THE NVL MAKER
新手教程
（四）
历史纪录与系统设定界面

一、历史记录界面

（1）界面编辑器

首先你一定已经知道怎么打开这个界面，
以及如何给它换背景，换各种按钮，换滚动条样式。
并且还知道可以用“文字范围”设定历史记录显示在画面上的什么位置。
如果真的这些都不知道，咳咳，试一下肯定知道了。
不过其他的一些详细设置，就需要用到 Config.tjs 了。
这是吉里吉里的一个通用设置文件，当然，NVL 里也可以用图形化工具来编辑它。

（2）Config.tjs

Config.tjs 与历史记录相关的设置如下。
在界面设定下点击“Config.tjs”即可自动打开编辑工具。
假如你真的完全不知道它们是干嘛的，请慎重修改。= =b

（3）脚本里操作历史记录

只有在有对话，并且开启历史记录的情况下，这个界面才会显示内容。因此游戏开始时，请在脚本中添加这个指令。

否则就不要问我为什么打开这个界面是一片空白啦。

另外关于“允许开启”，在普通对话框和大对话框的情况下，使用滚轮就可以自动打开历史记录界面，而透明全屏对话框下则禁止了这个操作。
基本上这些设定只是作者的习惯而已，并没有什么特别意义。
如果你觉得需要修改的话，可以修改 Data/macro/macro_play.ks。
[macro name=dia][macro name=scr][macro name=menu]三个宏里的[history]指令，将 enabled 的值改为 true 或者 false。

二、系统设定界面

（1）界面编辑器

你一定在想我到底还贴这个东西出来干嘛，只要看了教程怎么可能到现在还不知道怎么填这些东西呢哈哈哈……

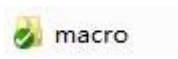
嗯没错是这样的，所以，接下来要讲的东西就是，怎么不通过界面编辑器，往游戏系统里加新的内容。

今天主要讲的内容就是怎么在系统设定界面里加入“都设为默认值”按钮。

（2）macro 文件夹

哎呀不要摆出“=口=”的表情，这些就算看不懂也没关系的。
这个还是很有需求的吧，为什么界面编辑器里没有呢？
……因为作者也忘了，后来就懒得加了（喂）。
所以干脆就利用这个机会，讲解一下“在你不知道的时候，THE NVL Maker 都在做什么”吧。


<=这个按钮你肯定戳过了对不对，没戳过的话就赶紧去戳一下。



<=然后我们再打开这个文件夹。也就是 project/你的游戏文件夹/Data/macro

这里面有两种类型的文件，一种是.ks，另外一种是.tjs。
不管哪一种都可以用最普通的文本编辑器比如记事本打开。

.tjs 部分是用 THE NVL Maker 的界面编辑器来修改的，用来记录你的界面背景，按钮图片，位置，字体等等信息。就不要随便动它了。

至于.ks，不就是之前已经见过的脚本吗。
只不过 scenario 文件夹下的部分是剧情脚本，这里则是系统脚本。
所有以“macro_”开头的都是宏设置。而其他的那些就是每个界面的具体代码了。

至于“系统设定”这个界面，就在 option.ks 里。

（3）自己来修改 option.ks！

因为其他复杂的东西都写在了 macro_ui 里，所以这个脚本很短。
我相信你不会觉得这个东西看着很吓人的。

而且其他的部分都不要在意，我们只是需要增加一个按钮。
这时候请重点把眼光放在“;描绘各种 ABC”这行囧注释上面。
只要在这个注释下面加入按钮，就可以显示在界面上了。

（4）设定按钮位置

假设我们把这个按钮添加在（200,50）的位置。
那么就在两个“;描绘各种 ABC”下面都插入一行：
[locate x=200 y=50]

```
;-----------------------------------------------------------
;系统设定
;-----------------------------------------------------------
*start
[locksnapshot]
[tempsave]
;-----------------------------------------------------------
*window

[history enabled="false"]

[locklink]
[rclick enabled="true" jump="true" storage="option.ks" target=*返回]
```

```
[backlay]
[image layer=14 page=back storage=&"f.config_option.bgd" left=0 top=0
visible="true"]
;隐藏系统按钮层
[hidesysbutton page="back"]

[current layer="message4" page="back"]
[layopt layer="message4" visible="true" page="back" left=0 top=0]
[er]
;描绘各种 ABC
[locate x=200 y=50]

[button_option page=back]
[trans method="crossfade" time=500]
[wt]

[s]

*刷新画面
[current layer="message4"]
[er]
;描绘各种 ABC
[locate x=200 y=50]

[button_option page=fore]
[s]

*返回
[jump storage="main_menu.ks" target=*返回]
```

（5）加上按钮

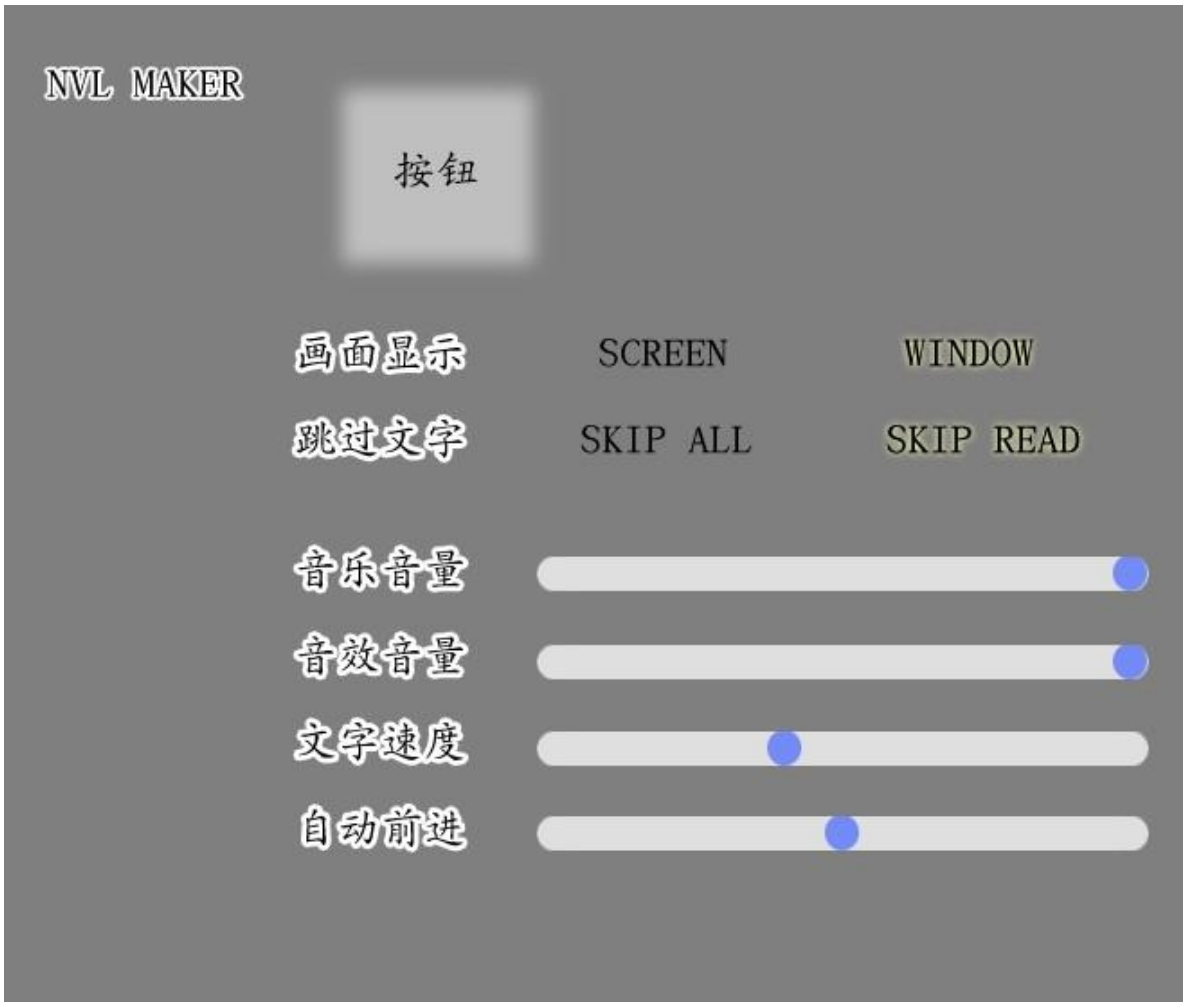
一般的按钮，使用[button]指令就可以了。重要的参数主要有以下几个：
Normal 按钮“一般”状态下的图片。
Over　按钮“选中”状态下的图片。
On　　按钮“按下”状态下的图片。
Target 按钮点下后，跳转到的标签。
因为现在没有别的图片，先随便代替一下吧。
[button normal=sample_off over=sample_on]

现在两个地方的内容应该都变成了这样：

```
;描绘各种 ABC
[locate x=200 y=50]
[button normal=sample_off over=sample_on]
```

第一个地方，是系统界面刚刚打开时显示的内容（渐变显示），第二个地方，则是点下任何一个按钮修改设定之后显示的内容（瞬间刷新）。
假如少加了其中一个地方，就会导致按钮在那个情况下不显示。

保存，测试下游戏。



现在系统设定界面上多了一个按钮。
虽然你怎么戳它都没有反应……

（6）添加标签

按钮戳下去以后没有反应，当然是因为没有设定接下来应该跳转到哪里。也就是没有“标签”。
现在，在 option.ks 里的最后，加入一个新的标签，叫“*恢复默认值”好了。
不过在做完“恢复默认值”的操作以后，肯定还要跳回来“*刷新画面”（把修改的值反映在界面上）。
所以，继续加上新的一行。

[jump storage="option.ks" target=*刷新画面]

同时按钮的定义也改成这样：
[button normal=sample_off over=sample_on target=*恢复默认值]

现在你的 option 脚本应该是这样的：

```
;-----------------------------------------------------------
;占位用的系统设定
;-----------------------------------------------------------
*start
[locksnapshot]
[tempsave]
;-----------------------------------------------------------
*window

[history enabled="false"]

[locklink]
[rclick enabled="true" jump="true" storage="option.ks" target=*返回]

[backlay]
[image layer=14 page=back storage=&"f.config_option.bgd" left=0 top=0
visible="true"]
;隐藏系统按钮层
[hidesysbutton page="back"]

[current layer="message4" page="back"]
[layopt layer="message4" visible="true" page="back" left=0 top=0]
[er]
;描绘各种 ABC
[locate x=200 y=50]
[button normal=sample_off over=sample_on target=*恢复默认值]

[button_option page=back]
[trans method="crossfade" time=500]
[wt]

[s]

*刷新画面
[current layer="message4"]
[er]
;描绘各种 ABC
[locate x=200 y=50]
```

```
[button normal=sample_off over=sample_on target=*恢复默认值]

[button_option page=fore]
[s]

*返回
[jump storage="main_menu.ks" target=*返回]

*恢复默认值

[jump storage="option.ks" target=*刷新画面]
```

（7）添加功能

加上标签以后，依然没有任何其他操作，因此点下按钮以后还是和原来一样。
接下来要做的就是在

```
*恢复默认值

[jump storage="option.ks" target=*刷新画面]
```

的中间插入内容。
既然说是恢复默认值，那么先来假定几个默认值吧。
默认游戏是窗口显示，只跳已读内容，并且音乐和音效音量都是 50。
文字速度和自动前进速度都在中间档（0~10）的 5。
把这些东西加上看看：

```
*恢复默认值
[eval exp="kag.fullScreen=false" cond="kag.fullScreen"]
[eval exp="kag.allskip=false"]
[eval exp="kag.bgmvolume=50"]
[eval exp="kag.sevolume=50"]
[eval exp="kag.textspeed=5"]
[eval exp="kag.autospeed=5"]
[jump storage="option.ks" target=*刷新画面]
```

现在，测试下游戏。

画面是不是变得很整齐了？www

三、扩展内容

（1）用 eval 执行 TJS 式

其实刚刚的各种[eval exp="TJS 式"]，就跟之前点下选择按钮时候执行的 TJS 式是一样的，可以对变数的值进行操作。

对应脚本编辑器里的这个指令：

（2）cond 属性和条件分歧

[eval exp="kag.fullScreen=false" cond="kag.fullScreen"]，这里的 cond="TJS 式"]，则相当于一个条件分歧。只有当“kag.fullScreen”的值为真时才执行。
（如果本来就是窗口，还是强制执行 kag.fullScreen=false 的话，在这个版本里会导致画面卡住，所以加上了这个判断。）

也就是说，

```
[eval exp="kag.fullScreen=false" cond="kag.fullScreen"]
```

等价于下面的三行，

```
[if exp="kag.fullScreen==true"]
[eval exp="kag.fullScreen=false"]
[endif]
```

（PS：kag.fullScreen 和 kag.fullScreen==true 是一个意思。）

cond 属性在判断是不是要执行单条指令的时候比 if，endif 轻松得多。
在自己撰写脚本的时候，请多利用。

（3）TJS 段落

不过刚刚这样一行一行的[eval exp="TJS 式"]是不是也挺烦人的？

其实，它也可以写成下面的样子：

```
*恢复默认值
[iscript]
if (kag.fullScreen) kag.fullScreen=false;
kag.allskip=false;
kag.bgmvolume=50;
kag.sevolume=50;
kag.textspeed=5;
kag.autospeed=5;
[endscript]
[jump storage="option.ks" target=*刷新画面]
```

[iscript]和[endscript]标起来的段落，是书写大段 TJS 代码的地方。
在.ks 脚本里插入 TJS 部分的时候经常会用到。

（4）“kag.”开头的变数

没有错，这些 kag.开头的东西，和 f.开头的东西都一样是变数。

只不过他们是 kag 系统提供的内置变数。
许多无法通过 KAG 指令实现的东西都可以通过修改这些变数来达成效果。

修改系统设定的值时，最常用的已经在上面列出来了。就是这些：

kag.fullScreen 是否全屏
kag.allskip 是否跳过未读部分
kag.bgmvolume 音乐音量
kag.sevolume 音效音量
kag.textspeed 文字速度
kag.autospeed 文字自动前进速度