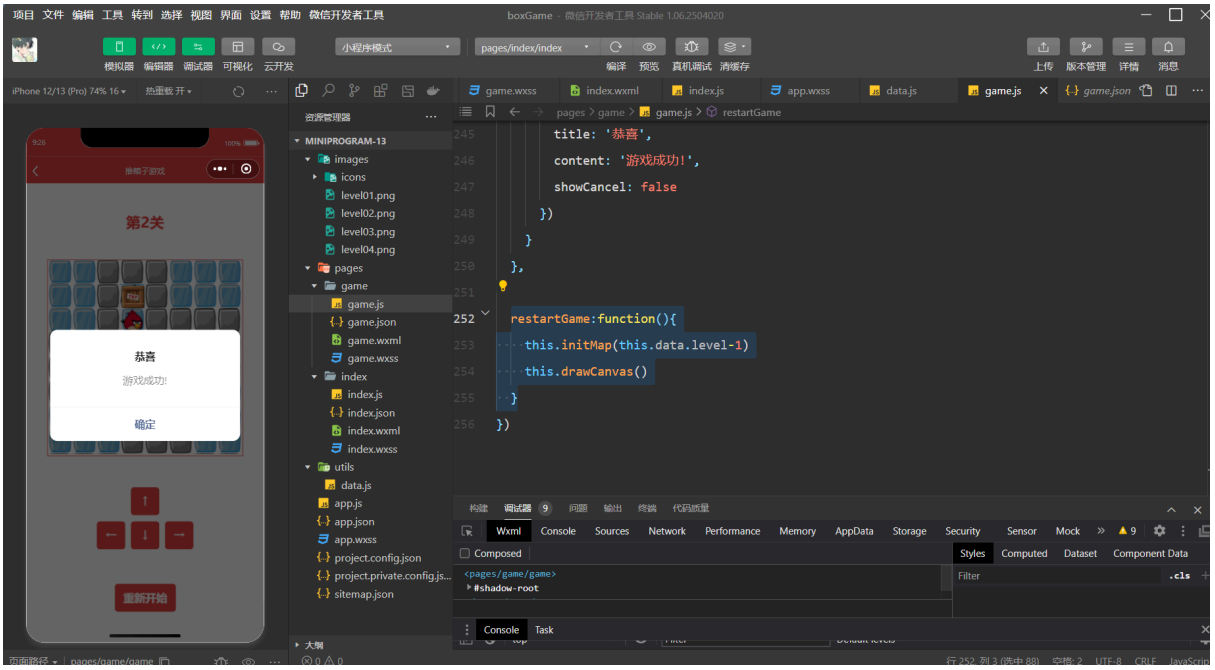
```
//game.js
restartGame:function(){
  this.initMap(this.data.level-1)
  this.drawCanvas()
 }
```

三、程序运行结果

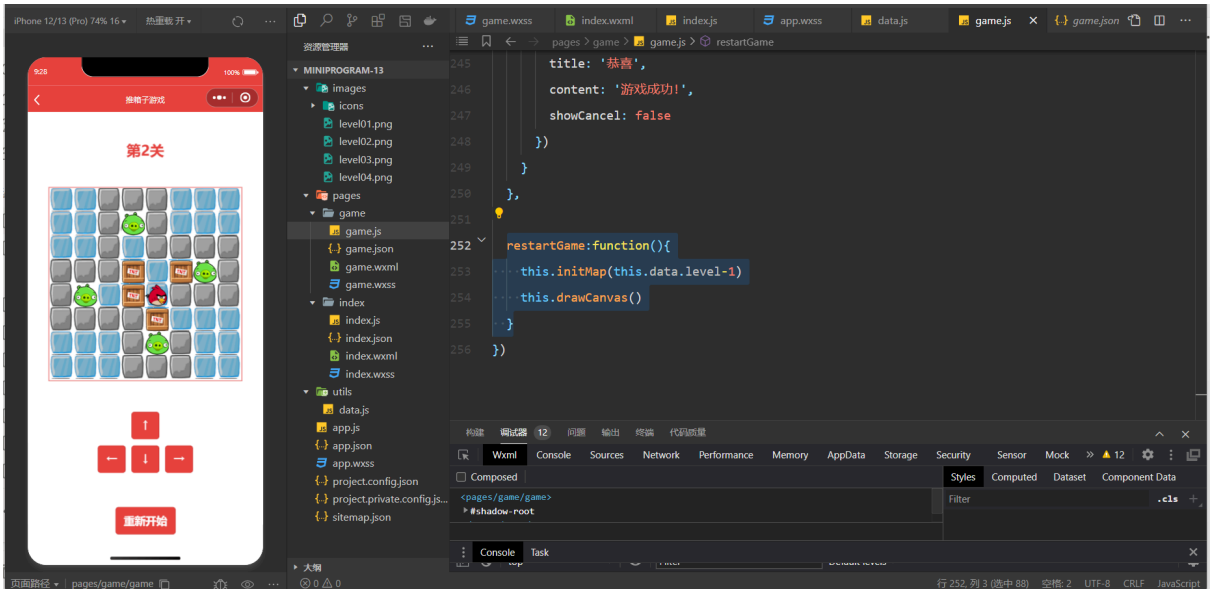
现在小程序就完成了，可以点击上下左右键进行移动。



当通关后会显示游戏成功



点击重新开始按钮后，游戏复原：



四、问题总结与体会

问题总结：

参考教程十分详细，没有遇到什么问题。

心得体会

我学会了使用二维数组绘制地图，驱动Canvas的重绘，同时我也了解了推箱子游戏的制作思路，实现玩家和箱子的移动逻辑，理解了边界条件的处理，确保游戏的规则可

以被正确的遵守，避免了意外情况的出现。同时使用isWin函数遍历箱子和目标点得到状态来判断游戏是否成功。