2025 年夏季《移动软件开发》实验报告

姓名和学号？	**聂宇航，23170001072**
本实验属于哪门课程？	中国海洋大学 25 夏《移动软件开发》
实验名称？	实验 6：推箱子游戏
博客地址？	《移动软件开发》实验六实验报告-CSDN 博客
Github 仓库地址？	这个是《移动软件开发》这门课的实验代码与报告

（备注：将实验报告发布在博客、代码公开至 github 是 加分项，不是必须做的）

一、实验目标

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 和绘图 API。

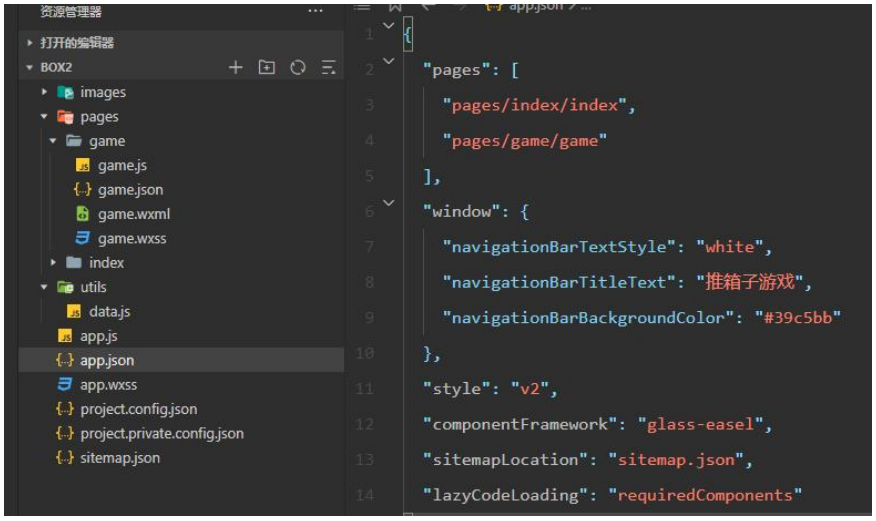
二、实验步骤

文件准备

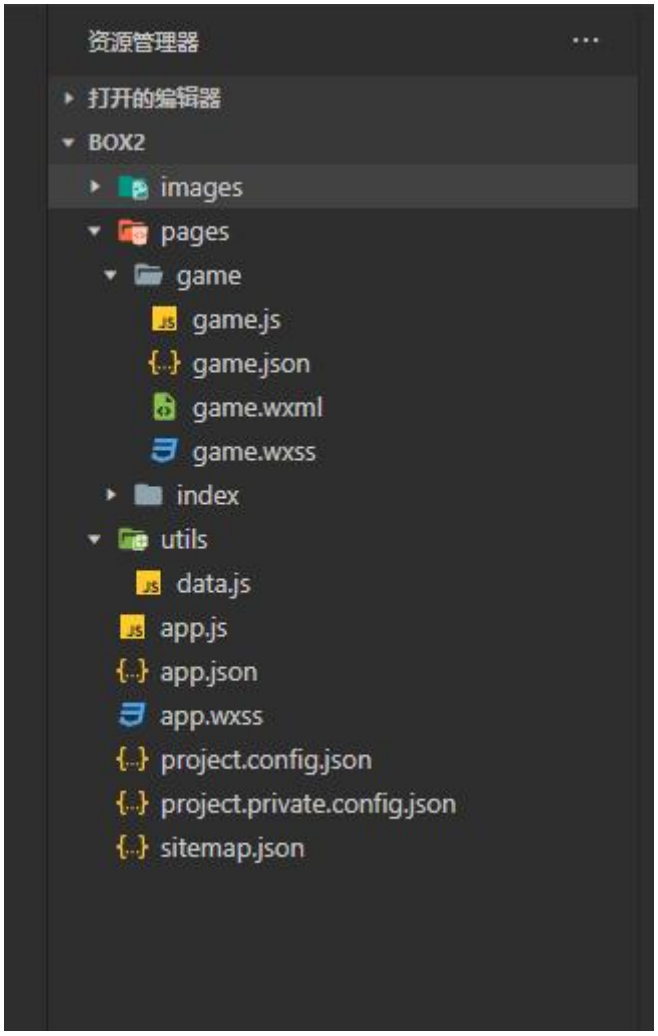
首先我们先把文件们都准备好

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”.



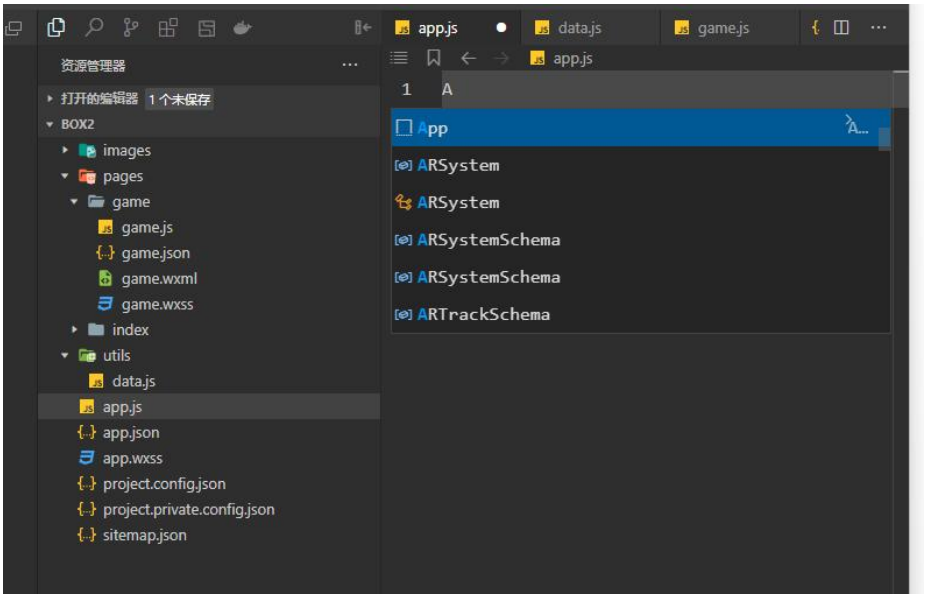
2.按快捷键 Ctrl+S 保存修改后会在 pages 文件夹下自动生成 game 目录。



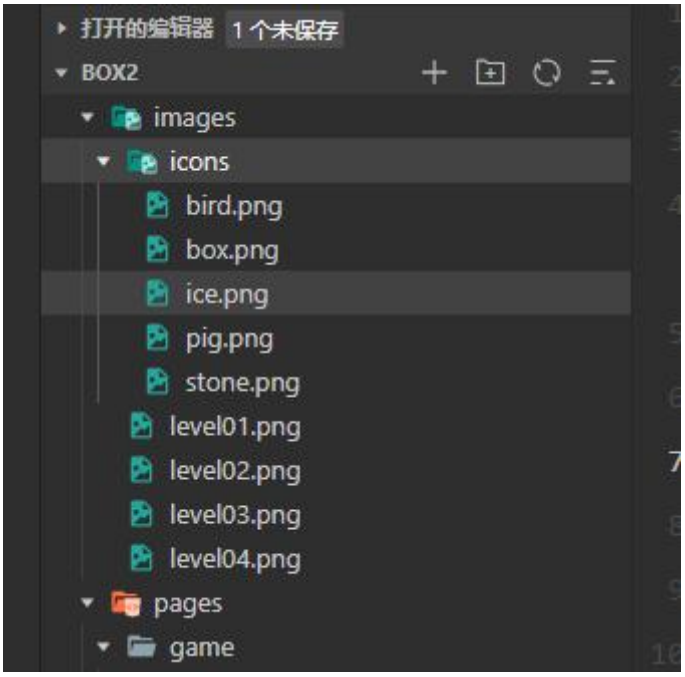
然后我们像前面的实验一样，把一开始默认的代码都删除或修改： (1)删除 utils 文件夹及其内部所有内容。 (2)删除 pages 文件夹下的 logs 目录及其内部所有内容, (3)删除 index.wxml 和 index.wxss 中的全部代码。 (4)删除 index.js 中的全部代码,并且输人关键词 page 找到第二个选项按回车键让其自动补全雨数。 (5)

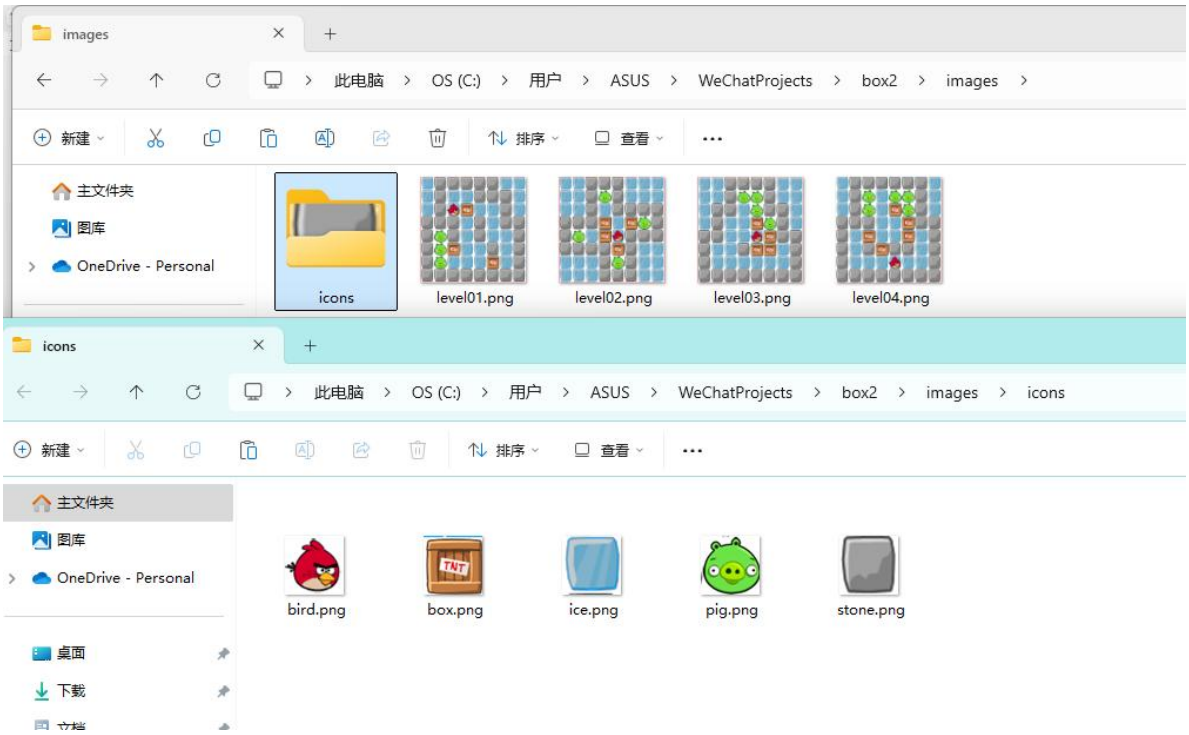
删除 app.wxss 中的全部代码。 (6)删除 app.js 中的全部代码,并且输人关键词 app 找到第一个选项按回车键让其自动补全函数。

选蓝色的：

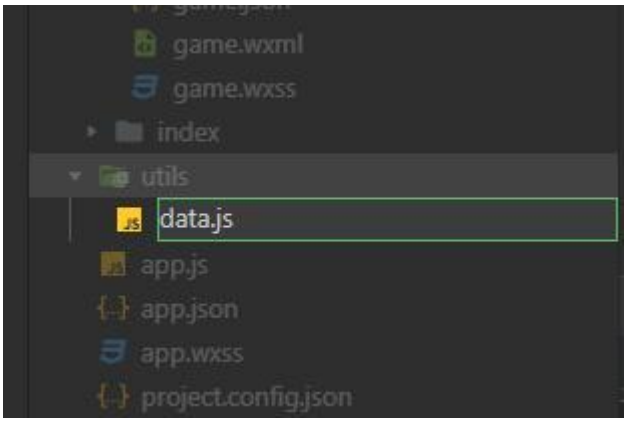


然后我们创建两个文件夹,并分别命名为 images 和 utils。 images:用于存放图片素材；utils:用于存放公共 JS 文件。 在 https://gaopursuit.oss-cn-beijing.aliyuncs.com/course/mobileDev/boxgame_images.zip 中下载图片文件，放入 images 文件夹





右击 utils 文件夹,选择“新建”,输人 data.js 后按回车键创建公共文件 data.js



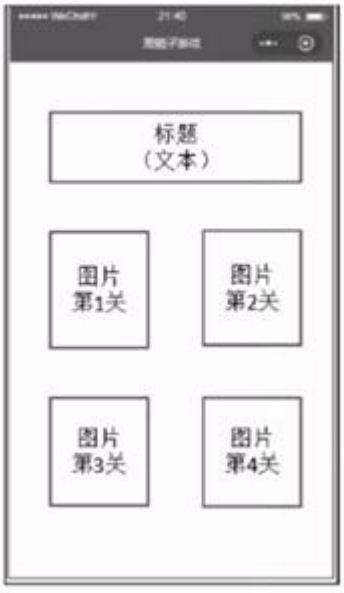
页面设计

在 app.json 中可以进行导航栏设置，我们将名字改为“推箱子游戏”，搞成青（#39c5bb）底白字：

```
{
  "pages": [
    "pages/index/index",
    "pages/game/game"
```

```
  ],
  "window": {
    "navigationBarTextStyle": "white",
    "navigationBarTitleText": "推箱子游戏",
    "navigationBarBackgroundColor": "#39c5bb"
  },
  "style": "v2",
  "componentFramework": "glass-easel",
  "sitemapLocation": "sitemap.json",
  "lazyCodeLoading": "requiredComponents"
}
```

在一开始的选关页面，我们预设了 4 个关卡：



所以我们按照如下程序对页面进行设计：

index.wxml 文件：

```
<view class='container'>
```

```
<!--标题--><view class='title'>游戏选关</view>

<!--关卡列表-->

<view class='levelBox'>

  <view class ='box'>

  <image src='/images/level01'></image>

  <text>第 1 关</text>

  <view class ='box'>

  <image src='/images/level02'></image>

  <text>第 2 关</text>

  <view class ='box'>

  <image src='/images/level03'></image>

  <text>第 3 关</text>

  <view class ='box'>

  <image src='/images/level04'></image>

  <text>第 4 关</text>

</view>




</view >

</view>
```

index.wxss 文件:

```
/*页面容器样式*/

.container{

  height:100vh;
```

```
  color:#E64340;
  font-weight:bold;
  display: flex;
  flex-direction: column;
  align-items:center;
  justify-content:space-evenly;
}
  /*顶端标题样式*/
.title {
  font-size:18pt;
}
/*关卡列表区域*/
.levelBox{
width:100%;
}
/*单个关卡区域*/.box {
width:50%;
float: left;
margin:20rpx0;
display:flex;
flex-direction: column;
align-items:center;
}
/*选关图片*/image{
  width:300rpx;
  height :300rpx;
```

```
}
```

这样做我们就可以看到选关页面的雏形，但按不了按钮：

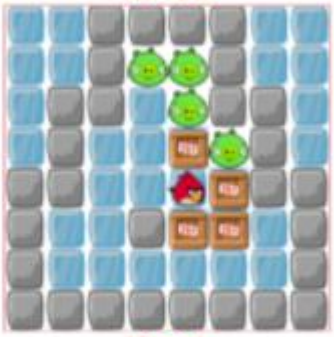
15:03 100%

推箱子游戏

游戏选关

第1关

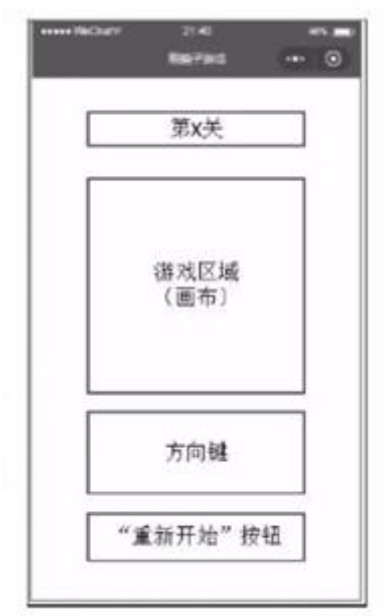
第2关



第3关

第4关

然后我们要做到如下的游戏页面：



所以我们按照如下程序对页面进行设计：

game.wxml:

```
<view class='container'>

<!--关卡提示--><view class='title'>第 1 关</view>

<!--游戏画布-->

<canvas canvas-id='myCanvas'></canvas >

<!--方向键-->

<view class='btnBox'>

<button type= 'warn'>↑</button >

<view >

<button type= 'warn'>←</button >

<button type= 'warn'>↓</button >

<button type='warn'>→</button >

</view>

</view>
```

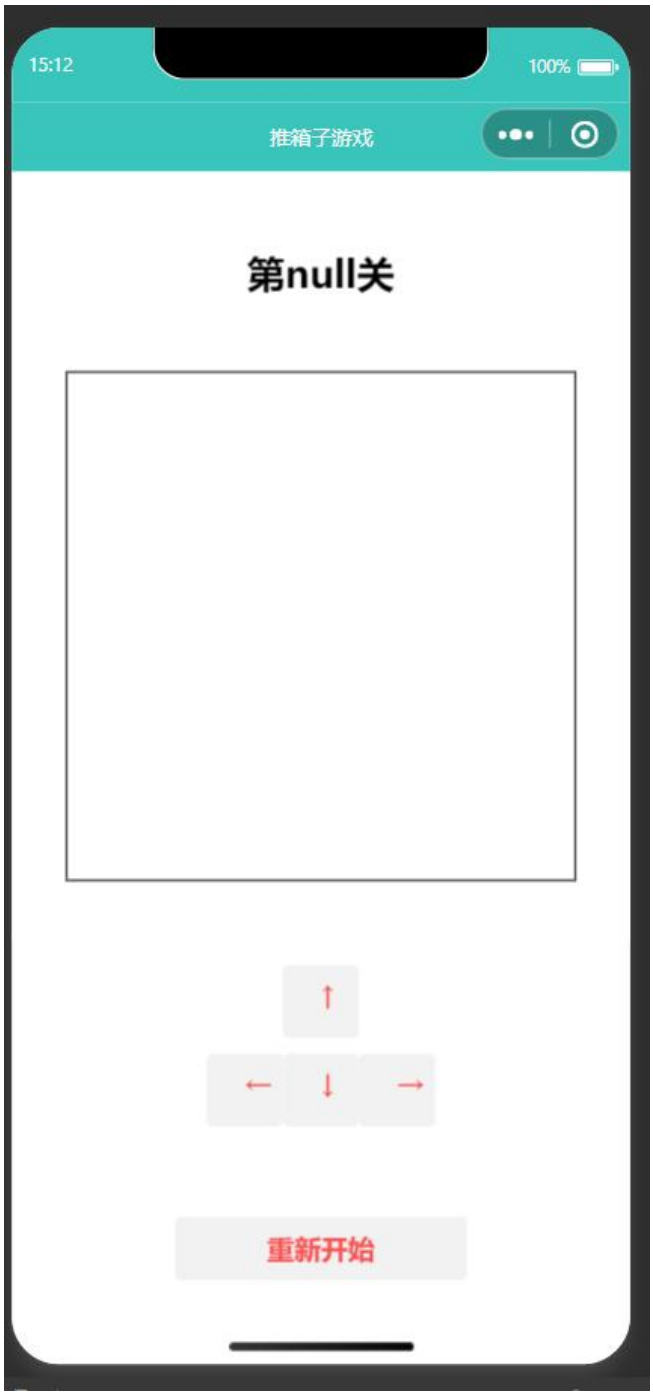
```
<!--“重新开始”按钮--><button type='warn'>重新开始</button></view>
```

game.wxss:

```
/*游戏画布样式*/

/*页面容器样式*/

.container{

  height:100vh;

  color:black;

  font-weight:bold;

  display: flex;

  flex-direction: column;

  align-items:center;

  justify-content:space-evenly;

}

  /*顶端标题样式*/

.title {

  font-size:18pt;

}

canvas{

border:1rpx solid;width:320px;height:320px;

}

/*方向键按钮整体区域*/.btnBox{



display:flex;

flex-direction:column;

align-items:center;
```

```
}
/*方向键按钮第二行*/
.btnBox view{
   display:flex;
flex-direction:row;
}
/*所有方向键按钮*/
.btnBox button{


width:90rpx;height:90rpx;
}
/*所有按钮样式*/
button{
margin:10rpx;
}
```

这样游戏页面也做好了：



逻辑实现

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,页而设计 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。 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;

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

```
var map1 = [
    [0,1,1,1,1,1,0,0],
    [0,1,2,2,1,1,1,0],
    [0,1,5,4,2,2,1,0],
    [1,1,1,2,1,2,1,1],
    [1,3,1,2,1,2,2,1],
    [1,3,4,2,2,1,2,1],
    [1,3,2,2,2,4,2,1],
    [1,1,1,1,1,1,1,1]
]

var map2 = [
    [0,0,1,1,1,0,0,0],
    [0,0,1,3,1,0,0,0],
    [0,0,1,2,1,1,1,1],
    [1,1,1,4,2,4,3,1],
    [1,3,2,4,5,1,1,1],
    [1,1,1,1,4,1,0,0],
    [0,0,0,1,3,1,0,0],
    [0,0,0,1,1,1,0,0]
]

var map3 = [
    [0,0,1,1,1,1,0,0],
    [0,0,1,3,3,1,0,0],
```

```
  [0,1,1,2,3,1,1,0],
  [0,1,2,2,4,3,1,0],
  [1,1,2,2,5,4,1,1],
  [1,2,2,1,4,4,2,1],
  [1,2,2,2,2,2,2,1],
  [1,1,1,1,1,1,1,1]
]

var map4 = [
  [0,1,1,1,1,1,1,0],
  [0,1,3,2,3,3,1,0],
  [0,1,3,2,4,3,1,0],
  [1,1,1,2,2,4,1,1],
  [1,2,4,2,2,4,2,1],
  [1,2,1,4,1,1,2,1],
  [1,2,2,2,5,2,2,1],
  [1,1,1,1,1,1,1,1]
]
```

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 8x8 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:

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

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。

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:

```
<view class='container'>

<!--标题--><view class='title'>游戏选关</view>

<!--关卡列表-->

<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>
```

我们在中间加入了一段循环

`wx:for='{{levels}}'`：循环渲染 `levels` 数组（在 `index.js` 的 `data` 里定义的）。 例如：`['level01.png','level02.png','level03.png']`。

`wx:key='levels{{index}}'`：用于提高渲染性能

`bindtap='chooseLevel'`：点击事件，调用 JS 里的 `chooseLevel` 方法。

`data-level='{{index}}'`：自定义属性，把当前循环的关卡下标传递给 `chooseLevel`。

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:

```
<view class='container'>

<!--标题--><view class='title'>游戏选关</view>

<!--关卡列表-->

<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>
```

加入了 chooseLevel 函数，我们来看看其是干什么的：

```
    chooseLevel : function(e){

      let level = e.currentTarget.dataset.level

      wx.navigateTo({

        url: '../game/game?level='+level,

       })

    },
```

事件绑定

在 `wxml` 里，`<view ... bindtap="chooseLevel" data-level="{{index}}">`。

点击某一关的时候，会触发 `chooseLevel` 方法。

微信小程序里，事件对象 `e` 里带有 `dataset`，里面存放着 `data-*` 的值。 所以 `e.currentTarget.dataset.level` 就是关卡索引（0,1,2...）。

获取关卡号

```
let level = e.currentTarget.dataset.level
```

比如点了“第 2 关”，`index = 1`，那么 `level = 1`。

页面跳转

```
wx.navigateTo({

  url: '../game/game?level=' + level,

})
```

使用微信小程序的 API 跳转到 `../game/game` 页面。

同时把 `level` 当做参数拼接到 URL 上： `../game/game?level=1`。

在 `game.js` 的 `onLoad(options)` 里，就能拿到 `options.level`，确定要加载哪一关。

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 页而进行携带数据的接收处理才可显示正确的游戏画而。

游戏逻辑实现

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

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:

```
onLoad: function (options) {

    let level = options.level

    this.setData({

      level:parseInt(level) + 1

    })

    this.ctx = wx.createCanvasContext('myCanvas')

    this.initMap(parseInt(level))

    this.drawCanvas()

  },
```

修改 WXML(pages/game/game.wxml)代码片段如下

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

```
<view class='container'>

<!--关卡提示--><view class='title'>第{{level}}关</view>

<!--游戏画布-->

<canvas canvas-id='myCanvas'></canvas >
```

先做一些准备工作 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：

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

//地图图层数据
```

```
var map =[

[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]]


//箱子图层数据


var box =[

[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]]

var w = 40

var row = 0

var col = 0
```

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。 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:

```
initMap: function(level) {

        // 读取原始的游戏地图数据

        let mapData = data.maps[level]

        // 使用双重 for 循环记录地图数据

        for (var i = 0; i < 8; i++) {

          for (var j = 0; j < 8; j++) {

            box[i][j] = 0

            map[i][j] = mapData[i][j]



            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

            }

          }

        }

      },
```

`mapData` 取自 `data.maps[level]`，即第几关的原始地图数据。

里面是一个 8×8 的二维数组（你在 `data.js` 定义的 `map1, map2, ...`）。

`map` 与 `box` 的分工

`map[i][j]` 保存地图地形（比如墙、地板）。

`box[i][j]` 专门保存箱子或目标点信息。

这样分开存储，可以避免数据混乱。

特殊格子的处理

`4 →` 箱子目标点

在 `box` 里标记目标点。

在 `map` 里改为地板（2），表示这个位置可走。

`5 →` 玩家角色起点

在 `map` 里改为地板（2）。

记录玩家的初始位置（`row`, `col`）。

初始化一关地图，把原始数据分离成地图层和对象层，同时记录角色的初始坐标。这样后续逻辑就能：

在 `map` 判断地形（墙/地板）。

在 `box` 判断目标点、箱子。

用 `row, col` 知道玩家的位置。

上述代码首先从公共雨数文件 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 中添加自定义雨数 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:

```
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()
```

```
    },
```

这个函数 `drawCanvas` 就是 把 `map` 和 `box` 这两个二维数组里的逻辑对象，转换成画布上的图片，并控制绘制顺序，确保最终显示正常。

图层渲染顺序

清空画布：避免上一帧残影。

绘制底图：

`0 → ice`（地板/道路）

`1 → stone`（石头/墙）

`3 → pig`（小猪/目标）

叠加箱子：如果该格子 `box[i][j] == 4`，绘制箱子在上面。

叠加玩家：最后绘制玩家（小鸟）在 `row, col` 的位置。

调用 `ctx.draw()`：一次性把所有绘制操作提交给画布。

效果

每次调用 `drawCanvas`，整个 8×8 地图会 重绘一帧。

角色和箱子始终叠加在地图之上。

保证了图层结构：背景 → 箱子 → 玩家。

这样我们就初始化好页面了：

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

```
<!--方向键-->

<view class='btnBox'>

<button type= 'warn'bindtap = 'up'>↑</button >

<view >

<button type= 'warn'bindtap = 'left'>←</button >

<button type= 'warn'bindtap = 'down'>↓</button >
```

```
<button type='warn'bindtap = 'right'>→</button >

</view>

</view>

<!--“重新开始”按钮--><button type='warn'bindtap = 'restartGame'>重新开始</button></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:

```
/**

 * 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 - 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()
    this.checkWin()
},
/**
 * 自定义函数 -- 方向键：下
 */
down: function() {
    //不在最底端才考虑下移
    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) {
            //箱子不在最底端才能考虑推动
            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()
    this.checkWin()
},
/**
 * 自定义函数 -- 方向键：左
 */
left: function() {
    //不在最左侧才考虑左移
    if (col > 0) {
        //如果左方不是墙或箱子，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) {
                //如果箱子左方不是墙或箱子
                if (map[row][col - 2] != 1 && box[row][col - 2] != 4) {
                    box[row][col - 2] = 4
                    box[row][col - 1] = 0
                    //更新当前小鸟的坐标
                    col = col - 1
                }
            }
        }
    }
    //重新绘制地图
    this.drawCanvas()
    this.checkWin()
},
right: function() {
  //不在最左侧才考虑左移
  if (col < 7) {
      //如果左方不是墙或箱子，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 < 7) {
                //如果箱子左方不是墙或箱子
                if (map[row][col + 2] != 1 && box[row][col + 2] != 4) {
                    box[row][col + 2] = 4
                    box[row][col + 1] = 0
                    //更新当前小鸟的坐标
                    col = col + 1
                }
            }
        }
    }
    //重新绘制地图
    this.drawCanvas()
    this.checkWin()
},


  isWin: function() {
      //使用双重 for 循环遍历整个数组
      for (var i = 0; i < 8; i++) {
          for (var j = 0; j < 8; j++) {
              //如果有箱子没在终点
              if (box[i][j] == 4 && map[i][j] != 3) {
                  //返回 false, 表示游戏尚未成功
                  return false
              }
```

```
        }
    }
    //返回 true, 表示游戏成功
    return true
},
```

关键规则

边界检查

`if (row > 0)`：保证小鸟不在最顶行，否则不能继续上移。

能否移动

上方是空地（`map != 1` 且 `box != 4`），则直接移动。

推动箱子

上方是箱子（`box[row-1][col] == 4`）时：

检查箱子再往上一格（`row-2`）是否有障碍。

如果是空地，则箱子被推上去，同时玩家占据箱子原来的格子。

渲染与胜利判定

调用 `drawCanvas` 重新绘制整个地图。

调用 `checkWin` 检查胜利条件（通常是所有箱子在目标点上）。

直观理解

玩家移动 = 直接走上去。

玩家推箱子 = 箱子前面必须有空地，否则不能推。

每次操作都会刷新画布，并检测是否过关。

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:

```
isWin: function() {
    //使用双重 for 循环遍历整个数组
```

```
      for (var i = 0; i < 8; i++) {
        for (var j = 0; j < 8; j++) {
          //如果有箱子没在终点
          if (box[i][j] == 4 && map[i][j] != 3) {
            //返回 false, 表示游戏尚未成功
            return false
          }
        }
      }
      //返回 true, 表示游戏成功
      return true
  },
```

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:

```
checkWin: function() {
  if (this.isWin()) {
    wx.showModal({
      title: '恭喜',
      content: '游戏成功!',
      showCancel: false
    })
  }
},
```

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:

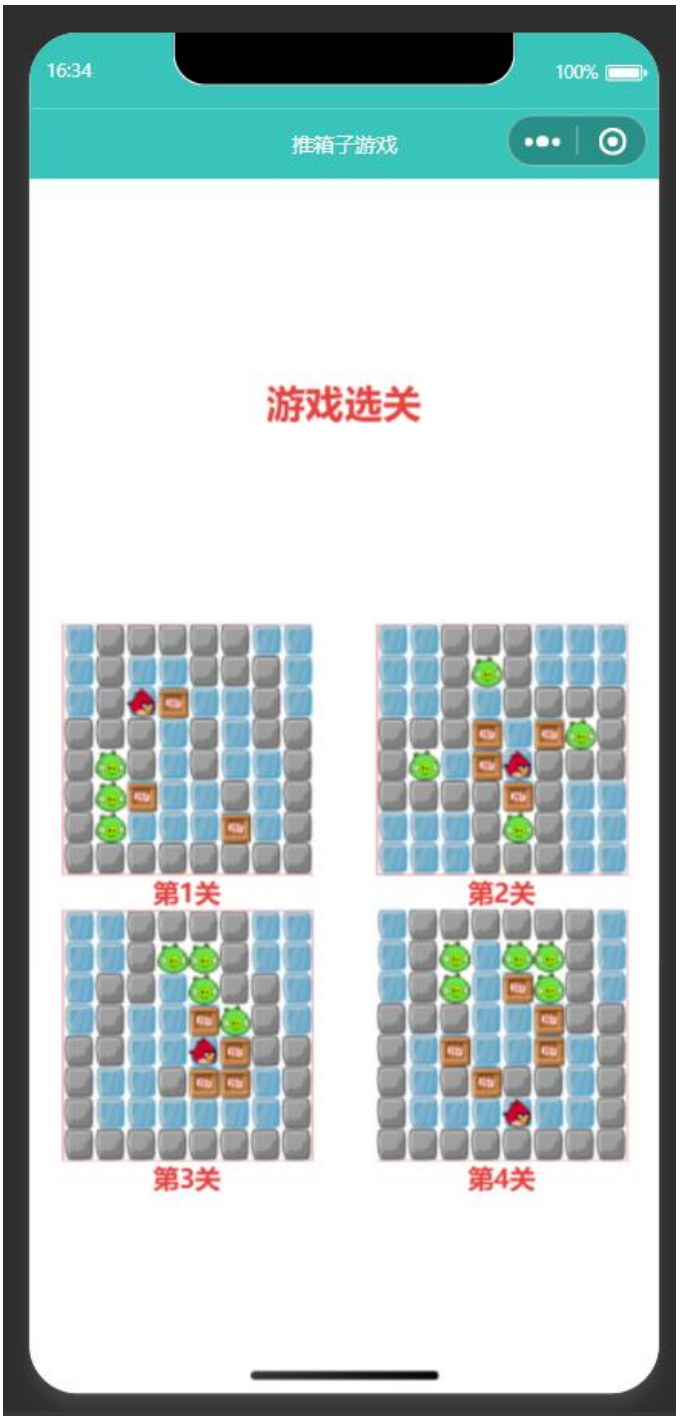
```
<!--“重新开始”按钮--><button type='warn'bindtap = 'restartGame'>重新开始</button></view>
```

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:

```
restartGame: function() {

  // 初始化地图数据

  this.initMap(this.data.level - 1)

  // 绘制画布内容

  this.drawCanvas()

}
```

三、程序运行结果

一开始打开小程序是长这样的：

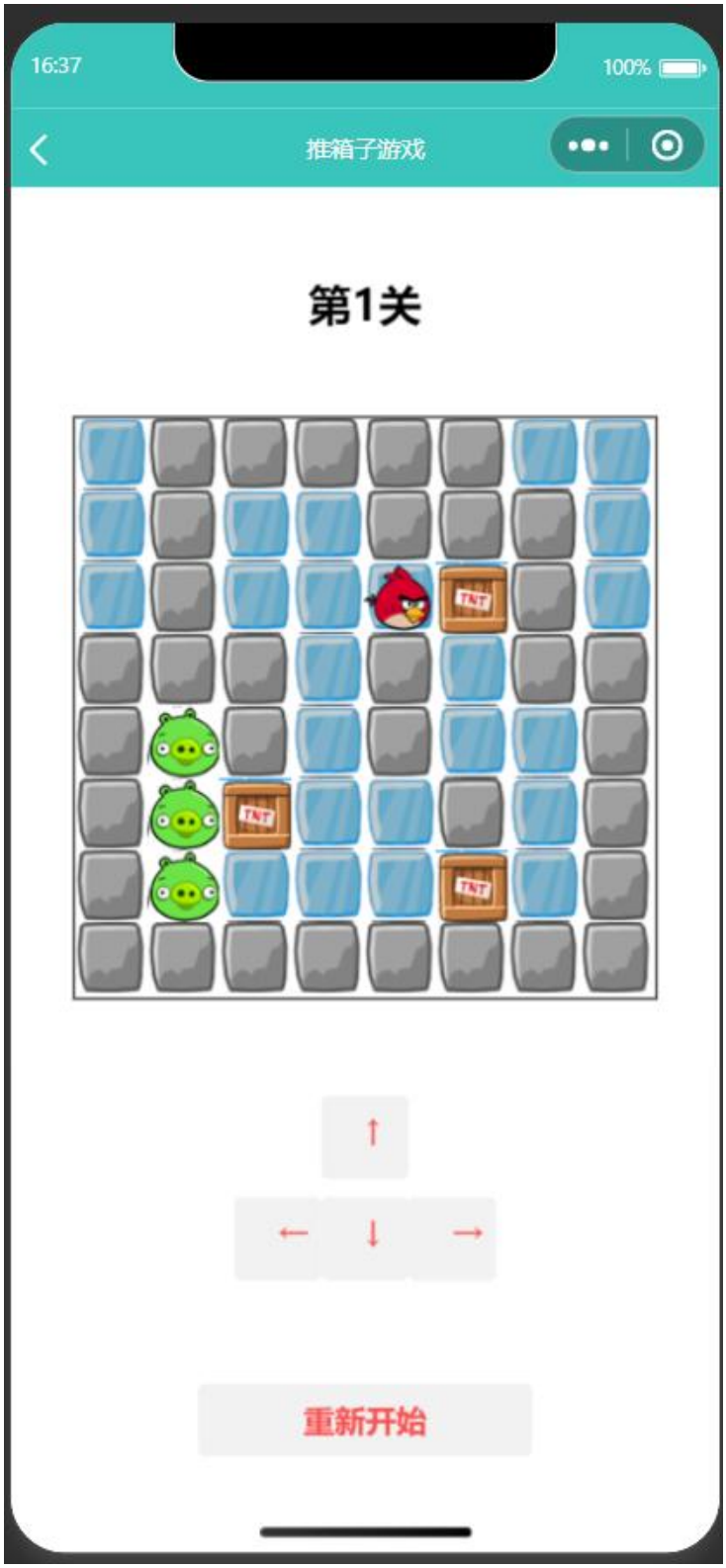


我们随便选择一关点进去，可以看到场景页面：

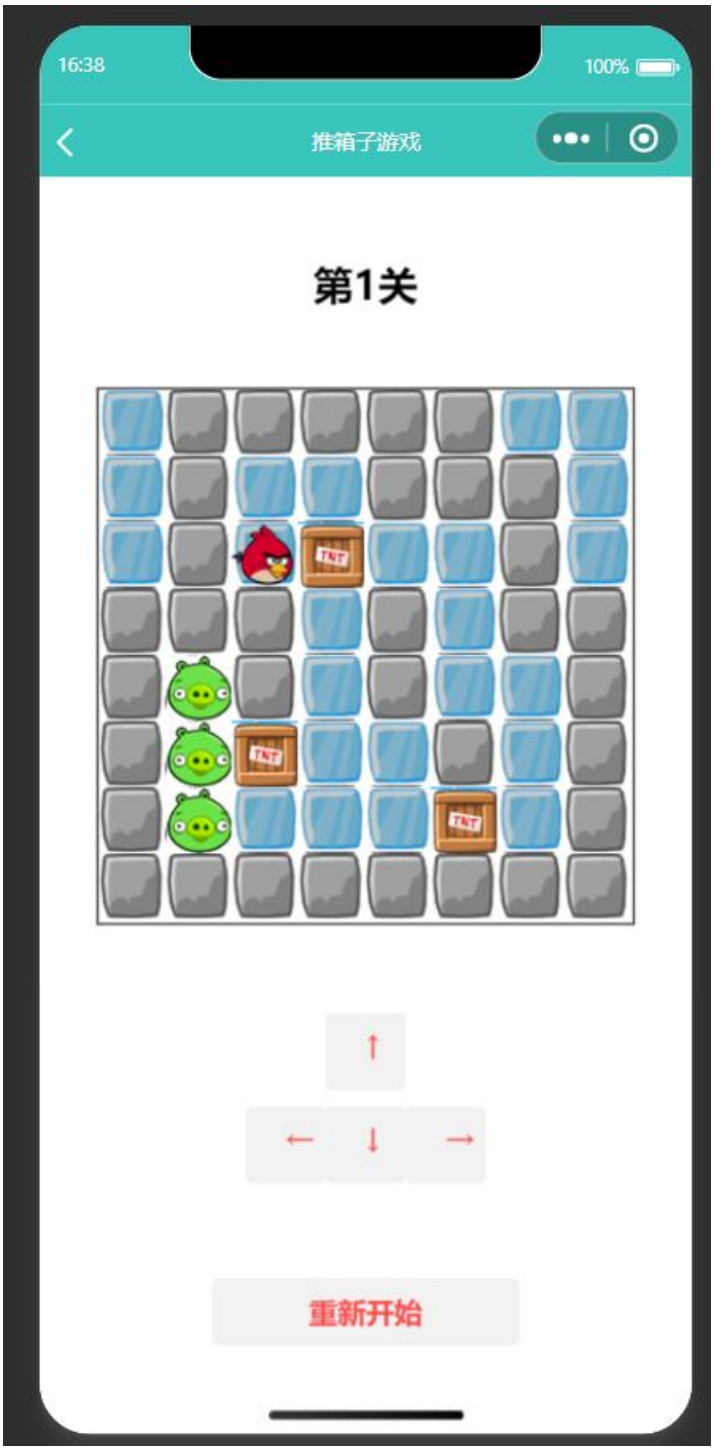
点击上键，小红鸟会向上走一格：

按右键，小鸟的右边有箱子，其可以把它推动一格

推失败了可以按重新开始来重新开始：



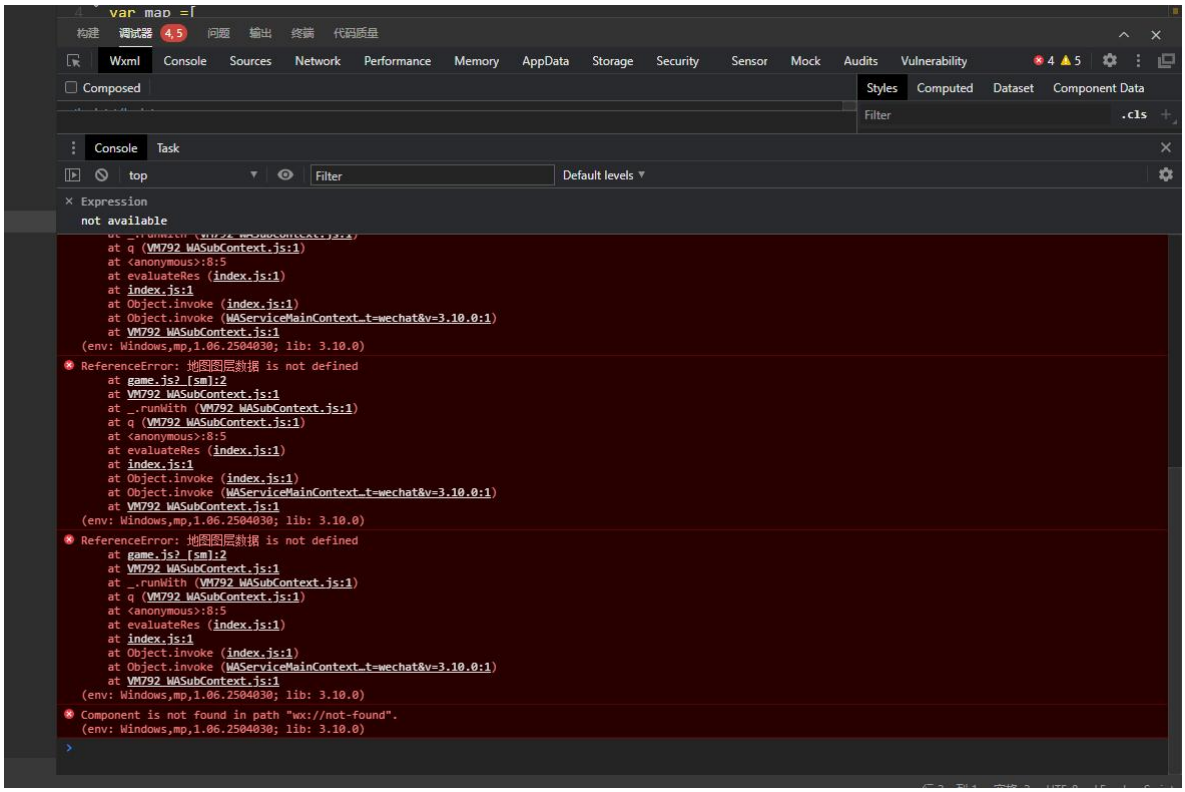
按下后：



如果推箱子成功了的话，就恭喜你：

四、问题总结与体会

在进入地图的时候会爆这样的错误：


var map =[
构建
调试器 4,5
问题
输出
终端
代码质量
Wxml
Console
Sources
Network
Performance
Memory
AppData
Storage
Security
Sensor
Mock
Audits
Vulnerability
4 5
Composed
Styles
Computed
Dataset
Component Data
Filter
.cls
Console
Task
top
Filter
Default levels
Expression
not available
at q (VM792 WASubContext.js:1)
at <anonymous>:8:5
at evaluateRes (index.js:1)
at index.js:1
at Object.invoke (index.js:1)
at Object.invoke (WAServiceMainContext…t=wechat&v=3.10.0:1)
at VM792 WASubContext.js:1
(env: Windows,mp,1.06.2504030; lib: 3.10.0)
ReferenceError: 地图图层数据 is not defined
at game.js? [sm]:2
at VM792 WASubContext.js:1
at _.runWith (VM792 WASubContext.js:1)
at q (VM792 WASubContext.js:1)
at <anonymous>:8:5
at evaluateRes (index.js:1)
at index.js:1
at Object.invoke (index.js:1)
at Object.invoke (WAServiceMainContext…t=wechat&v=3.10.0:1)
at VM792 WASubContext.js:1
(env: Windows,mp,1.06.2504030; lib: 3.10.0)
ReferenceError: 地图图层数据 is not defined
at game.js? [sm]:2
at VM792 WASubContext.js:1
at _.runWith (VM792 WASubContext.js:1)
at q (VM792 WASubContext.js:1)
at <anonymous>:8:5
at evaluateRes (index.js:1)
at index.js:1
at Object.invoke (index.js:1)
at Object.invoke (WAServiceMainContext…t=wechat&v=3.10.0:1)
at VM792 WASubContext.js:1
(env: Windows,mp,1.06.2504030; lib: 3.10.0)
Component is not found in path "wx://not-found".
(env: Windows,mp,1.06.2504030; lib: 3.10.0)

问了 gpt，屁用没有：

这个报错信息核心在这里：

```javascript
Error: module 'utils/data.js' is not defined, require args is '../../utils/data.js'
```

意思是：你在 `game.js` 里写了

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

但小程序运行时没有找到这个模块。

常见原因有：

1. 路径写错

- 如果 `game.js` 在 `pages/game/` 目录下：

```kotlin
project
├── utils/
|   └── data.js
└── pages/
    └── game/
        └── game.js
```

那么从 `game.js` 到 `utils/data.js` 的相对路径是：

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

⚠️ 注意 `../` 的数量要对应层级：

- `pages/game/game.js` → `../../utils/data.js` ✅
- `pages/index/index.js` → `../utils/data.js` ✅

如果你 utils 目录其实在 **根目录**下，那路径没问题；如果它在 `pages/` 里，就得改成 `../utils/data.js`。

无意直接发现了错误：

居然是写注释的时候不小心少打了一个/，没注释上

```
var data = require('../../utils/data.js')
/地图图层数据
```