2023年夏季《移动软件开发》实验报告

姓名：檀宗晗 学号：21020007087

姓名和学号?	**檀宗晗，21020007087**
本实验属于哪门课程?	中国海洋大学23夏《移动软件开发》
实验名称?	实验6：推箱子游戏
博客地址?	https://www.cnblogs.com/-tcxm
Github仓库地址?	https://github.com/tzhcyd/class.git （dev分支）

（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canvas和绘图API。

二、实验步骤

1、页面配置

根据要求创建页面文件并删除和修改文件

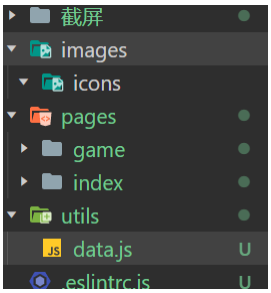
```
App({

  /**
   * 当小程序初始化完成时，会触发 onLaunch（全局只触发一次
   */
  onLaunch: function () {

  },

  /**
   * 当小程序启动，或从后台进入前台显示，会触发 onShow
   */
  onShow: function (options) {
```

并创建其他自定义文件，包括images文件夹，用于存放图片素材；utils文件夹，用于存放公共js文件，并将四副关卡图片以及五个游戏图标素材放入images文件夹中；在utils文件夹中创建公共文件data.js,结构如下图：



2、导航栏设计

app.json代码如下：

```
"window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#E64340",
    "navigationBarTitleText": "推箱子游戏",
    "navigationBarTextStyle":"white"
  },
```

上述代码可将导航栏背景色置为珊瑚红色、字体为白色，效果如下：

3、页面设计

3.1公共样式设计

在app.wxss中设置页面容器和顶端标题的公共样式，代码如下：

```
.container{
    height: 100vh;
    color: #e64340;
    font-size: bold;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: space-evenly;
}

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

3.2首页设计

首页包括两部分内容，即标题和关卡列表，wxml代码如下：

```
<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>
```

wxss代码如下：

```
.levelBox{
    width: 100%;
}
.box{
    width: 50%;
    float: left;
    margin: 20rpx 0;
    display: flex;
    flex-direction: column;
    align-items: center;
}
image{
    width: 300rpx;
    height: 300rpx;
}
```

得到效果如图：（当前可显示标题和一个临时关卡）

3.3游戏页面设计

游戏页需用户点击首页关卡列表，然后在新窗口中打开该页面，包括游戏关卡列表、游戏画面、方向键和“重新开始”按钮，wxml代码如下：

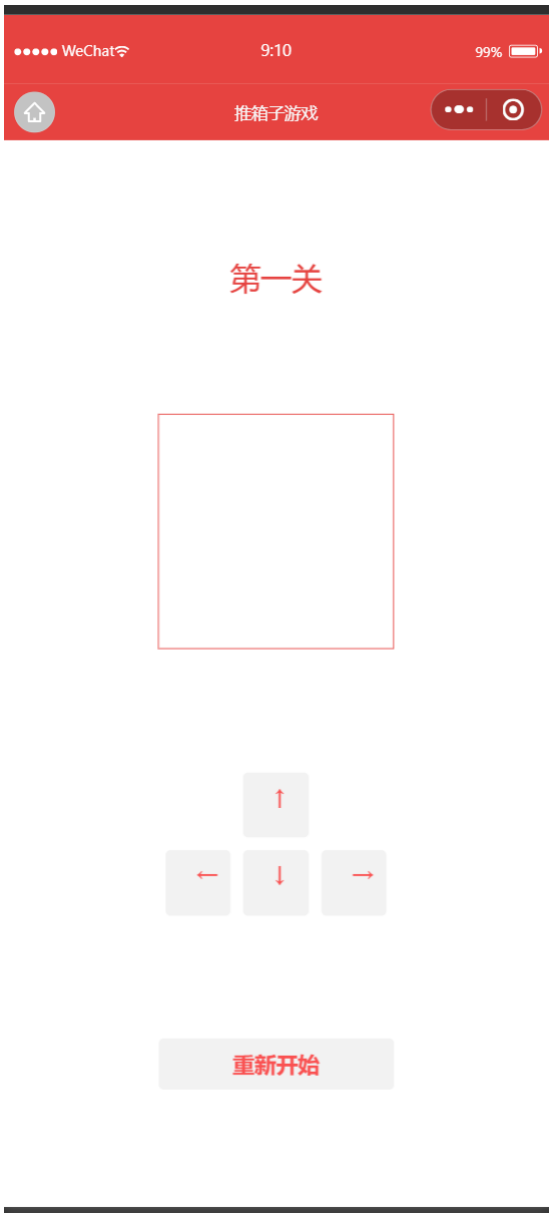
```
<view class = "container">
    <view class = 'title'>第一关</view>
    <canvas canvas-id = 'myCanvas'></canvas>
    <view class="btnBox">
        <button type="warn" style="width: 90rpx; height: 90rpx; margin:
10rpx;">↑</button>
        <view>
            <button type="warn" style="width: 90rpx; height: 90rpx; margin:
10rpx;">←</button>
            <button type="warn" style="width: 90rpx; height: 90rpx; margin:
10rpx;">↓</button>
            <button type="warn" style="width: 90rpx; height: 90rpx; margin:
10rpx;">→</button>
        </view>
    </view>
    <button type="warn">重新开始</button>
</view>
```

可以在wxml文件中使用style属性给button组件设置样式

wxss代码如下：

```
canvas{
    border: 1rpx solid;
    width: 320rpx;
    height: 320rpx;
}
.btnBox{
    display: flex;
    flex-direction: column;
    align-items: center;
}
.btnBox view{
    display: flex;
    flex-direction: row;
}
```

得到的效果如图：



4、逻辑实现

4.1公共逻辑

在公共js文件中配置游戏地图数据，1为墙、2为路、3为终点、4为箱子、5为人物、0为墙的外围，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],
]
```

最后使用module.exports语句暴露数据出口，代码如下：

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

在game页面的js文件顶端引用公共js文件，代码如下：

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

4.2首页逻辑

4.2.1关卡列表展示

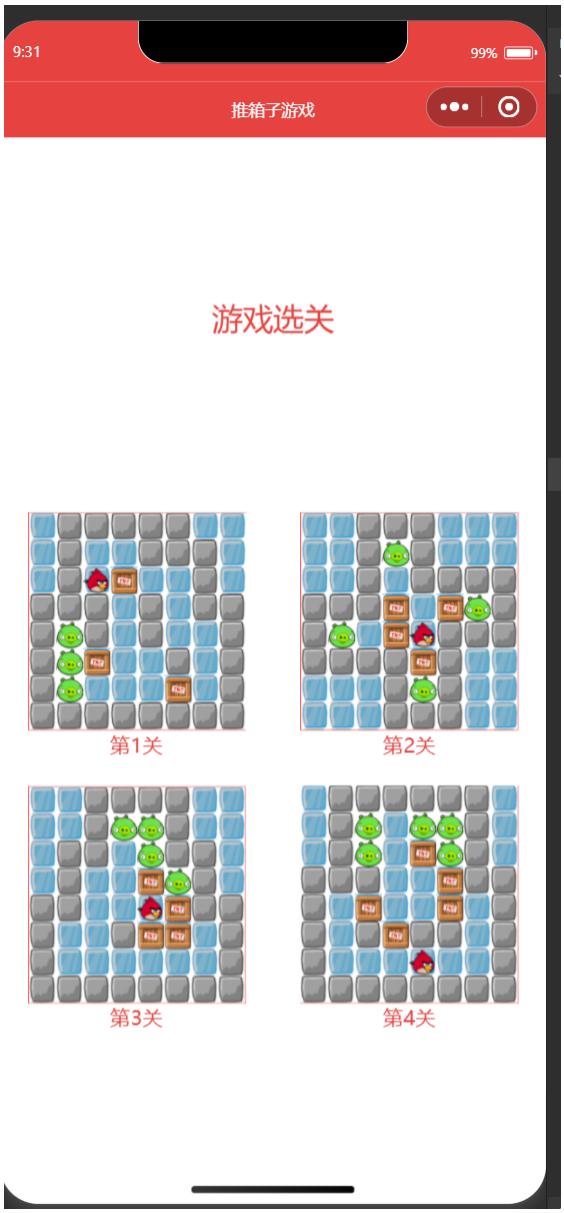
在index.js文件的data中录入关卡图片的数据信息，代码如下：

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

wxml添加wx:for属性循环显示关卡列表数据和图片，代码如下：

```
<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>
```

效果如下图：



4.2.2点击跳转游戏页面

实现用户点击即可实现跳转，wxml代码如下：

```
<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>
```

添加了自定义点击事件chooseLevel，并使用data-level属性携带了关卡图片的下标信息

函数内容如下：

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

当前可实现点击跳转，但仍需在game页面进行携带参数的接受处理才能正确显示游戏画面

4.3游戏页逻辑

4.3.1显示当前是第几关

因为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，现将其显示出来

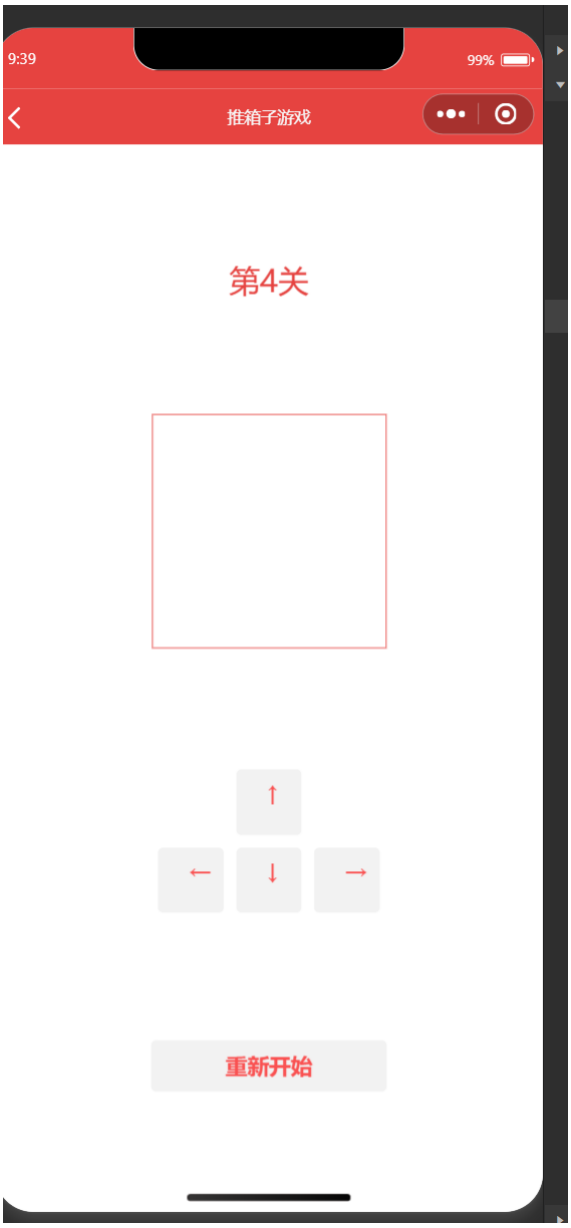
js代码如下：

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

wxml代码修改如下：

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

将其设为动态数据并获取赋值即可实现该效果，效果如下：



4.3.2游戏逻辑实现

在game.js文件顶端记录游戏初始数据信息，代码如下：

```
//地图数据
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]
]
//方块高度
var w = 40
//初始化小鸟行和列
var row = 0
var col = 0
```

初始化游戏页面：

在game.js文件中添加initMap函数，用于初始化游戏地图，mapData即为从公共js文件中读取的地图数据，更新当前游戏的初始地图数据、箱子数据以及游戏主角所在位置，4为箱子，5为人物，并在此记录下小鸟的位置，对应代码如下：

```
//初始化数据地图
    initMap : function(level){
        let mapData = data.maps[level]
        //读取原始的游戏地图数据
        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
                    //记录小鸟当前行和列
                    row = i
                    col = j
                }
            }
        }
    },
```

添加自定义函数drawCanvas，用于将地图信息绘制到画布上，首先需要清空画布，以防后期点击重新开始画布还存有上一局的游戏内容，随后使用双重for循环绘制8*8的地图，并在地图的基础上叠加绘制箱子和小鸟，代码如下：

```
//绘制地图
    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8*8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,console.warn())
        ctx.draw()
    },
```

最后在onLoad函数中创建画布上下文，并依次调用自定义函数initMap和drawCanvas，使内容成功显示，代码如下：

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

效果如图：

方向键逻辑实现：

修改wxml页面中的四个方向键，并为其绑定点击事件，代码如下：

```
<button type="warn" bindtap="up" style="width: 90rpx; height: 90rpx; margin:
10rpx;">↑</button>
        <view>
            <button type="warn" bindtap="left" style="width: 90rpx; height: 90rpx;
margin: 10rpx;">←</button>
            <button type="warn" bindtap="down" style="width: 90rpx; height: 90rpx;
margin: 10rpx;">↓</button>
            <button type="warn" bindtap="right" style="width: 90rpx; height: 90rpx;
margin: 10rpx;">→</button>
        </view>
```

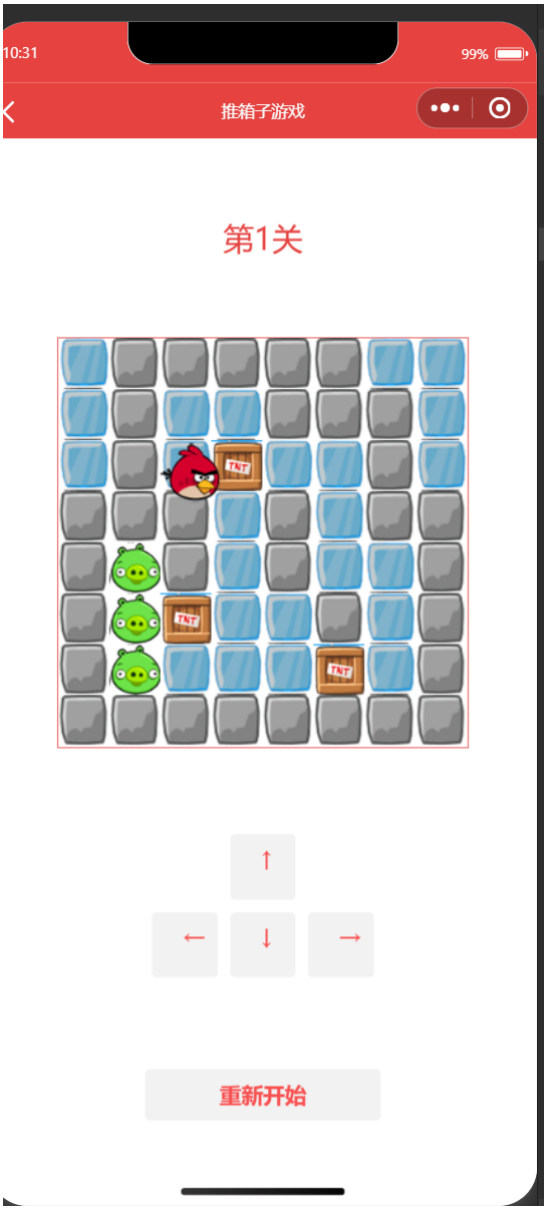
添加自定义函数up、down、left、right，分别用于实现小鸟的上下左右移动，每次点击在条件允许的情况下移动一格，具体判断条件见代码中的注释，对应代码如下：

```
  //方向键：上
```

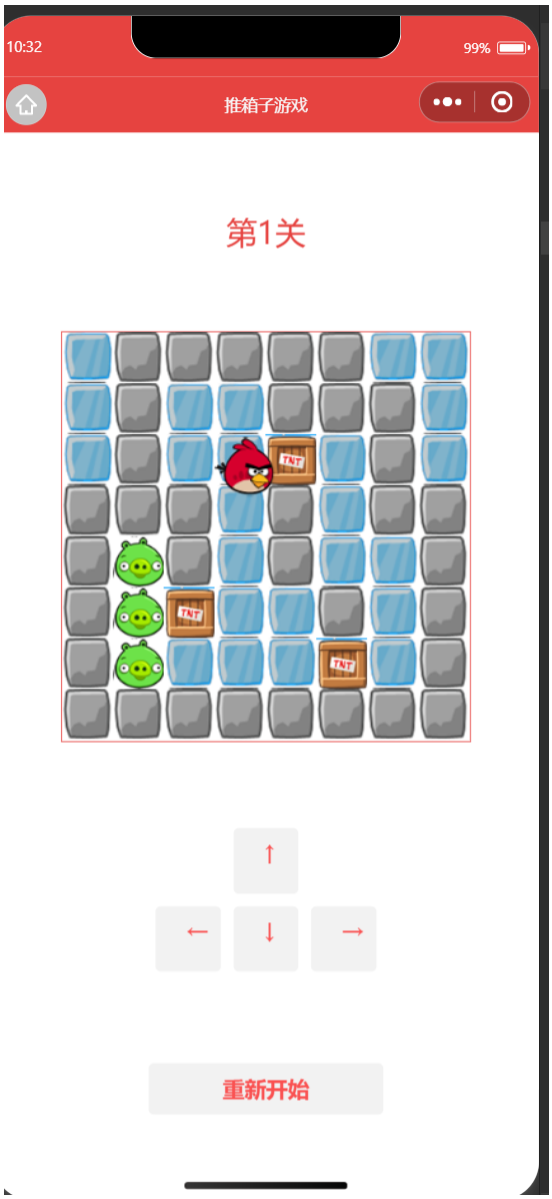
```
    up : function(){
        //不在最顶端才考虑上移
        if(row>0){
            //如果上方不是墙或箱子，可以移动小鸟
            if(map[row-1][col] != 1 && box[row-1][col] !=4){
                //更新当前小鸟坐标
                row = row -1;
            }
            //如果上方是箱子
            else if(box[row - 1][col] == 4){
                //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
                        //更新当前小鸟坐标
                        row = row - 1
                    }
                }
            }
            //重新绘制地图
            this.drawCanvas()
        }
    },
    //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
                        //更新当前小鸟坐标
                        row = row + 1
                    }
                }
            }
            //重新绘制地图
            this.drawCanvas()
        }
    },
    //方向键：左
```

```
    left : function(){
        //不在最左侧才考虑上移
        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 - 2] = 4
                        box[row][col - 1] = 0
                        //更新当前小鸟坐标
                        col = col - 1
                    }
                }
            }
            //重新绘制地图
            this.drawCanvas()
        }
    },
    //方向键：右
    right : function(){
        //不在最右侧才考虑上移
        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 + 2] = 4
                        box[row][col + 1] = 0
                        //更新当前小鸟坐标
                        col = col + 1
                    }
                }
            }
            //重新绘制地图
            this.drawCanvas()
        }
    },
```

初始状态如图：



点击向右移动一格：



4.3.3判断游戏成功

在js文件中添加自定义函数isWin，用于判断游戏是否已经成功，对应代码如下：

```
//判断游戏是否成功
    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，表示游戏游戏尚未成功
                    return false
                }
            }
        }
        //返回true
        return true
    },
```

上述代码的判断逻辑是只要有一个箱子没在终点位置就判断游戏尚未成功，然后在添加自定义函数checkWin，一旦游戏成功就弹出提示对话框，代码如下：

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

最后在四个方向键函数中追加关于游戏成功判断的函数，在最后加上如下代码：

```
//检查游戏是否成功
this.checkWin()
```

游戏成功效果如下：

4.3.4重新开始游戏

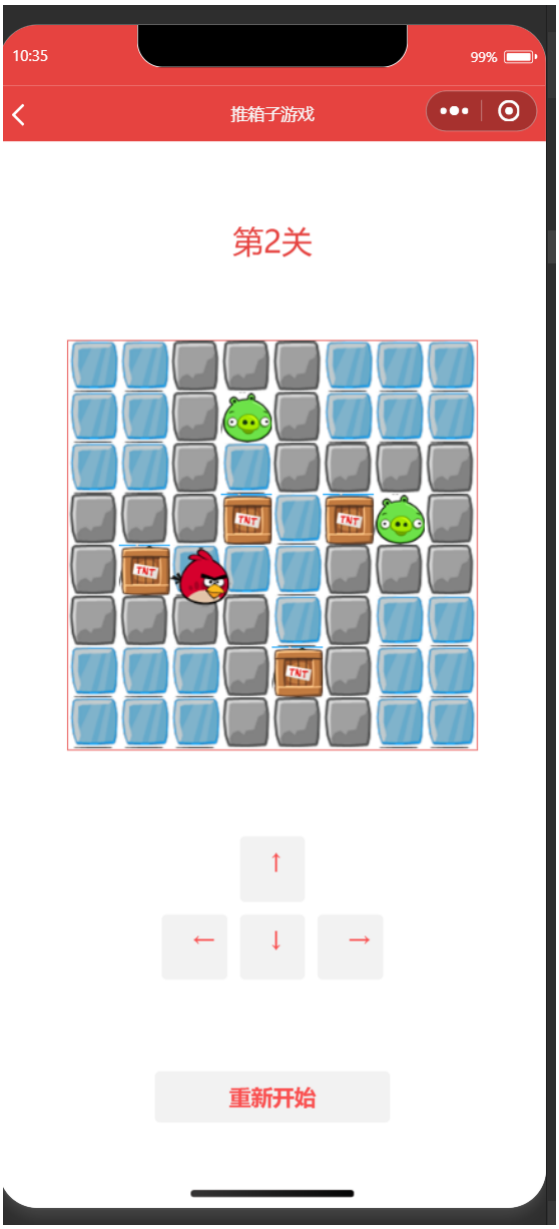
在wxml中为重新开始按钮添加restartGame函数，代码如下：

```
<button type="warn" style="margin: 10rpx;" bindtap="restartGame">重新开始</button>
```

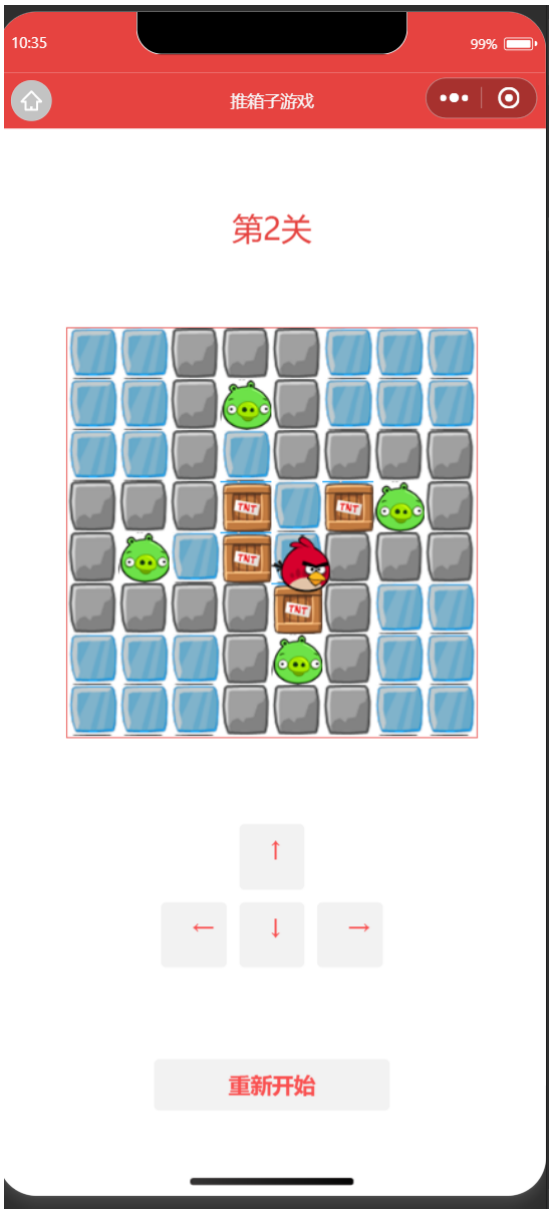
函数代码如下：

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

点击重新开始前效果：

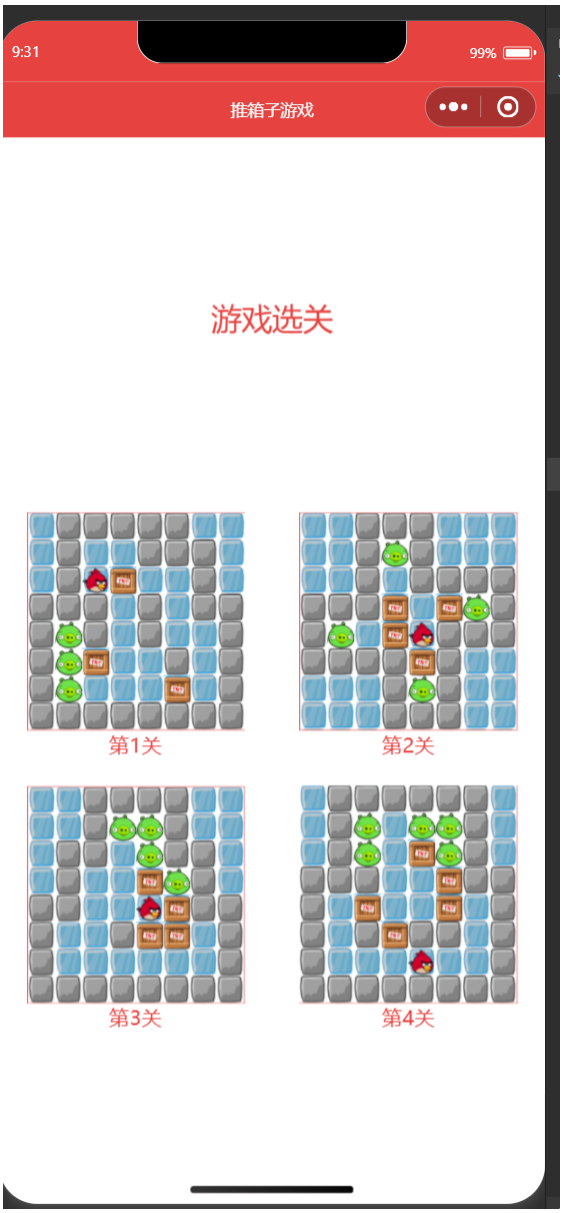


点击后：



三、程序运行结果

游戏主页：



游戏关卡：

游戏成功：

四、问题总结与体会

本次实验让我对于canvas组件的用法有了更深刻的了解，并且已经可以较为熟练的将其应用到小程序当中，但是最后完成实验时发现我的按键颜色与预期有所不同，结合同学在群里所说，小程序一直以来采用的都是 AppService 和 WebView 的双线程模型，基于 WebView 和原生控件混合渲染的方式，小程序优化扩展了 Web 的基础能力，保证了在移动端上有良好的性能和用户体验。官方在 WebView 渲染之外新增了一个渲染引擎 Skyline，但是这个新的引擎还不是特别完善，很多样式不能够兼容，会有很多问题，因为我是直接在button的按钮组件中定义的style样式，所以可能会与预期效果不同。

通过本次实验我也掌握了如何去完成一个较为完成的小游戏，并掌握了其中的逻辑关系，页面跳转传参等内容以及画布方面的知识，受益匪浅。