1. RPGMV的前沿技术

迈向未来

1. RPGMV 1.7.1

看到好像有些朋友还不知道，我来发一个pixi5的升级补丁吧，直接覆盖即可，此补丁将mv的pixi版本从4.5.4提升到5.3.8，pixi从v5版本用上了webGL2.0，我个人觉得是MZ流畅提升的关键，群里已经有一个“RPGMV 1.7.1”的民间补丁，那个补丁使用的pixi版本是5.3.3，也是可以的，但是存在一点BUG，所以还是更推荐用这个  
我不是作者，这是原作者的链接  
https://github.com/kins-dev/rpgmaker-mv-pixi5

之后强烈建议再升级一下mwjs版本，mv默认的版本已经是远古时代的了，chrome版本太老旧了，这个可以直接去nw.js官网下载，然后替换掉mv目录下的nwjs-win和nwjs-win-test里面的旧版本（注意备份），然后只需要把nw.exe改名为game.exe即可，如果嫌弃nwjs不够稳定，还可以研究一下用electron发布你的游戏，也不是很复杂

1. 基于JavaScript的RPGMV项目插件编写
2. 自定义一个窗口

如何直接做一个窗口类[[[1]](#footnote-0)]？调用重写了很多构造函数。这是无插件指令的写法。

Scene\_Splash.prototype.create 方法、Scene\_Splash.prototype.start 方法、Scene\_Splash.prototype.stop 方法、Scene\_Splash.prototype.terminate 方法、Scene\_Splash.prototype.update 方法等（当然除了构建器部分外，其它方法都是可选的）。下面就在Scene\_Splash类中重写这些方法。

图像：想把图像放到正中间，可以考虑this.centerSprite()方法[[[2]](#footnote-1)]。

1. 在Sprite类及其子类添加图片

一般我们在Sprite类的create方法内写导入图片，我们要先用this的方式来添加一个新的变量作为导入的图片，然后用导入图片的方法导入图片即可。随后Sprite类就会出现这个图片。为了美观，可以在create方法内部顺便定义好图片的坐标值。

为什么要用this的方式添加变量而不是用var的方式呢？作者做了一个实验，发用this的方式定义的新变量才可以正常使用。当前无法解释其原理。

我们在导入图片的时候，还要使用到addChild方法，将图片正式添加到Sprite类中，这样才可以把局部变量变成全局变量。

1. TilingSprite满版精灵类的使用——实现图像的移动

《针对SIAKO.Mobi 14节课——满版图像》

TilingSprite类是pixi源码里的类，称为“满版精灵类”。为了使得导入的图像可以持续的移动，我们要先给图像确定中心，然后才能移动。

用TilingSprite.prototype.move方法来确定图像的位置和大小，其中，大小的参数我们通常会用Graphics.width和Graphics.height的方式。其本质是Graphics静态类调用图像的width和height属性。

我们在TilingSprite对象的所在场景的update方法[[[3]](#footnote-2)]内，通过调整此对象的origin成员变量附带的x，y坐标值，来实现TilingSprite对象在场景内部的坐标移动。我们之所以使用origin成员变量是因为origin在TilingSprite中是专门使用与图像移动的一个坐标控制点。

在网课中，专门提到了move方法和origin成员变量的相互使用，这两个用法必须同时使用。暂时无法解释其必要性。

1. 在已有的菜单栏中创建新的窗口——写一个自己的窗口类

《SIAKO.Mobi15节课——在菜单栏窗口中创建一个新的窗口[[[4]](#footnote-3)]》

主要的算法是，在已知的“窗口场景”的内部里面，再添加一个新的窗口。

在Scene\_Menu.prototype.createCommandWindow方法内部，添加新的窗口。输出数据的方式是使用Window\_Base窗口基类的drawTextEx方法来输出一行字体。可以输出字符串。数据的输出方式也是当做是字符串输出。如果我们要把数值当做字符串输出时，我们可以用String()方法来进行类似于其他语言的“强制类型转换”。输出字体一般写在场景类的refresh方法内部。

网课里面用的是重写Window\_Gold类，最重要的是重写一个窗口类的以下方法：定义、initialize、windowWidth、windowHeight、refresh[[[5]](#footnote-4)]。我们写的窗口类是以Window\_Base作为父类的，而不是选择Window类。现在无法解释。

1. 创建一个新的场景类

《SIAKO.Mobi16节课——新建场景，嵌入窗口》

主要算法：先新建一个场景类，然后再把窗口显示在场景内部。SIAKO.Mobi指出，不可以用addChild方法代替addWindow方法。但是在我们自己的实验中，是可以实现互换的。

我们新建的一个场景类，是用Scene\_MenuBase作为父类的，试了一下用Scene\_Base作为父类，发现背后的场景是全黑的。并不是原来的半透明效果。

试了一下用Stage类作为父类，发现直接报错。说明Stage类并没有具体实现一些方法。

新建的场景类主要重写Scene\_Menu菜单场景、继承于Scene\_MenuBase菜单基类场景。因为我们需要使用到菜单场景自带实现的半透明背景图，所以一般会选择Scene\_MenuBase作为基类。主要重写的方法有：定义、initialize、create、start。如果需要场景刷新，可以看情况选择重写update和refresh方法。

1. 建立自己的游戏启动场景

《SIAKO.Mobi17节课——建立自己的游戏启动场景》

首先要建立一个新的场景，主要重写Scene\_Gameover场景类。重写的方法有：initialize、create、start、update、stop、terminate。

在create方法中、这个场景需要开始执行播放背景音乐，创建启动界面图。故需要在自建的Scene\_Splash场景类添加playSplashBGM和createSplashSecneImage方法。

在playSplashBGM方法中，要先关闭背景音乐。故需要AudioManager.stopBgs()和AudioManager.stopMe()方法。随后开始声明音乐类型。实际上完成播放的核心方法仅有 AudioManager.playBgm方法。AudioManager.stopAll()方法停止全部的音频工作。

createSplashSecneImage方法给Scene\_Splash类添加一个新的图片精灵对象。1：新建一个Sprite类精灵类图片对象。2：用ImageManager.loadSystem方法导入图片。3：用addChild将图片添加进入Scene\_Splash类。

在update方法中，需要监听用户的输入流。因此需要重写isTriggered方法。使之可以监控键盘和屏幕的输入流。update方法需要转移场景到Scene\_Title标题场景类内部。

其余的大部分代码都不需要重写。只需要保留在类中即可。他们会自动执行。

1. 更改一个窗口的背景图片，并设置其位置、大小、透明度等参数

这里会有一个误区：“我们能修改菜单窗口背景图，其他的窗口都不能修改其背景。”这个误区一直“指导”我们来修改Scene类的背景图。我不记得这个误区是不是从SIAKO.Mobi的视频里面学来的，反正我印象深刻。凡是一个Scene都可以通过addChild方法来添加一张图片。注意，我们只是添加了一张图片。Scene可以使用addChild方法，是毋庸置疑的。核心证据是最深层的Stage类实现了PIXI的Container容器类。所以Container类的子类均可以使用addChild方法[[[6]](#footnote-5)]。用这种追根溯源的方式，我们甚至可以看到任何的窗口类都可以使用addChild的证据[[[7]](#footnote-6)]以及精灵类可以使用addChild的证据[[[8]](#footnote-7)]。

我们直接修改的对象是来自Window基类的\_windowBackSprite变量。\_windowBackSprite表示的就是一个窗口的背景，这个背景就是以一个Sprite精灵的方式来描述的。我们只需要给这个精灵对象赋值即可。

但是这里的赋值又会出现一个问题，我们不能new一个新的Sprite来直接代替好这个\_windowBackSprite。而是要在\_windowBackSprite.bitmap里面，添加位图对象。**给精灵的位图添加对象，添加位图，而不是精灵。**目前我无法解释，为什么要这样写，反正程序就是达不到效果。具体的写法可以如下：

var backImage\_Bitmap = ImageManager.loadSystem('2020\_temp');

this.\_windowBackSprite.bitmap = backImage\_Bitmap;

现在一般默认用ImageManager. 图片管理者的方式来导入位图。这个方法是最稳妥的。在上面的教程中已经提及到过。

一般我们是先采取导入图片，再对Sprite对象进行调整。mv的Sprite类继承并重写了来自PIXI的Sprite类。从源码易知，可以更改tint、alpha、scale等参数。scale参数是修改一个Sprite对象大小的核心要素[[[9]](#footnote-8)]。不能直接用width和height的方式来直接修改一个Sprite对象。这个没有效果。

我更愿意用alpha来代替使用opacity。因为opacity的本质是修改一个Sprite的透明度，其实就是封装了alpha变量[[[10]](#footnote-9)]。这个用法是直接调用PIXI的定义的。

***待检验：用setFrame方法来实现导入图片的大小修改。（本方法由QQ ： 2335937510 提供）***

1. 利用$gameActors输出一个角色所具有的技能

严格来说，人物角色和技能是被专门地放在两份.json数据库文件的。我们肯定要找到人物技能和人物之间的关系的。按照这种思路，我们肯定要用人物来输出人物附属的技能。

在DataManager类中，明确说明了$gameActors是Game\_Actors类的成员。

***$gameActors = new Game\_Actors();***

接下来在Game\_Actors类中，我们发现其本质就是包装了Game\_Actor类，其中的Game\_Actors.prototype.actor方法告诉我们了如何使用这个$gameActors来访问具体存放角色数据的方式——在其形参内填写角色的编号。值得一提的是，我们要通过具体的调试才能准确判别角色编号。有可能会出现数组下标为0时，所指向的并不是第一个角色的情况。

***Game\_Actors.prototype.actor = function(actorId) {***

***if ($dataActors[actorId]) {***

***if (!this.\_data[actorId]) {***

***this.\_data[actorId] = new Game\_Actor(actorId);***

***}***

***return this.\_data[actorId]; }***

***return null; };***

在Game\_Actor类中，我们可以使用其Game\_Actor.prototype.skills方法来输出其技能。在这里，$gameActors调用到了$dataSkills。易知，skills方法输出的是这个角色的全部技能，这个技能是以一个数组的形式输出的。

***Game\_Actor.prototype.skills = function() {***

***var list = [];***

***this.\_skills.concat(this.addedSkills()).forEach(function(id) {***

***if (!list.contains($dataSkills[id])) {***

***list.push($dataSkills[id]); }***

***});***

***return list; };***

综上所述，输出一个角色的核心代码写法为：$gameActors.actor(1).skills()[1].name 这里的name表示的是技能名而不是角色名。注意数组下标的选取。易知，这种写法类似于一个二维数组。

值得说明的是，直接调用.json数据库的方式是$gameActors[index]，直接把“$gameActors”当做是一个数组名，访问数组内容，即数据库内部的内容。而不是一个对象来访问其成员方法。

1. [] 简书.鳗驼螺.【RPG Maker MV插件编程】【实例教程2】制作一个启动画面 <https://www.jianshu.com/p/8b67041f02d5> [↑](#footnote-ref-0)
2. [] 鳗驼螺.4.标题画面 <https://www.jianshu.com/p/527a82a2fd6a> [↑](#footnote-ref-1)
3. [] update方法更新每一帧，而图像移动是更新每一帧的。故图像的更新写在场景类的update方法内。 [↑](#footnote-ref-2)
4. [] SIAKO.Mobi 15节 <https://www.bilibili.com/video/av23706785?p=15> [↑](#footnote-ref-3)
5. [] refresh方法是在更新的时候才调用。和update不一样。 [↑](#footnote-ref-4)
6. [] Scene\_Base.prototype = Object.create(Stage.prototype);说明了任何场景类均实现了Stage类。

   Stage.prototype = Object.create(PIXI.Container.prototype); 说明了Stage类实现了PIXI的容器类。 [↑](#footnote-ref-5)
7. [] Window.prototype = Object.create(PIXI.Container.prototype);说明了Window类实现了PIXI的容器类。 [↑](#footnote-ref-6)
8. [] @extends PIXI.Container 在PIXI源码内，这说明了Sprite类继承了容器类。 [↑](#footnote-ref-7)
9. [] width: {

   get: function () {

   return Math.abs(this.scale.x) \* this.texture.orig.width;

   },

   set: function (value) {

   var sign = utils.sign(this.scale.x) || 1;

   this.scale.x = sign \* value / this.texture.orig.width;

   this.\_width = value;

   }

   }, 说明了Sprite类的宽高属性本质上就是scale和其材质texture的操作。 [↑](#footnote-ref-8)
10. []

    Object.defineProperty(Sprite.prototype, 'opacity', {

    get: function() {

    return this.alpha \* 255;

    },

    set: function(value) {

    this.alpha = value.clamp(0, 255) / 255;

    },

    configurable: true

    }); 说明了opacity的本质就是封装了Sprite的alpha值。 [↑](#footnote-ref-9)