

# MobileCreator 移动跨平台集成开发环境



## 中文使用说明书 V 1.6

福州动友网络科技有限公司

2013 年 11 月

\* 鉴于本产品还在不断的改进过程中，本手册仅供使用参考，如遇手册和实际操作不一致的情况，请以软件产品的实际操作结果为准。

## 目录

|                   |     |
|-------------------|-----|
| 软件概述.....         | 4   |
| 主要功能.....         | 5   |
| 支持平台.....         | 6   |
| 系统配置需求.....       | 6   |
| 获得帮助.....         | 6   |
| 开始使用.....         | 6   |
| 工作方式.....         | 6   |
| 界面.....           | 7   |
| 主界面.....          | 7   |
| 菜单栏.....          | 8   |
| 角色面板.....         | 22  |
| 计时器面板.....        | 32  |
| 路径面板.....         | 33  |
| 区域面板.....         | 34  |
| 已有角色.....         | 35  |
| 事件.....           | 35  |
| 行为.....           | 50  |
| 脚本.....           | 71  |
| 脚本编辑器.....        | 71  |
| 脚本参考.....         | 79  |
| 特殊角色.....         | 79  |
| 角色属性.....         | 79  |
| 系统函数.....         | 83  |
| 一般函数.....         | 83  |
| 绘制及截图函数.....      | 101 |
| 音视频函数.....        | 105 |
| 文件/变量的基本操作函数..... | 109 |
| 文件/变量的读写函数.....   | 110 |
| 文件/变量的删除函数.....   | 112 |

|                 |     |
|-----------------|-----|
| 文件/变量保存函数.....  | 112 |
| SQLite 数据库..... | 113 |
| 游戏控制器.....      | 120 |
| 键盘函数.....       | 122 |
| 网络相关函数.....     | 129 |
| 物理世界函数.....     | 141 |
| 手势支持函数.....     | 150 |
| 输入框函数.....      | 153 |
| 运营充值函数.....     | 153 |
| 部分绘制函数.....     | 156 |
| 定位函数.....       | 157 |
| HTML 函数.....    | 157 |
| 帮助函数.....       | 158 |
| 设备鉴定函数.....     | 170 |
| 其他设备相关函数.....   | 171 |
| 标准 C 函数.....    | 172 |

## 软件概述

MobileCreator 是一款主要用于开发游戏的，具有跨 PC 和手机平台特性的集成开发环境。

它解决了目前手机游戏开发中遇到的四个最重要的瓶颈问题：

- **原生工具复杂：**各平台原生开发环境各不相同，平均每个平台至少需要三个月研究才能正式进行游戏类产品开发；
- **人力成本高：**需要的人力资源成本较高，需要熟练掌握游戏开发方法的人员才能形成一定的开发能力；
- **游戏架构专业：**没有专业的游戏搭建平台，完全靠编程人员从头构建，开发人员需要编写大量非游戏逻辑代码以保证游戏的稳定性；
- **代码移植性差：**平台之间代码复用性低，移植性差，大部分时候必须为每个平台单独编写代码。

针对上述四个瓶颈，MobileCreator 有四大特色功能帮助解决：

- **一套环境，一次开发：**无论是开发 iOS 还是 Android 游戏，都无需购买 MacPC 或者安装各种 SDK，只需要可以绿色安装的 MC，立刻就可以开发适应多种平台的游戏；
- **入门门槛低，开发速度快：**所见即所得的界面搭建方式，会简单编程就能做商业游戏，愤怒的小鸟玩法部分 5 分钟即可搭建完成；
- **专业游戏组件，快速开发复杂游戏：**游戏设计所需的图形图像处理功能无需编程即可获得，实现透明、移动、旋转缩放、色调变换等效果，内置小型文件型数据库，支持各种中文字体，支持网络访问 SQL 数据库；
- **一键导出，快速部署：**在 MC 上开发的代码，无需做修改，即可通过一键打包方式在 iOS、Android 或是直接在 Windows 上运行。

综上所述，MobileCreator 的出现，很好的解决了目前游戏开发中遇到的最大的瓶颈——跨平台问题，其生成的游戏运行效率接近采用原生工具开发的游戏。是跨平台游戏开发最好的解决方案之一。



## 主要功能

- 多人网络游戏
- 根据事件开发
- 传递事件至另一个角色
- 在单一文件中包含多个级别
- 移动角色穿过复杂地图
- 帮助角色
- 透明角色
- 无限制角色
- 角色深度控制
- 克隆和删除角色
- 每一个角色有各种动画
- 设置父角色角色
- 创建角色路径
- 全屏或窗口模式
- 可配置分辨率和帧速率
- 可配置音频格式
- 像控制角色一样控制 view
- 平铺绘制工具
- 全局和本地脚本
- 全局和本地角色变量
- 周期和随机计时器
- 角色旋转缩放
- 角色色调变化
- 物理世界支持
- 电话/短信支持
- 手势支持
- 读取 jpeg, gif, png, bmp, pcx, tga, xpm, xcf, lbm and tif 格式图片
- 读取 Ogg, wav 音乐/声音文件

- 导出生成的 .dat 文件可以在 Android 和 iPhone 上使用

## 支持平台

- Windows 2000/2003/XP/Win7
- Android 2.2 以上
- iOS 4.0 以上

## 系统配置需求

Windows XP SP3 以上

1GB RAM 以上

x86 CPU (Pentium, AMD, ...)

2.0 GHz

3D 显卡 (OpenGL and Direct3D hardware acceleration recommended)

## 获得帮助

### 文档

MC 提供完整的使用说明让你熟悉 MC 的世界。

MC 有许多特色来帮助开发者创建游戏世界。

### 技术支持

MC 全体工作人员同样提供在线和电子邮件技术支持：

技术支持邮件：[mc@dongyo.cn](mailto:mc@dongyo.cn)

MC 论坛：<http://www.dongyo.cn/bbs/forum.php>

## 开始使用

## 工作方式

### 基本概念：

MC 是以行为(events)和事件(actions)为基础的。它采用所见即所得的直观方式来创

建游戏。想象你有一个很大的桌子。现在，桌面上的角色组成了你的游戏。在这张桌子上你能看到你所有的游戏，对角色的定位有一个很好的概念。这就是 MC，它就是你的大桌子，你能以很直观的方式对角色进行操作。

在使用 MC 前，必须记住角色 (Actor)，动画 (Animations)，事件 (Events) 和行为 (Actions) 的概念。

### 角色 (Actors):

角色在 MC 的世界里可以是游戏中的角色、背景或者物品之类。每一个角色可连接一组连续的动画 (animations)，能以另外一个角色作为父角色，跟随一个路径，可以是透明的，可以是一个文本等等。

### 动画 (Animations):

动画可以是一个静态的图片或一组有顺序的图片，用来在屏幕上绘制角色。

注意，并不是所有的动画都会动！

### 事件 (Events):

事件就是游戏里发生的事件，例如当用户按下一个按键或者点击了鼠标，或者一个 actor 碰撞了另一个。

### 行为 (Actions):

行为是在一个事件发生后该发生什么来响应。例如当一个角色与火箭碰撞后，应该产生一个爆炸。MC 有一套预定义的行为，任何事件可触发一个或多个行为。

这四个基本概念组合在一起，可以用逻辑的方式表述为：让**角色**中的某组**动画**当某个**事件**发生的时候做某个**行为**。

## 界面

### 主界面

在关闭欢迎界面后，你将看到 MC 的主界面。

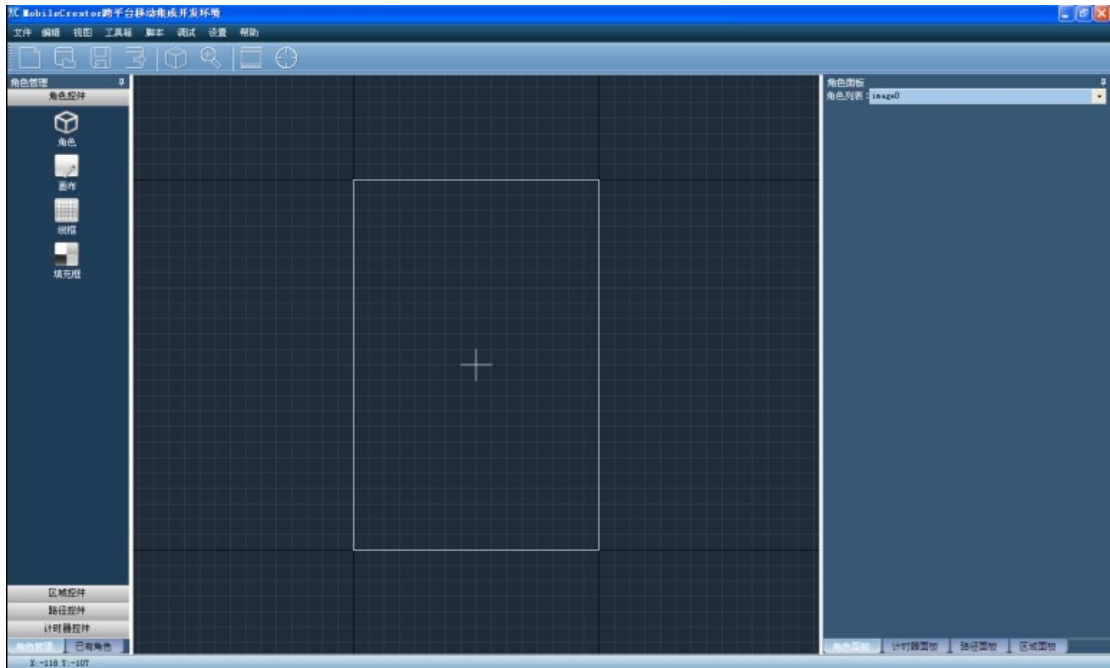


图 1

其中最中心最大的一块区域叫主编辑区域，将是你用来编辑游戏的主要区域。主编辑区域像一张很大的网格纸。网格的尺寸和颜色可以重设：选择“设置->编辑器设置”。

白色的矩形是一个特殊的角色，叫“view”。view 像相机一样调节游戏的某部分，使它在某一个给定的时刻显示。它拥有和用户创建的角色一样的属性，但是不能设置动画或文本。view 的尺寸就是游戏的分辨率：320×480，480×800，1024×768 等等。屏幕分辨率可改变：”设置->游戏设置”。

图 2

## 菜单栏

## 文件



图 3



图 4

**新建游戏：**清空当前游戏并开始设计新游戏。

**载入游戏：**清空当前游戏并读取之前保存的游戏

**合并游戏：**合并当前游戏与新读取的游戏。（如果读取的游戏中有角色名和当前游戏的角色名相同，那结果将是不可预料的。）

**保存：**保存当前游戏所做的改动。

**另存为：**在另外一个位置或取一个新名字保存当前游戏。

**导出：**将游戏打包。

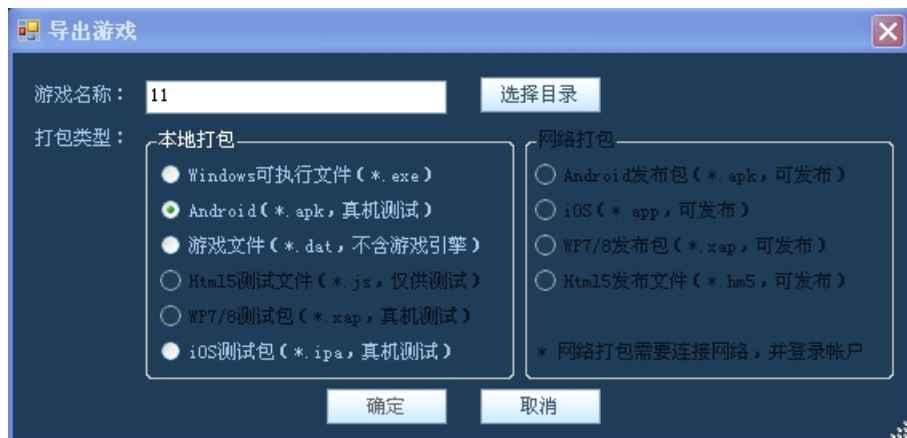


图 5

有效的打包类型包括：

- Windows: .exe 可执行文件（可兼容的 Windows 2000/2003/xp/Win7）。
- Android 测试包：.apk 真机测试。
- 游戏文件：.dat 不含游戏引擎。
- iOS 测试包：.ipa 真机测试。

**最近文件：**单击【文件】，从弹出的下拉菜单中选择【最近文件】，可以看到最近打开的文件有哪些，也可以选择一个载入。

**退出：**退出 MC

**工具箱：**

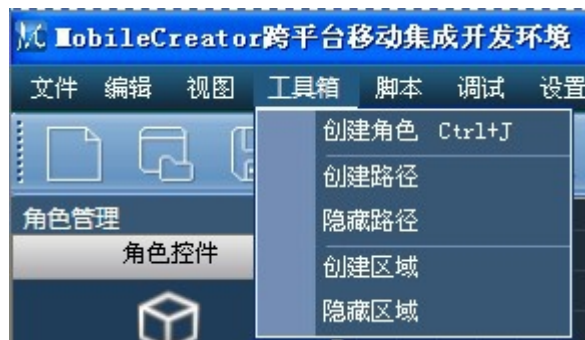


图 6



图 7

**角色名：**角色的名字必须以字符开头，由字符、数字或下划线组成。不能包含脚本方法和变量（“vars”）名。

有效的：MyActor, ball, dog\_catcher

无效的：/right, CreateActor, fgets

**角色控件：**

在 MC 的左侧会显示角色管理中你可以创建的类型：角色，画布，线框，填充框等。



图 8

**角色：**可接收动画的角色(与角色相对应的图片文件)。此角色能被用来创建程序中最多最全的行为。



图 9

**画布：**当你使用画布时，想象自己是小毕加索。用 drawing 方法在此角色上画画。此角色可画直线，圆等等。

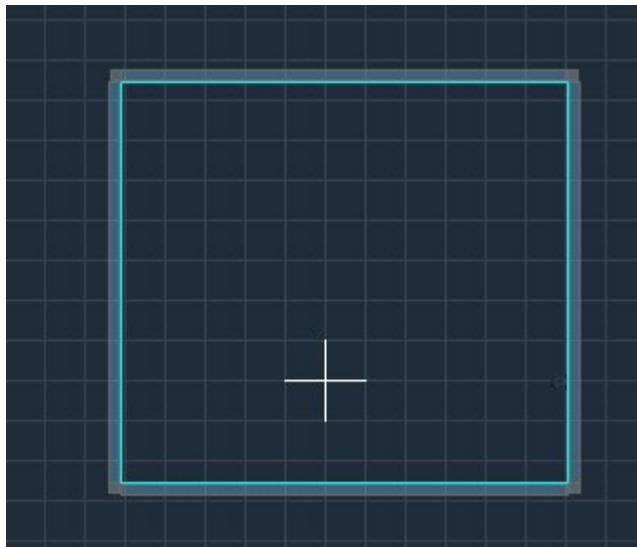


图 10

画布在 MC 内为深蓝色长方形, 可移动或者调整大小。矩形在测试模式下是看不见的。

**线框:** 它用来做传感器实现行为, 诸如像“打开门”的时候。比如, 一个普通的角色进入线框时, 它也可以用来创建障碍物反弹某个角色, 比如球。例如在弹珠游戏里, 球会被一个线框类型的障碍物弹开。线框不支持鼠标点击事件。

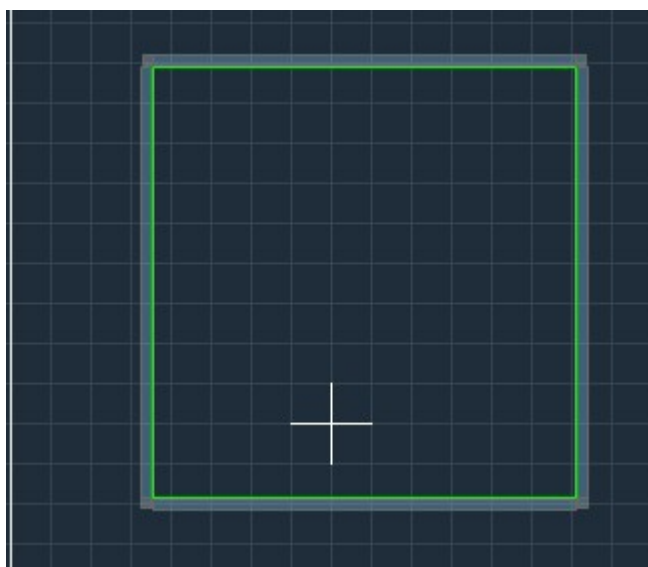


图 11

线框在 MC 内显示为一个绿色的矩形。可以移动或者调整大小。矩形在测试模式下是看不见的。

**填充框:** 和线框不同, 填充框像普通角色一样支持鼠标点击。允许用户创建热点或者传感器来响应鼠标点击事件而无需使用普通角色。



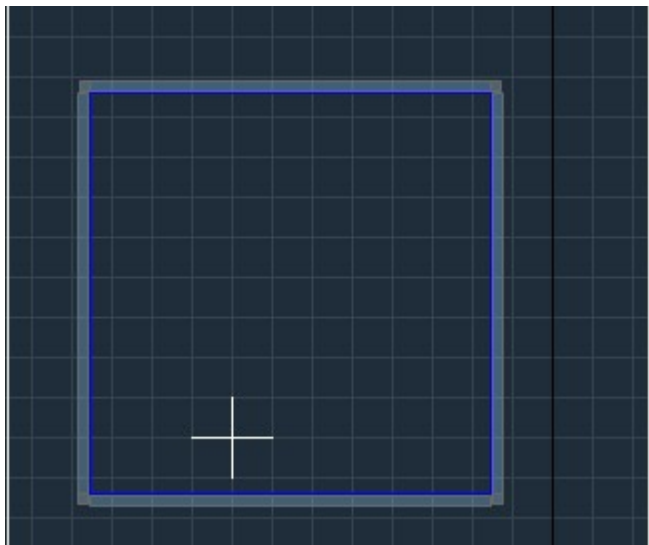


图 12

填充框在 MC 内显示为一个蓝色矩形。可以移动或者调整大小。这些矩形在测试模式下是看不见的。

路径控件：

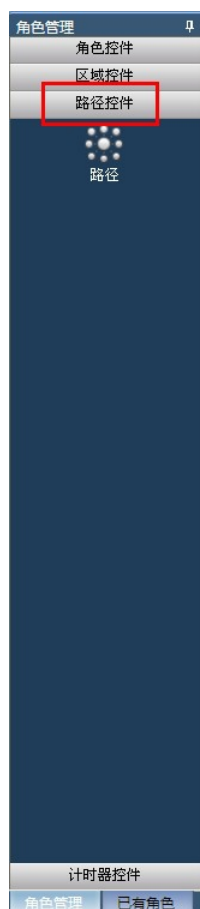


图 13

创建路径：创建一个新路径并且编辑。

移除路径：移除当前已有路径。

路径列表：选择一条已有路径进行编辑。

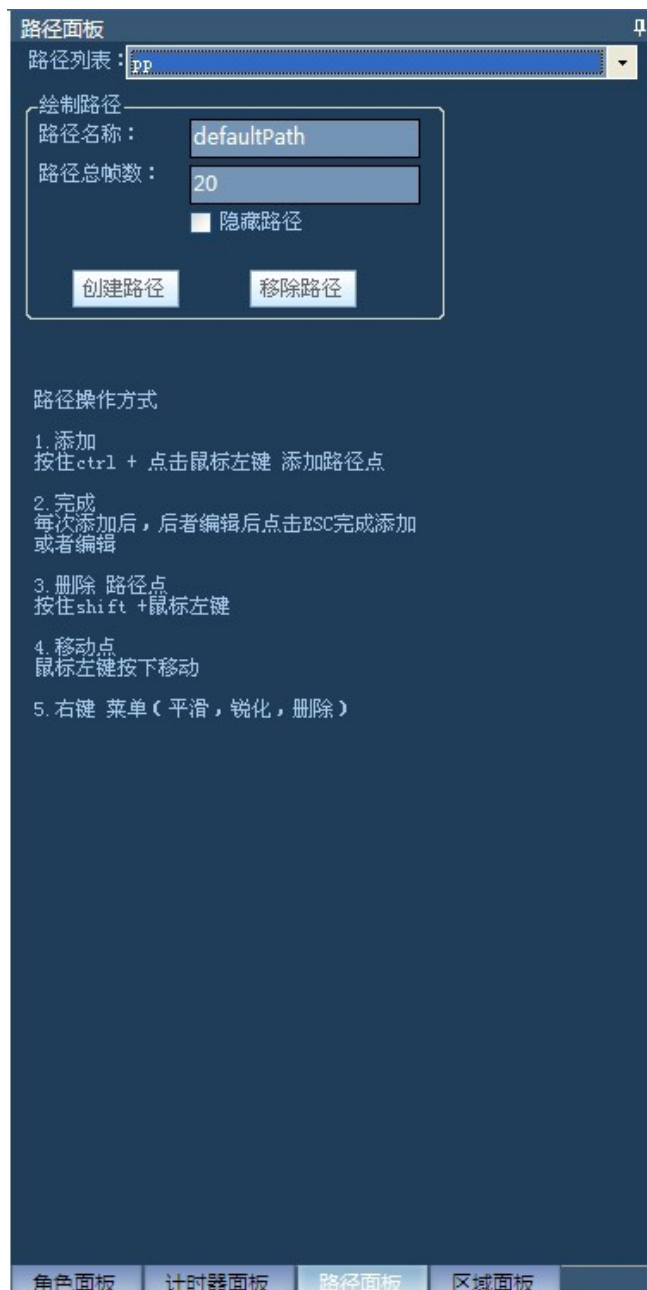


图 14

添加/移动路径节点：

- 按住 Ctrl+鼠标左键，添加路径节点。
- 当绘制结束时松开 Ctrl 键退出编辑状态。
- 在路径节点上按住鼠标左键可以拖动该节点。

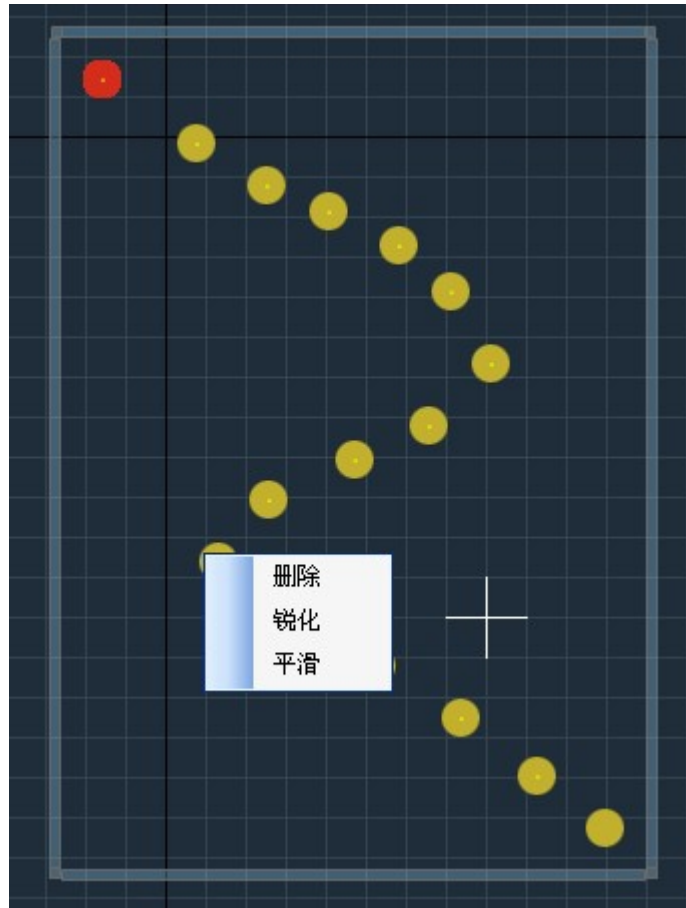


图 15

在路径节点上右击：

- 删除:删除当前节点。
- 锐化:设置当前节点为直线。
- 平滑:设置当前节点为曲线。

注意：鼠标点击拖动第一个路径节点就能移动整个路径。

**区域控件：**



图 16

**活动区域:**

活动区域是一个包含角色在内的边界框区域，区域和里面的角色只有在和 view 相交时才会被读取。当区域与 view 不再相交时区域和在区域内的所有角色将被删除。

活动区域在 MC 内显示为一个黄色矩形。可以移动或者调整大小。矩形在测试模式下是看不见的

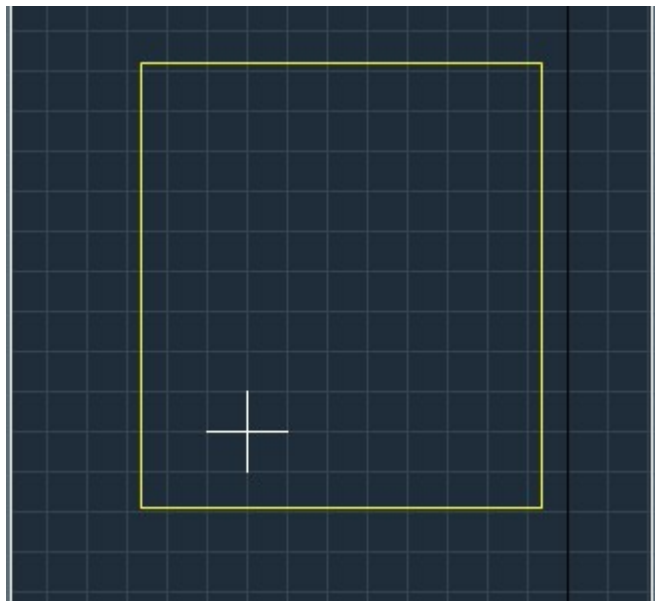


图 17

右键点击活动区域的边界会弹出菜单，该菜单可删除活动区域。

当一个角色移出活动区域，只有当 view 和它不再相交时，角色才会被删除。当角色上升到 view 范围之外，会被活动区域删除。可以定义很多重叠的活动区域。所有活动区域内的角色，角色的父角色和角色说涉及到的脚本都会被读取。

用“创建角色”方法创建的角色会被分配到可见的活动区域。如果 view 离开区域之后又回来，所有的角色都会被重新读取，用“销毁角色”方法删除的除外。任何角色都能用“创建角色”方法读取，即使已被删除。

如果没有创建任何活动区域，所有的角色都默认将在游戏开始的时候被读取。

下面是一个有三个活动区域的多图层游戏：



图 18

显示/隐藏区域：

显示或隐藏活动区域和区域角色。

## 脚本

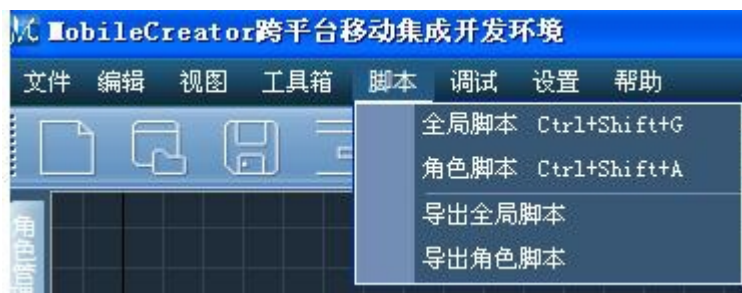


图 19

选择全局代码选项可自由添加任何 C 代码(数组, 结构体和函数)。

在脚本菜单 C 程序员能直接使用 C 语言的全部功能。

## 调试

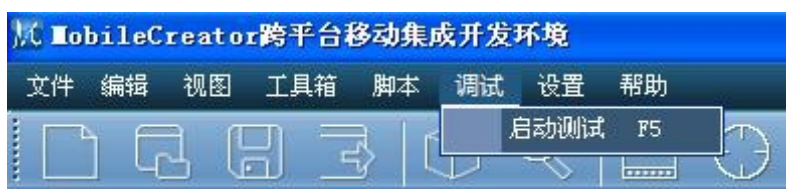


图 20

让 MC 模式下进行 运行/测试游戏。按下<ESC>退出“游戏测试”回到编辑界面。

## 设置



图 21

游戏设置:

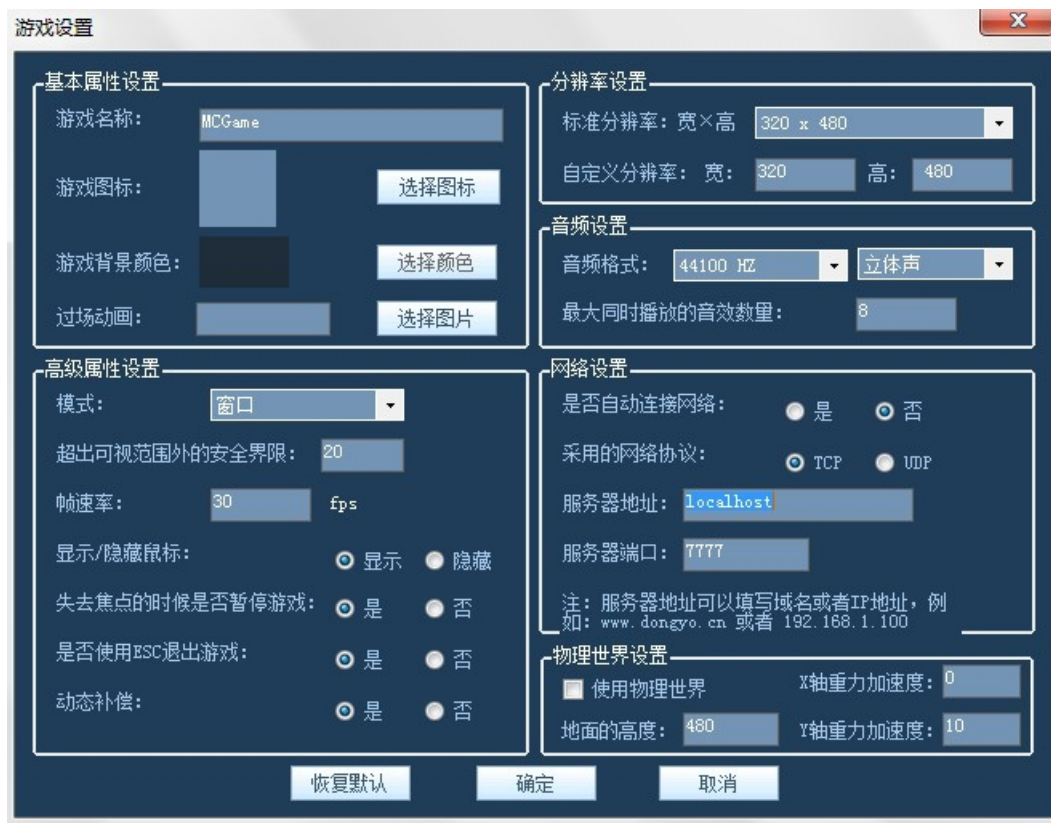


图 22

**基本属性设置:**

- 游戏名称: 指定当游戏打包时的标题。
- 游戏图标: 指定游戏图标。
- 游戏背景颜色: 游戏运行的背景颜色。
- 过程动画: 设置游戏载入时的等待画面。

**高级属性设置:**

- 模式: 指定窗口或全屏模式。
- 超出可视范围外的安全界限: 在使用“在可视范围外”事件时增大 view 的区域。
- 帧速率: 指定游戏的画面绘制的频率(帧/秒)。
- 显示/隐藏鼠标: 指定是否显示鼠标光标。
- 失去焦点时是否暂停游戏: 设置成“YES”, 当窗口不是激活状态, 或者掌上电脑/手提电脑/智能手机接收系统信息时, 暂停游戏。
- 是否使用 ESC 键退出游戏: 选择“是”, 当按下 ESC 按钮时退出游戏。如果你不想使用 ESC 按键来退出, 请选择“否”。可以在任何行为上使用 ExitGame() 方法来退出。
- 动态补偿: 选择“是”, 如果角色帧率慢于游戏帧率的话, 调整角色至恰当的帧率。

**分辨率设置：**

- 标准分辨率：选择常见的分辨率。
- 自定义分辨率：用户自行输入游戏分辨率。

**音频设置：**

- 音频格式：设置声音采样频率，单声道或立体声。
- 最大同时播放的音效数量：一般设置为 8 个。

**网络设置：**

- 是否自动连接网络：选择“是”游戏开始时会自动连接到网络；选择“否”，则不连接。
- 采用的网络协议：TCP 是点对点的协议，相对可靠；UCP 是广播的协议，相对不可靠。
- 服务器地址：可填写域名或 IP 地址。
- 服务器端口：服务器的端口号

**物理世界设置：**

- 使用物理世界：勾选则表示模拟物理世界
- 地面的高度：距离地面的高度。
- X 轴、Y 轴的重力速度：物理世界存在一定的加速；X 轴正值向右，负值向左，Y 轴正值表示向下，负值表示向上。

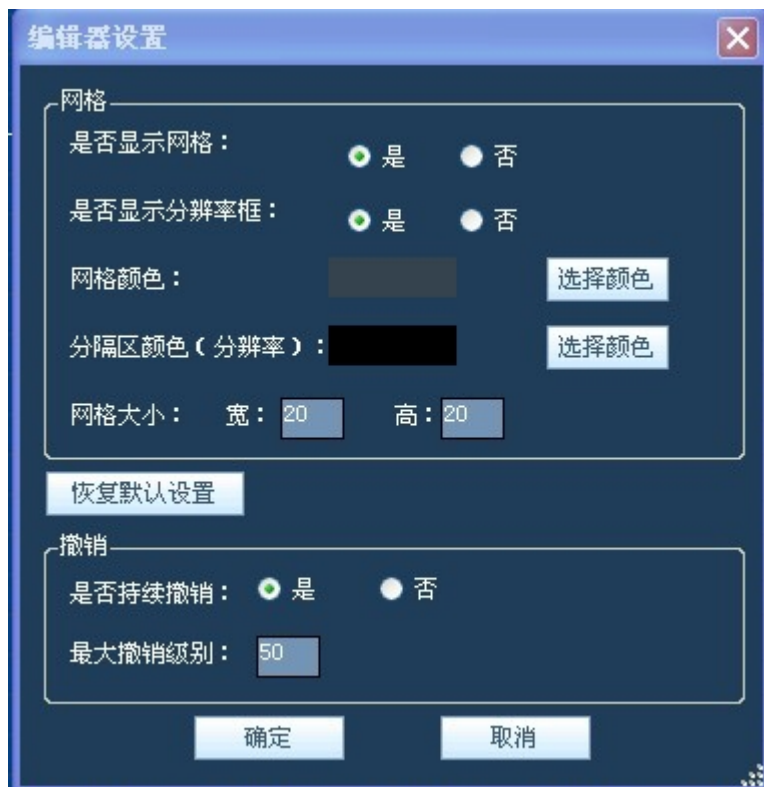
**编辑器设置：**



图 23

**网格：**

- 是否显示网格：显示或隐藏主编辑区域中的网状小格子。
- 是否显示分辨率框：按分辨率大小显示框格。
- 网格颜色：设置网状小格子的颜色。
- 分隔区（分辨率）颜色：设置游戏分辨率框格的边缘颜色。
- 网格大小：设置网状小格子的大小。

**撤销：**

- 持续撤销：如果你选择“是”MC 将会查找对应的. undo 文件并在下次读取游戏时恢复。
- 最大撤销级别：记录撤销的次数；只有当“持续撤销”选项设置为“是”时，此设置才会创建空间。减少“最大撤销级别”设置来减少 undo 文件的大小。

**帮助**

图 24

**状态栏：**

图 25

鼠标相对于游戏中心的坐标。在点击到角色时，显示相对于其父角色的坐标。

**工具栏：**

图 26

常用功能快捷按钮：新建游戏、载入游戏、保存、导出、创建角色、查找角色、测试游戏、返回中心。

## 角色面板

角色面板赋予你创造独一无二的行为事件的能力。角色面板可以设置角色上显示动画或文本的操作，跟随某个路径，并且/或者当有分配到事件或行为时作出特定的响应。角色面板是一个强大的功能。如果你想打造成功的游戏，最关键的是你必须熟悉该面板的所有功能。



图 2

角色面板提供无数有用的功能。接下来我们将讨论每一个功能和它的作用。



图 27

### 角色列表：

单击此选项弹出的菜单会显示一个列表，该列表包含游戏里的所有角色，可选择任何一个角色进行修改或编辑。

- **角色名：**

指示当前正在编辑的角色，可以命名/修改该角色。

- **克隆序列：**

对克隆的角色进行区分，序号从“0”开始。序号“0”表示没有任何克隆的角色本体。

- **接收可视范围外事件：**

勾选该选择则表示该角色在可视窗口之外仍然接收事件，反之则不接收。

- **启动时创建：**

角色可以在游戏启动的时候或者之后创建。此选项指定角色会在游戏启动时被创建，如果你不选择“是”，你可以使用“创建角色”行为来创建。

### 位置大小设置：



图 28

- **X、Y：**

设置角色起始位置，以主编辑区域中心为原点，向右和向下为正值，向左和向上为负值。

- **宽、高：**

显示当前角色的原始尺寸，只读，不可设置。

- **比例：**

设置当前角色缩放的与原尺寸的比例。

- **角度：**

设置当前角色沿动画中心点旋转角度。

动画设置：

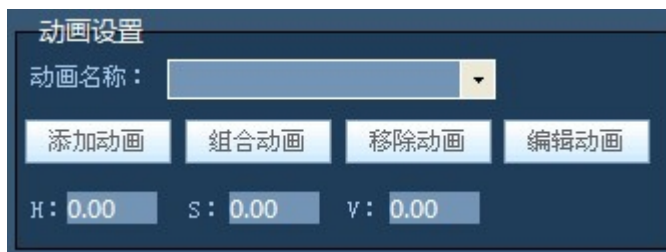


图 29

为角色添加、组合、编辑或删除动画。

- **添加动画：**

在选择的角色上添加一个新动画。



图 30

- 动画名称：指定动画名称，默认与文件名相同。
- 打开文件：选择动画文件，支持格式：jpeg, gif, png, bmp, pcx, tga, xpm, xcf, lbm, tif。
- 单文件/多文件：指定单一文件中存在的一组动画还是由多个文件组成的一组动画。
- 是否自动透明：勾选则自动设置该动画透明，否则该动画留有白边。

- 横/纵向帧数：如果动画包含在一个单一文件里，指定水平和垂直帧数，设置后将按指定帧数平均分割此动画。并且自动按从上到下，从左到右方式播放和编号。
  - 帧速率：指定每秒播放动画的帧数。
- **组合动画：**
- 定义基于动画的帧序列。

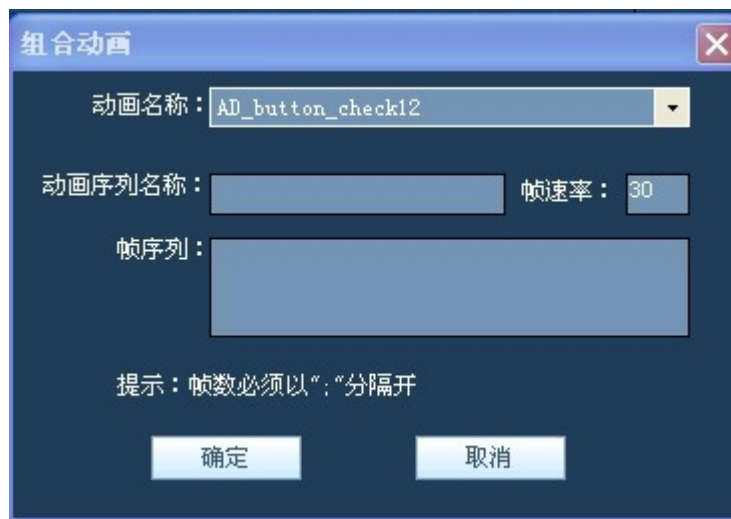


图 31

- 动画名称：在列表中选择待重新组合的动画。
  - 动画序列名称：指定新的动画序列名。
  - 帧速率：指定每秒播放动画序列的帧数。
  - 帧序列：输入一个新的帧序列。（例如：14;2;9;22;2;3;4...）帧数必须以“;”分隔开。
- **编辑动画：**
- 编辑已有的序列或覆盖动画文件。



图 32

- 指定帧：指定从第几帧开始播放。

### 文本设置：

指定角色文本与格式(文本角色不支持碰撞事件)，字体在设置之后不能继续添加动画，只能通过删除角色来重新设置动画。



图 33

### - 文本内容：

在此输入文本并且编辑字体样式。如果没有输入，则该角色之前输入的文本也将被删除，同时也无法编辑字体。

### - 普通字体：

在字体设置框中设置字体、字型、大小、平滑、颜色等。



图 34

### - 图片字体：

用图片文件创建字体。

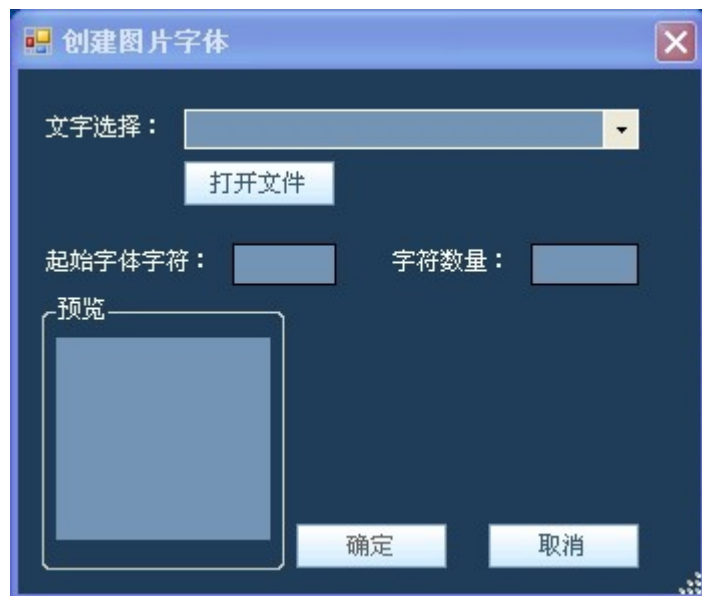


图 35

- 文字选择：读取图片文件(jpeg, gif, png, bmp, pcx, tga, xpm, xcf, lbm 和 tif)  
图像文件在一行或一列中必须包含所有字符图像(以 ASCII 表为序)
- 起始字体字符：设置在读取的文件中设置起始图片代表的字符
- 字符数量：设置读取文件中的字符个数

#### 特性设置：

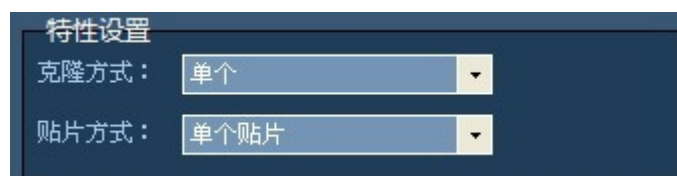


图 36

#### - 克隆：

克隆一个已创建的角色。该角色共享相同的行为和动画。

- 单个：此选项只克隆单个角色
- 序列：创建一个矩形的角色数组。通过改变参数可实时观看到变化。

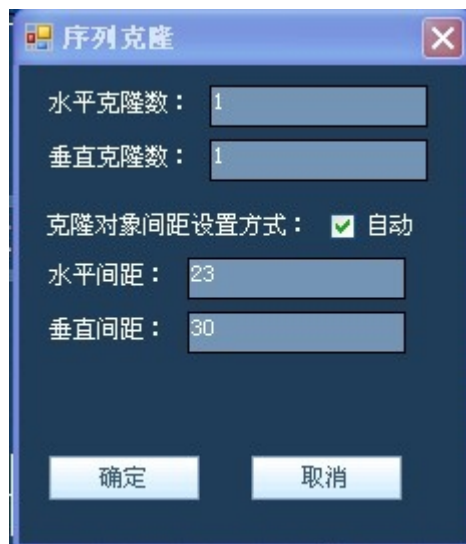


图 37

- ✓ 水平克隆数：指定单行的克隆数。
- ✓ 垂直克隆数：指定单列的克隆数。
- ✓ 克隆角色间距设置方式：选择“自动”自动排列。
- ✓ 水平距离：每行角色与角色之间的间距。
- ✓ 纵向距离：每列角色与角色之间的间距。
- 沿着路径：创建克隆角色跟随一个选中的路径。改变参数可实时观看变化。



图 38

- ✓ 路径名称：从下拉框中选择一个已设置好的路径。
- ✓ 克隆个数：在选择的路径中指定克隆角色的数量。
- 贴片：
  - 在角色上绘制贴片
  - 单个贴片：创建单一的贴片



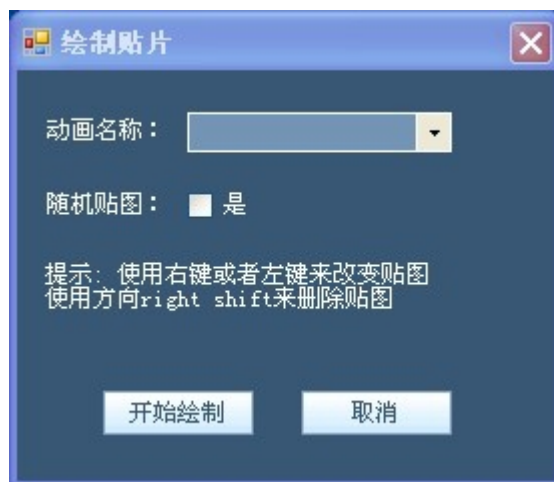


图 39

通过左右方向键改变动画帧。使用鼠标左键来增加贴片，鼠标右键删除贴片。

**特殊属性：**



图 40

**- 透明度：**

透明度设置角色的透明度。角色透明度如果大于 50%就会开始显示背景的部分图片。如果设置成 0%（默认）则完全不透明。

**- Z 轴深度：**

此选项设置游戏里角色的深度。比如，游戏里有树角色)和兔子角色)，你要设置树角色在兔子角色的后面，才能让兔子显示在树的前面。

**- 父角色：**

设置角色的父角色(游戏里的任何角色都可以作为父角色,包括"view"或者不设置父角色)。如果一个角色有了父角色，当父角色移动时，子角色也跟着移动。

**- 路径：**

设置角色的路径（路径可以是已设置的、“*random path*”，或“*(none)*”）。

- **填充扩展：**

- 正常：只显示一个角色。
- 平铺：角色按一定比例铺满所有界面。
- X 轴平铺：角色按一定的比例在 X 轴上平铺。
- Y 轴平铺：角色按一定的比例在 y 轴上平铺。

- **事件继承于：**

使当前角色继承从选择的角色传来的事件。当前角色将拥有所有的事件加上选择的角色的事件。

例子：

你有许多不同的”敌人”角色可以左右走和跳。

可以创建一个”基础敌人”角色并添加事件来移动：

右方向键：向右移动

左方向键：向左移动

上方向键：跳

在选择“敌人”角色后“事件继承于”选择“基础敌人”。

如果你想让一些敌人在不同的窗口跳，只需要新添加一个上方向键的“按键按下”事件。

当用户按下上方向键时，执行的事件就是“敌人”角色里定义的事件。

**移除角色：**在角色上点击鼠标右键，选择删除选中的角色

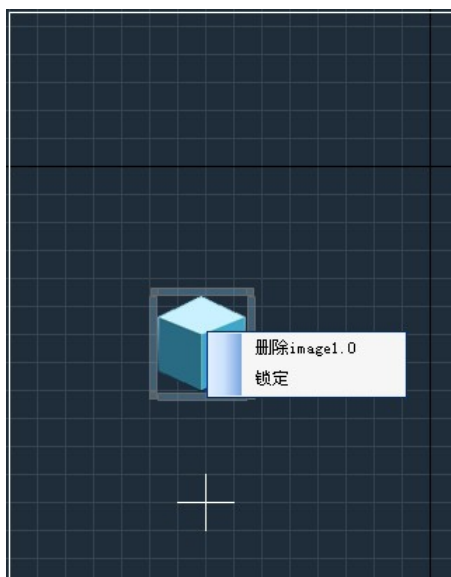


图 41

物理属性：



图 42

模拟真实世界中的物体，存在物体密度、形状、力、速度、加速度等。开启物理世界必须在“设置”→“游戏设置”→“物理世界”→勾选“使用物理世界”。

- **是否开启物理属性：**

当游戏世界需要模拟物理世界时，都需要勾选该选项。

- **静态物体：**

静止的物体，处于悬浮状态或与任何物体发生碰撞时都不会发生位移。通常用来设置成物理世界的墙或者不动的柱子等物体。

- **矩形物体：**

矩形、类似矩形的物体都可以勾选。

- **圆形物体：**

圆形，类似圆形的物体都可以勾选。

- **X、Y 轴的初始速度：**

物理世界存在加速度，所以需要设置角色 X、Y 轴方向的初始速度

- **密度：**

物体的密度，用来替代设置物体的质量。密度越大，质量越大。

- **摩擦系数：**

两个物体相互摩擦力的大小，系数越大摩擦力越大。

- **弹力系数：**

两个物体相互碰撞产生的弹力的大小，弹力系数越大弹力也越大。

- **空气阻力:**

物体在上升或下降中会有空气阻力，阻力越大，物体的运动速度越来越慢。

- **旋转阻力:**

物体在做旋转运动时会有旋转阻力，阻力越大，旋转速度越慢。

## 计时器面板



图 43

- **计时器列表:**

如果创建了多个计时器，可以通过该列表选择。

- **计时器名称:**

显示计时器的名称。

- **计时器类型:**

分为 Periodic（周期性）和 Random（随机性）两种的计时器类型。

- **运行间隔:**

计时器运行的时间间隔。最小不低于 12ms。

- **运行最小间隔:**

计时器运行的最小间隔。

- **运行类型：**  
分为 Forever（永久的）和 Specify Quantity（指定数量的）指定的两类
- **运行次数：**  
指定计时器运行的次数。仅在运行类型为 Specify Quantity 时有效。

## 路径面板

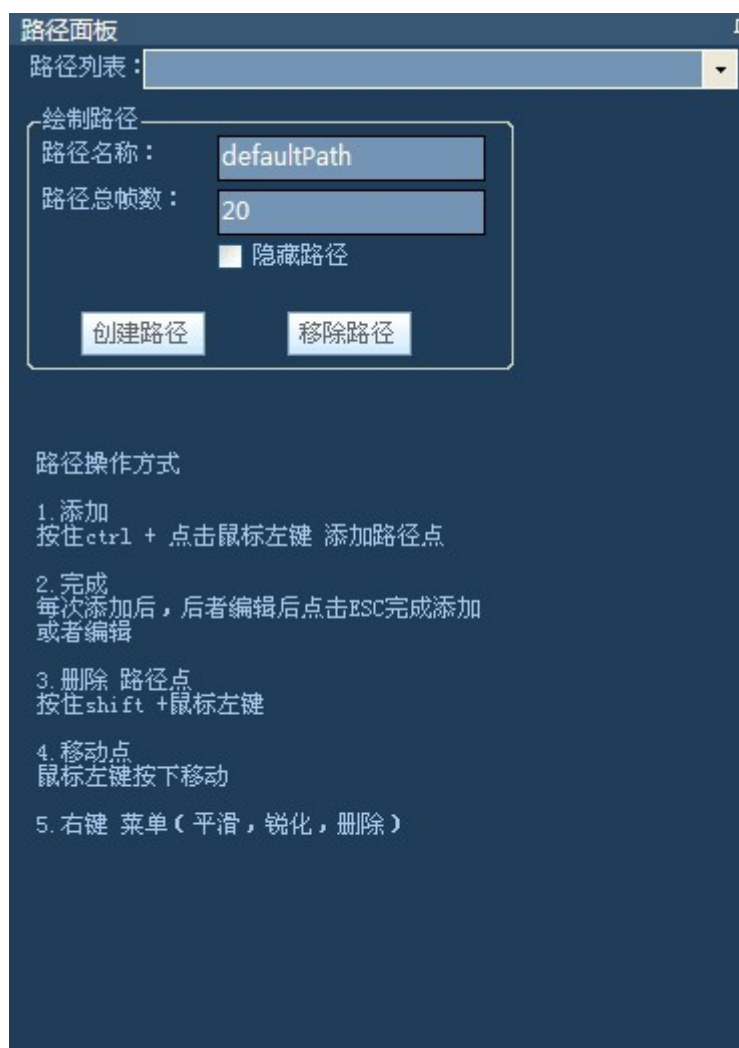


图 14

- **路径列表：**  
若创建了多个路径，通过该列表选择不同的路径。
- **路径名称：**  
设置路径的名称。
- **路径总帧数：**  
设置路径的总节点数。路径上的点分为两种，一种是明显可见的主要节点，一种是一个像素的浅色小节点，两种点的总和等于路径总帧数+1。路径中主要节点越多，路径的表现效果越多样。

## 区域面板

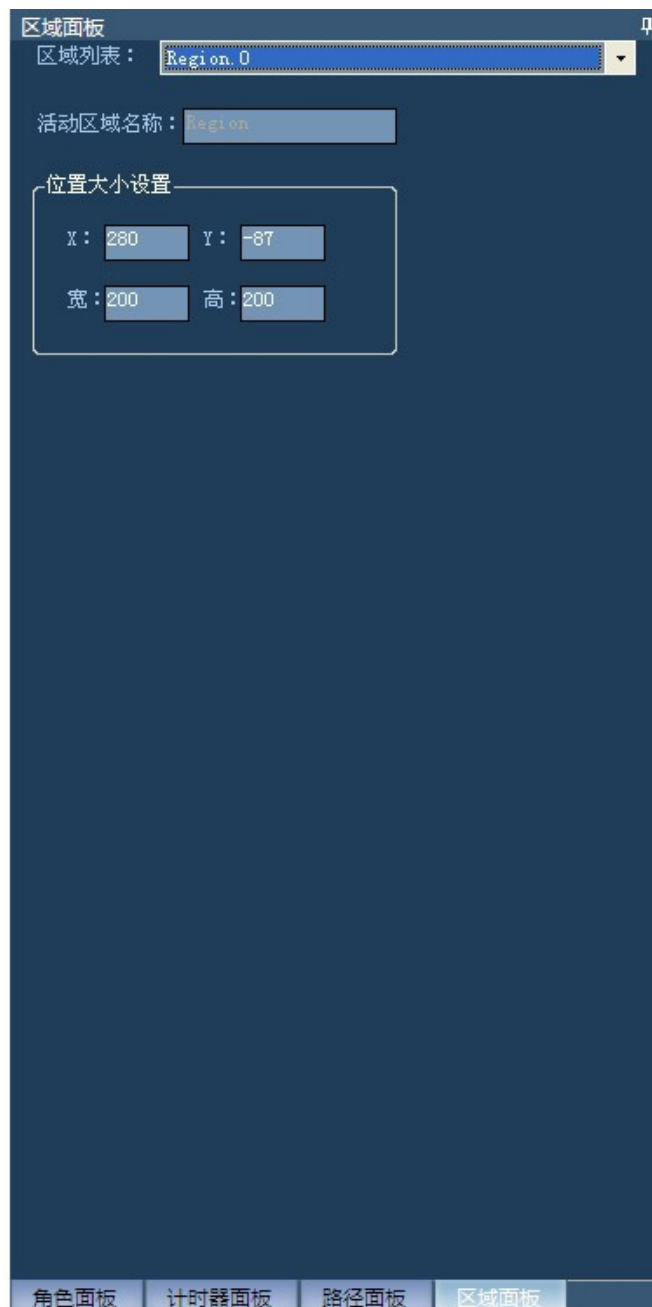


图 44

- **区域列表:**  
若有设置多个区域，则可以在该选项中选择。
- **活动区域名称:**  
设置活动区域的名称。
- **位置大小设置:**  
设置区域的 X、Y 轴的坐标，宽、高的大小。

已有角色



图 46

显示之前创建的所有角色。

事件

事件给了你定义角色事件发生的机会。比如，当用户点击角色或者特定的按键被按下。事件和行为有密切关系。一个事件可定义多个行为，并且这些行为之间以顺序的方式执行。

- MC 支持以下事件：

|       |       |      |      |      |
|-------|-------|------|------|------|
| 按下鼠标键 | 松开鼠标键 | 绘制角色 | 网络消息 | 网络错误 |
|-------|-------|------|------|------|

|        |      |        |        |           |
|--------|------|--------|--------|-----------|
| 动画完成   | 路径完成 | 键盘按键按下 | 键盘按键弹起 | 计时器 Timer |
| 碰撞     | 碰撞结束 | 创建角色   | 销毁角色   | 在可视范围外    |
| 鼠标进入   | 鼠标离开 | 移动结束   | 长按角色   | 多点触控      |
| 物理物体接触 | 单击触屏 | 异步调用结束 | 滑动     | 通知事件      |

- 按下鼠标键：

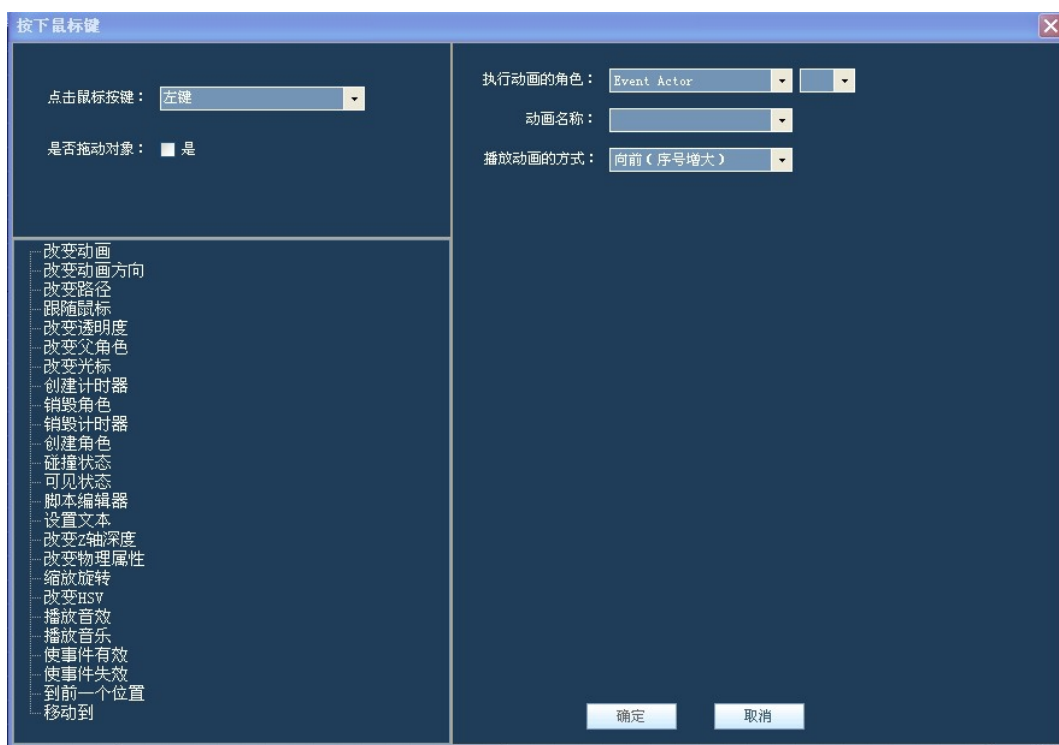


图 47

指定的鼠标按键按下后产生

· 是否拖动角色：

当角色可以被拖动时选择“是”

- 松开鼠标键：



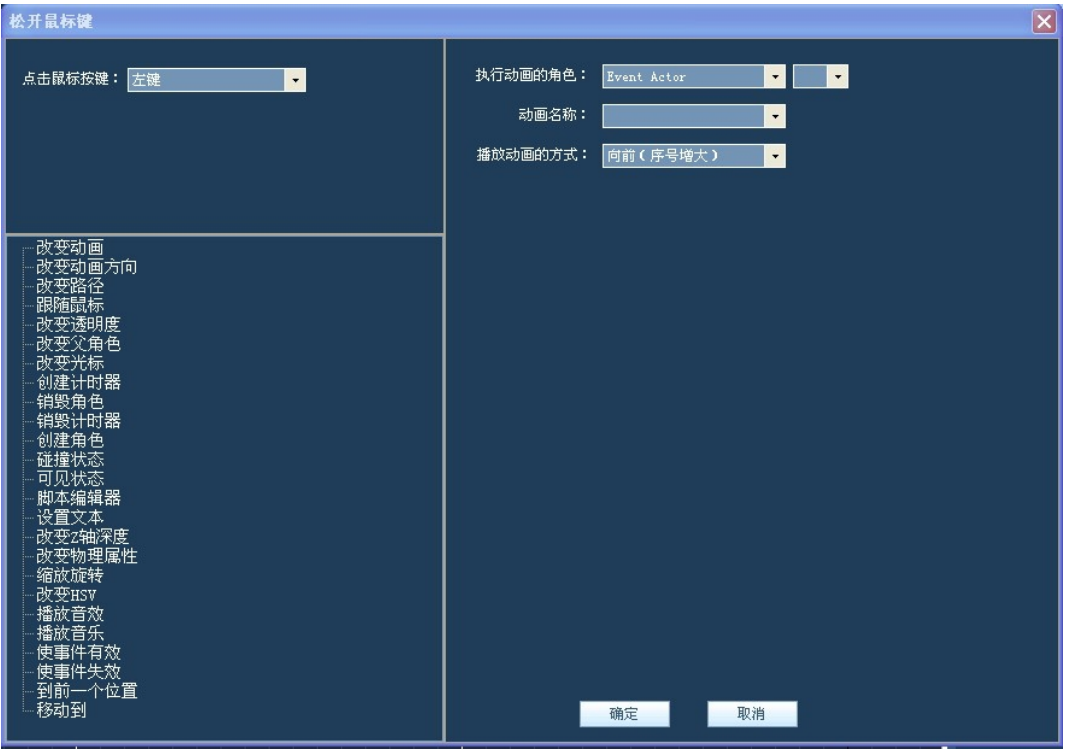


图 48

鼠标放开后产生

- 绘制角色:

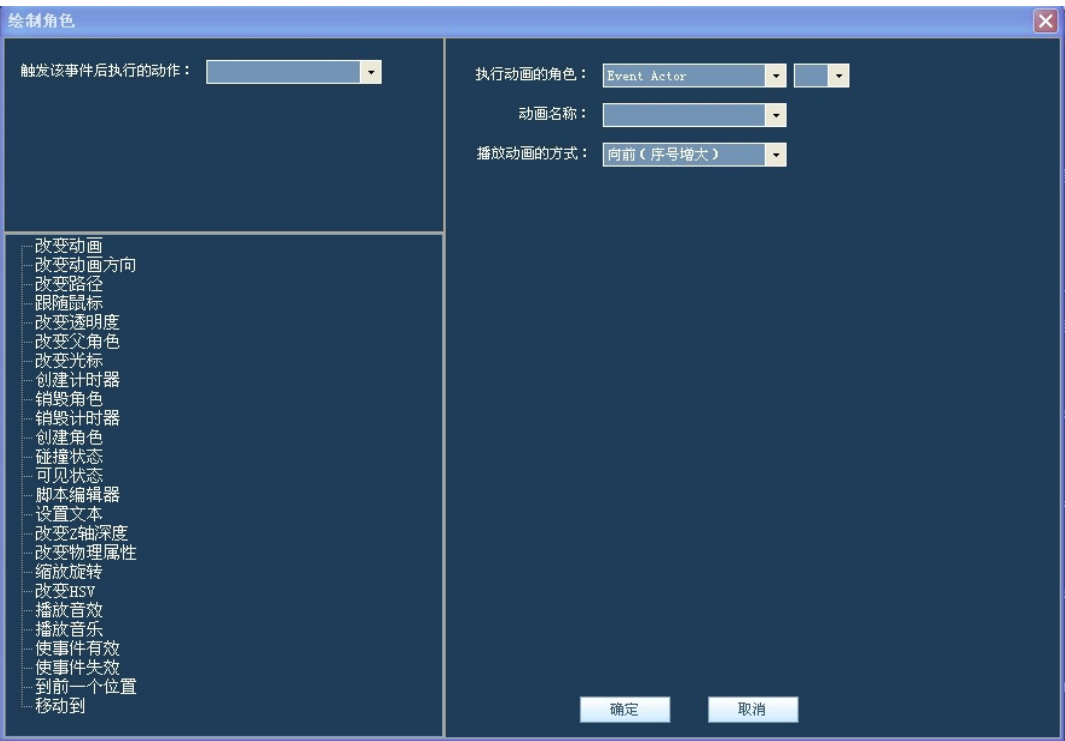


图 49

产生于指定游戏帧率。在此事件里的行为将会在每一个游戏帧被执行。

- 网络消息:

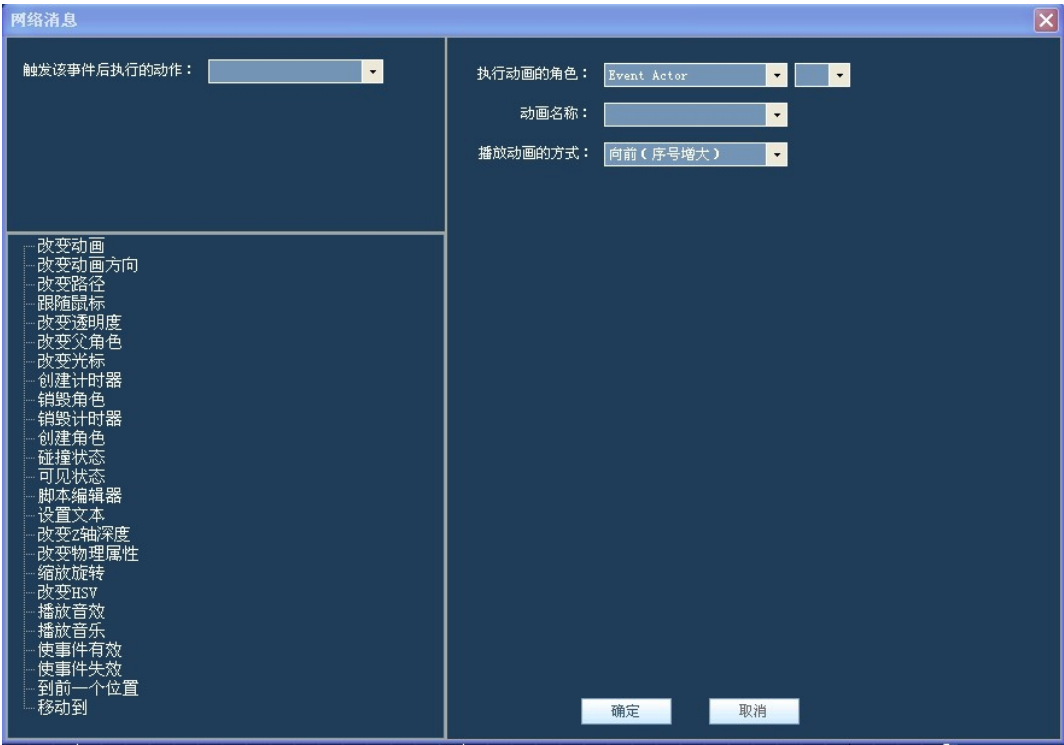


图 50

当接收到网络来源的信息时，执行相应的行为。

- 网络错误:

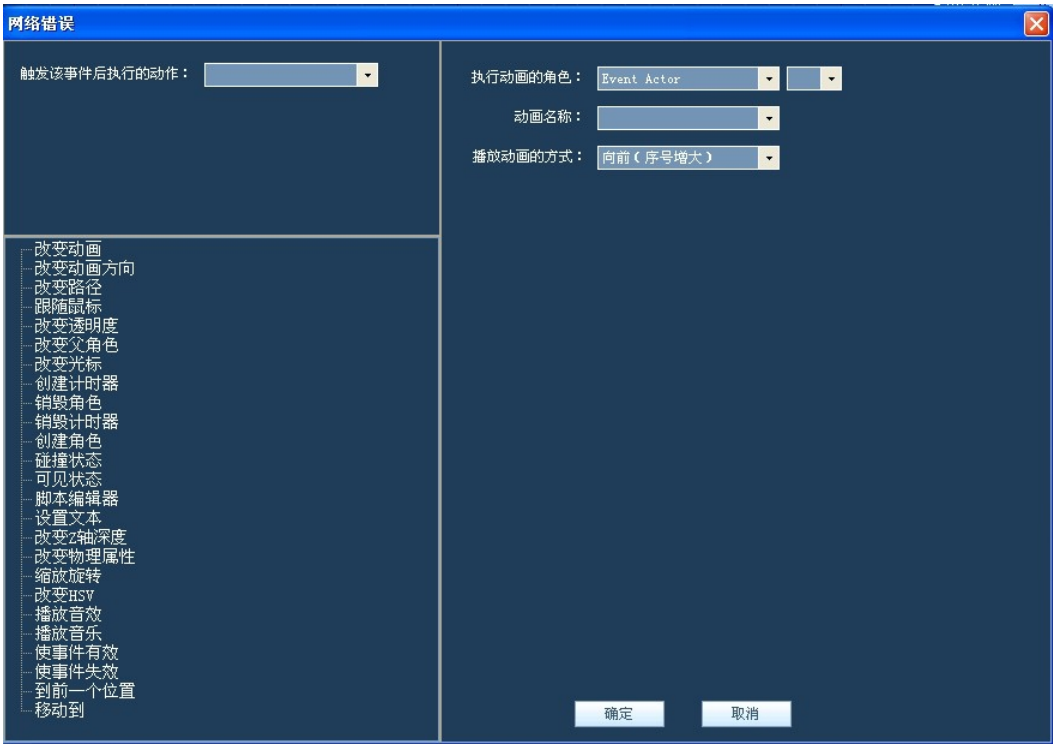


图 51

当网络上发生错误时，执行相应的行为。

- 动画完成:

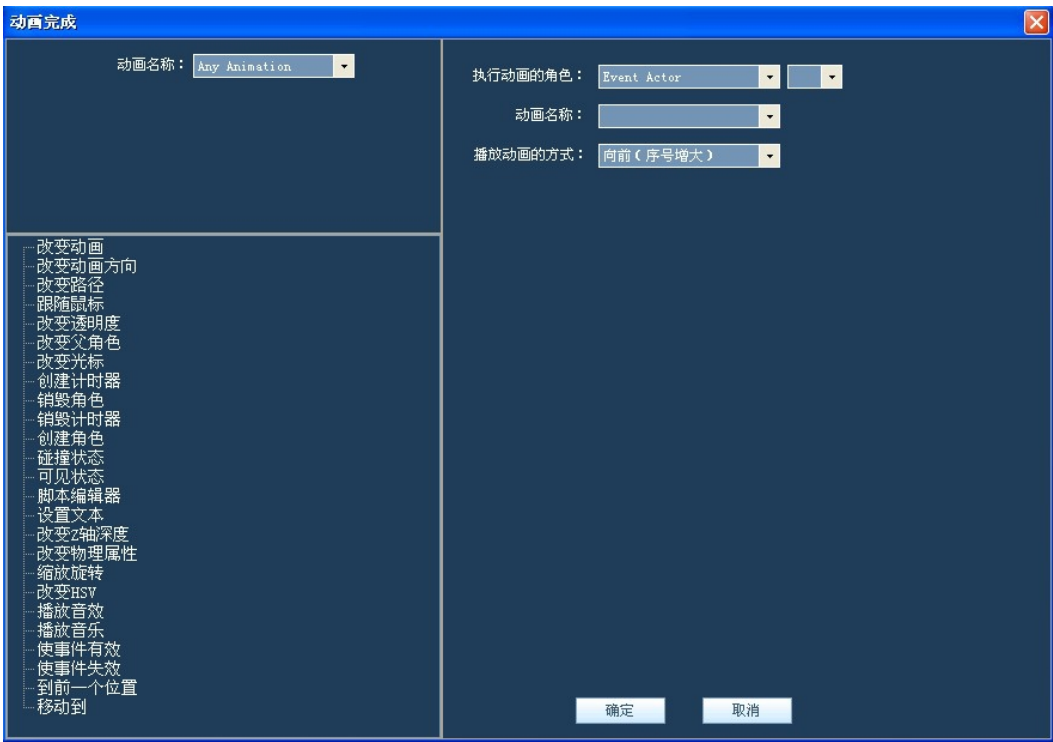


图 52

当动画结束时产生相应的行为

- 路径完成:

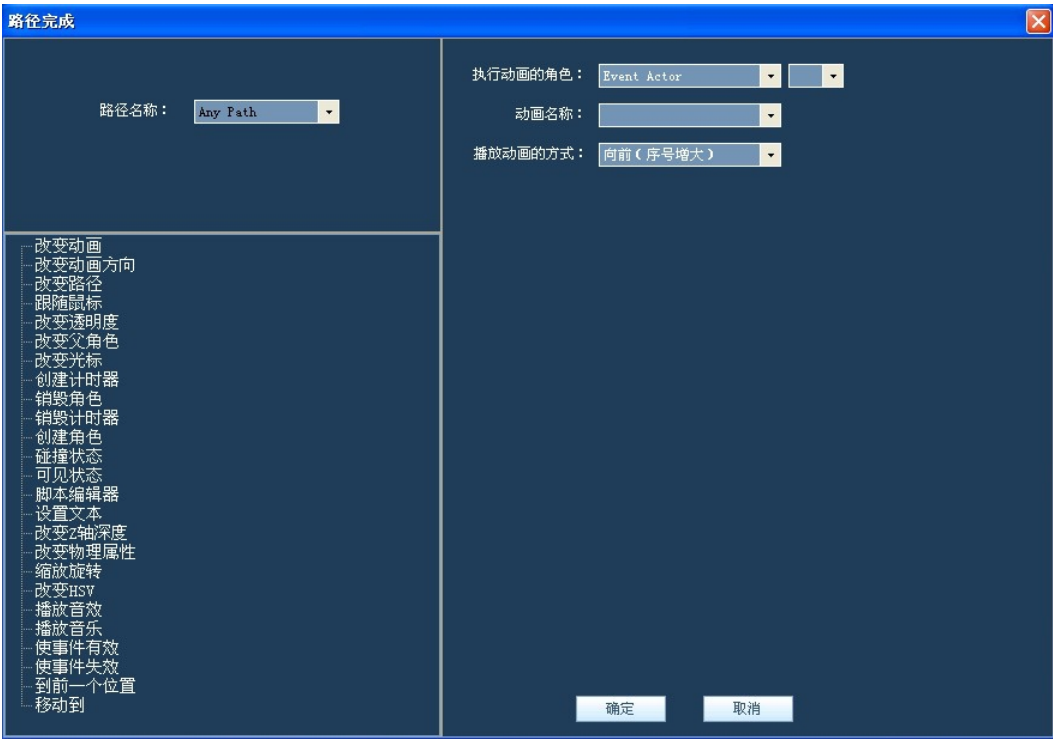


图 53

当选择的路径完成时发生相应的行为

- 键盘按键按下:

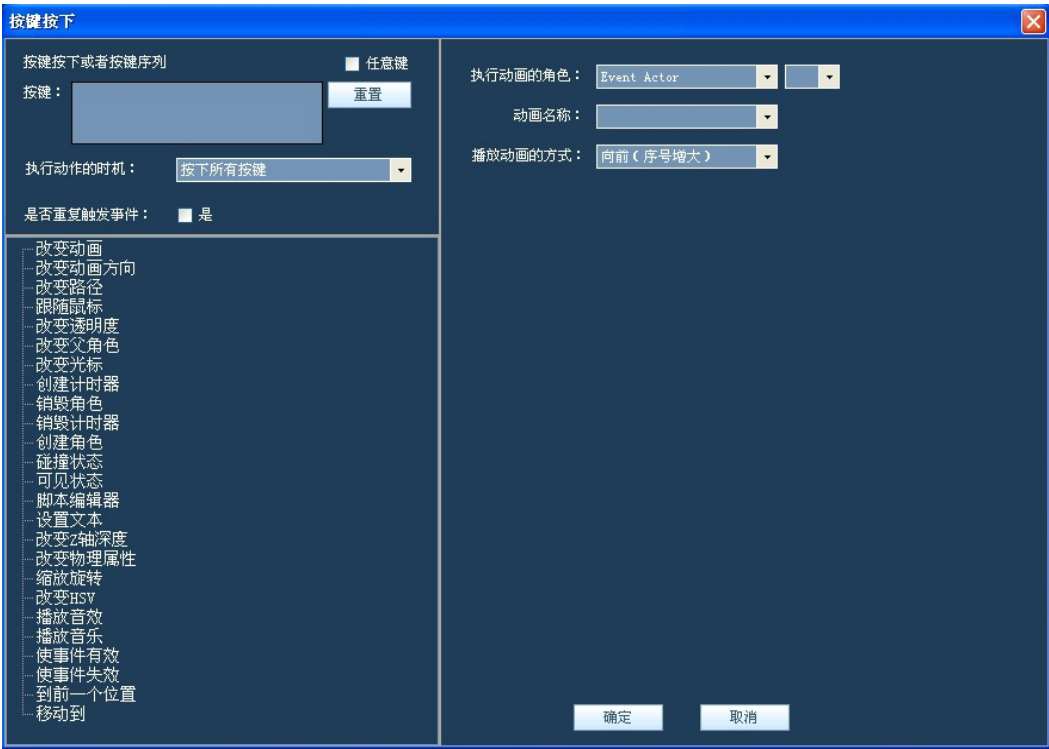


图 54

- 指定按键或者按键序列被按下时产生。有多达 12 个按键能指定。
- **重置：**  
点击以删除所有指定的按键。
  - **执行动作的时机：**  
选择“至少按下一个按键”，至少一个指定的按键按下时执行行为。  
选择“按下所有按键”，当所有指定按键被按下时执行行为。  
选择“按顺序按键”当按键按照顺序按下时执行行为。你可以用此选项为你的游戏制作秘籍代码。如果只指定一个按键，此选项无效。
  - **是否重复触发事件：**  
当选择为“是”时, 则按键被持续按下，事件也会被持续发送。
  - **任意键：**  
勾选后按键框中显示“any key”，表示当任何按键按下时都会接收此事件。
  - **键盘按键弹起：**

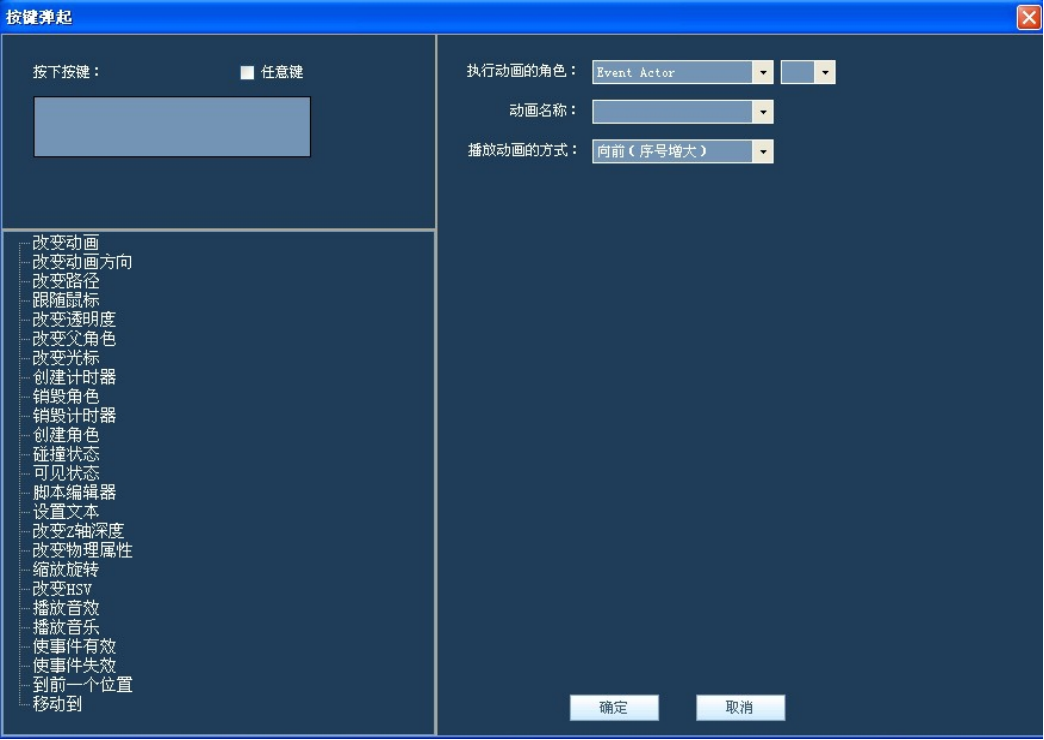


图 55

指定按键放开后产生

- 任意键：

勾选后按键框中显示 “any key”，表示当任何按键放开时都会接收此事件。

- 计时器 Timer：

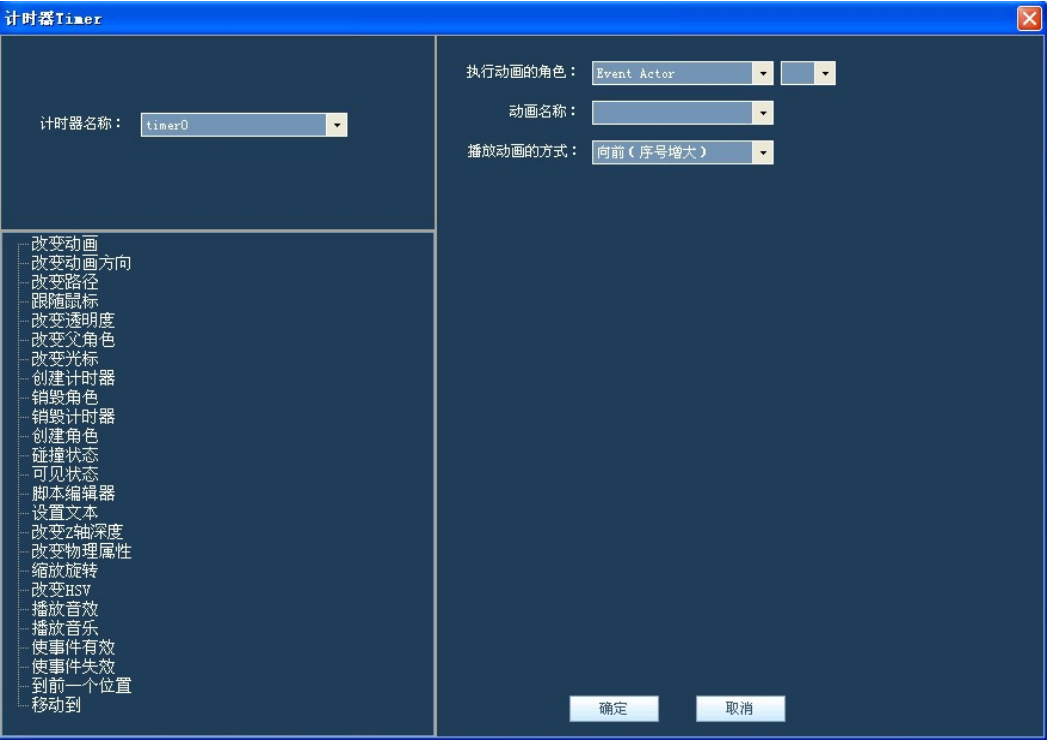


图 56

当计时器条件达到的时候发生。在之前必须在行为里有创建计时器 Timer。

- 碰撞：

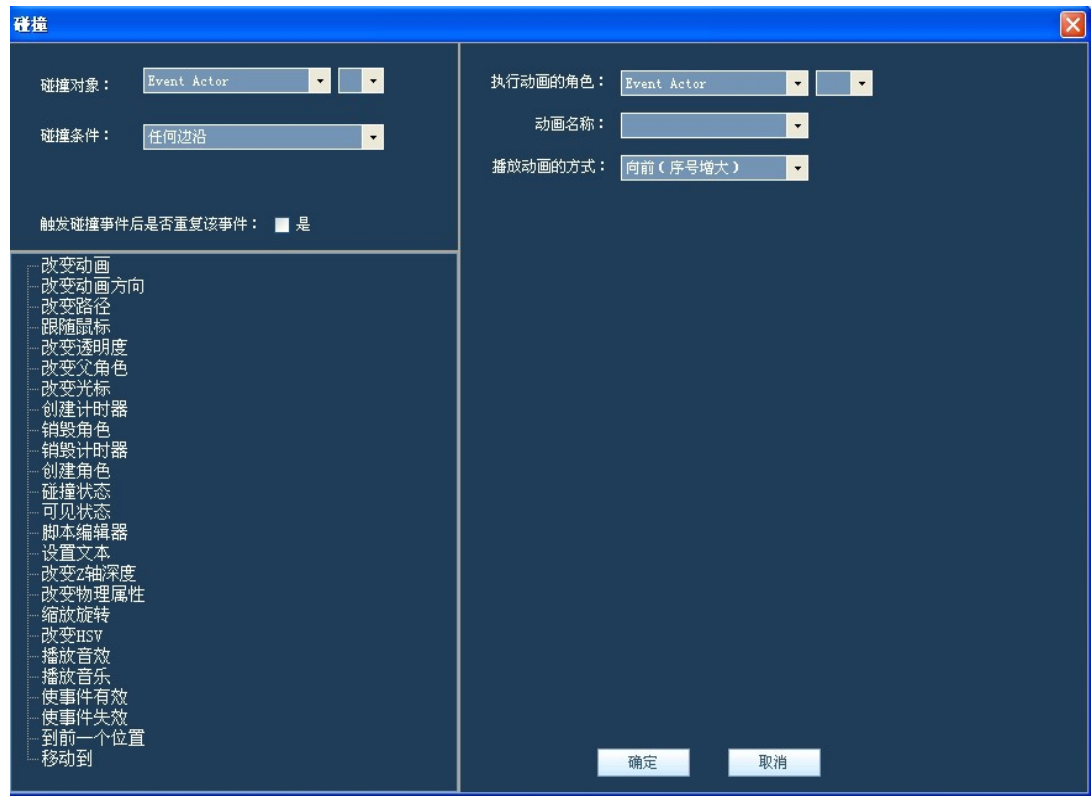


图 57

当指定的角色在选中的边界(任意边，上，下，左，右，上或下，左或右)发生碰撞时产生。

在传统游戏里，当玩家与怪物的上边发生碰撞时，“物体碰撞”事件删除怪物，当玩家与怪物的左边或右边发生碰撞时，“物体碰撞”删除玩家。

• 触发碰撞事件后是否重复该事件：

选择“是”当角色发生碰撞时持续发送碰撞事件，若不选择, 碰撞事件只在碰撞开始的时候发送。

- 碰撞结束：

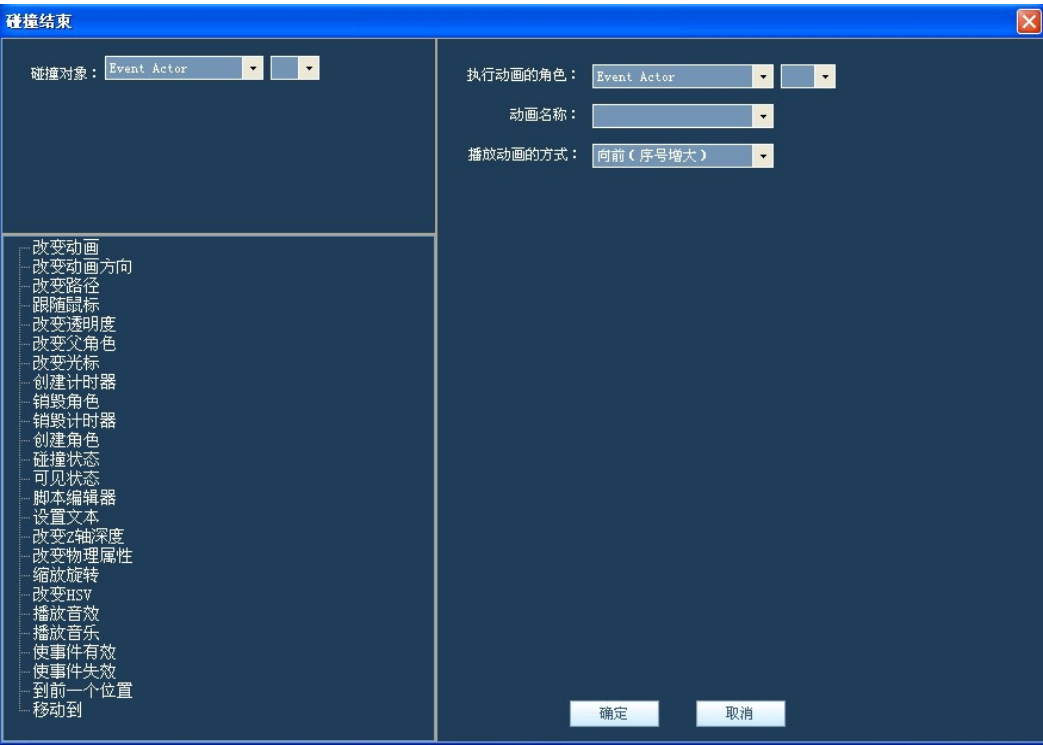


图 58

当和一个选择的角色碰撞结束时产生。

- 创建角色：

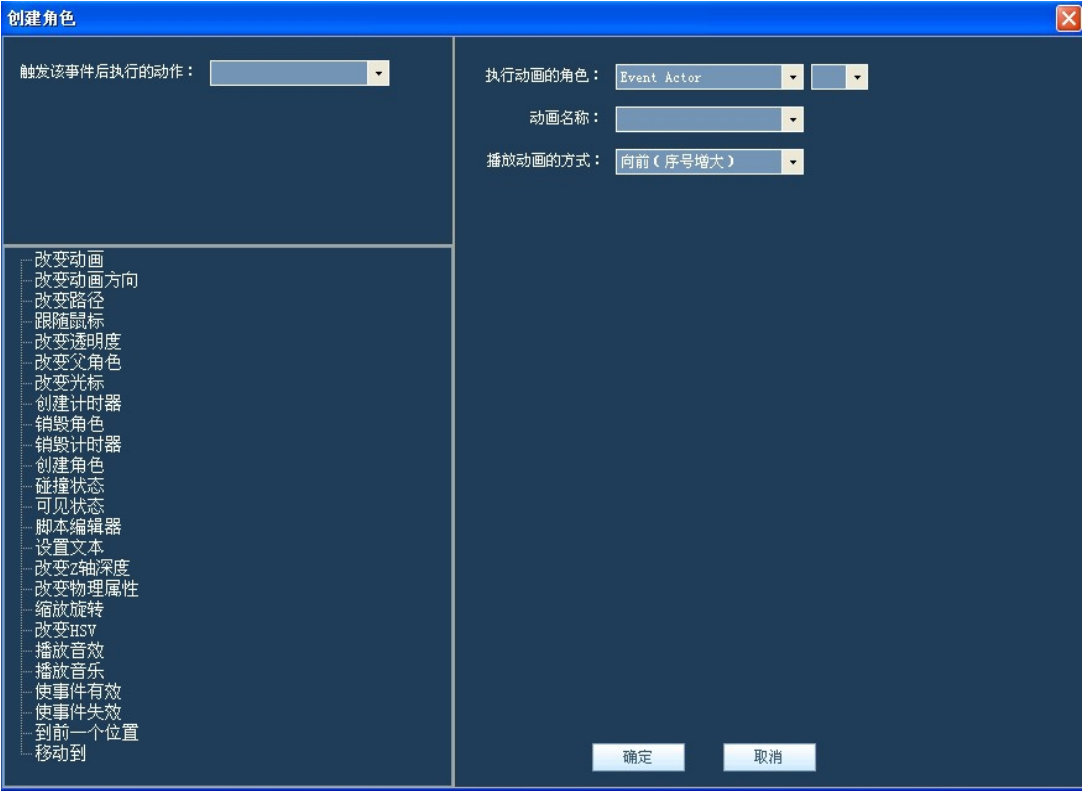


图 59

当前角色被创建时产生。循环进行角色间的创建事件和创建行为，容易导致错误。例如角

色 A 的创建事件发生的时候，进行创建角色 B 的行为，然后 B 创建事件发生的时候又创建 A……

- 销毁角色：

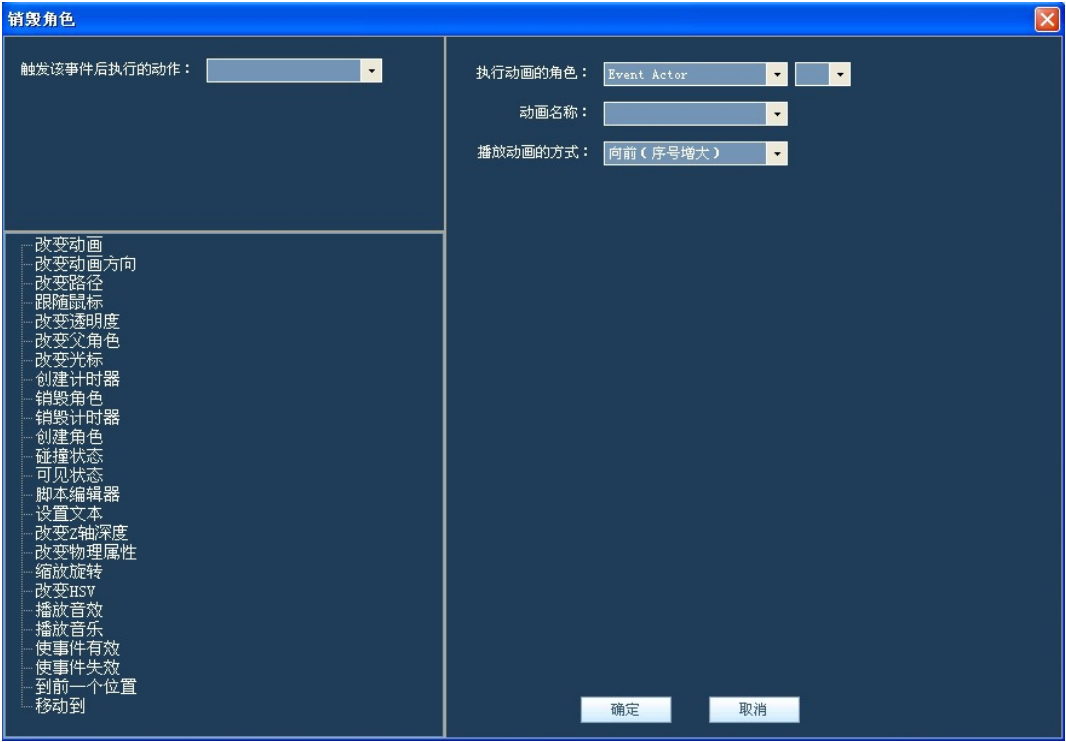


图 60

当前角色被删除后产生。

- 在可视范围外：

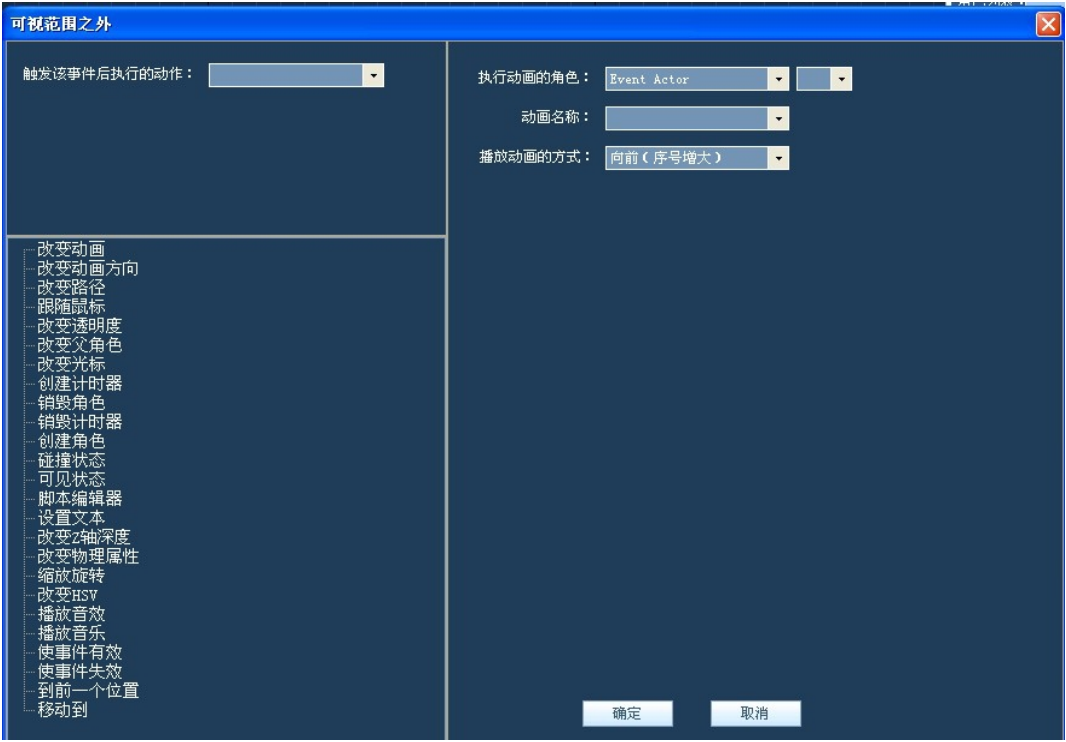


图 61



当角色离开视野时发生(当前 view 的尺寸加上游戏设置中安全界限里设置的大小)。

- 鼠标进入：

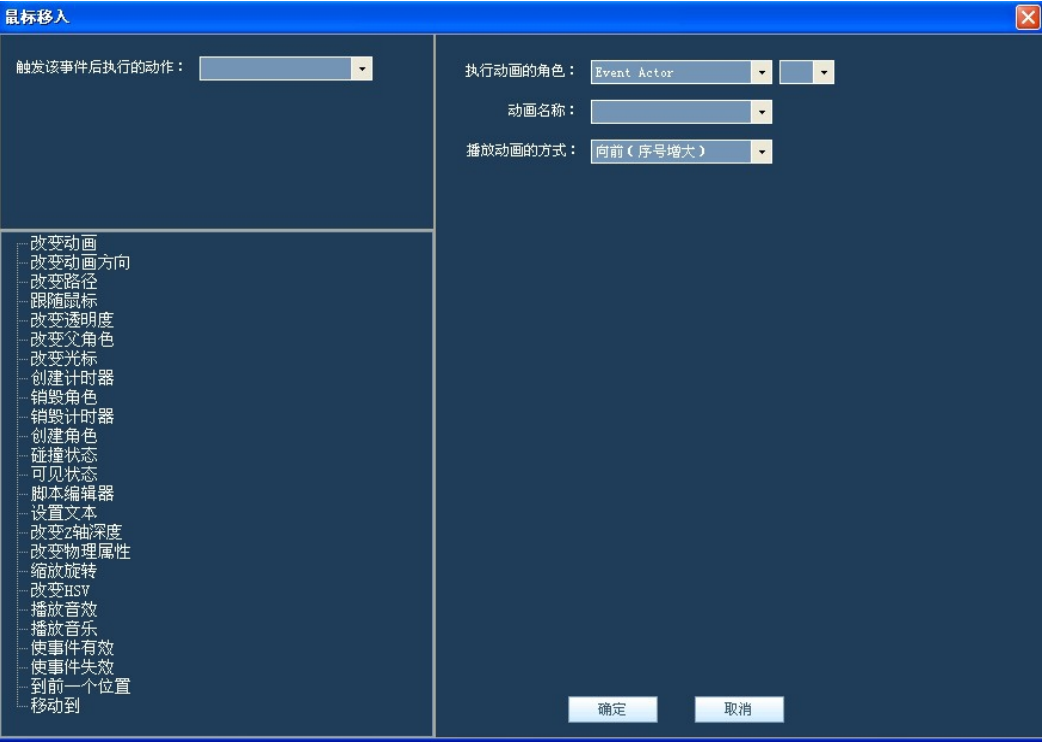


图 62

鼠标移进角色后产生

- 鼠标离开：

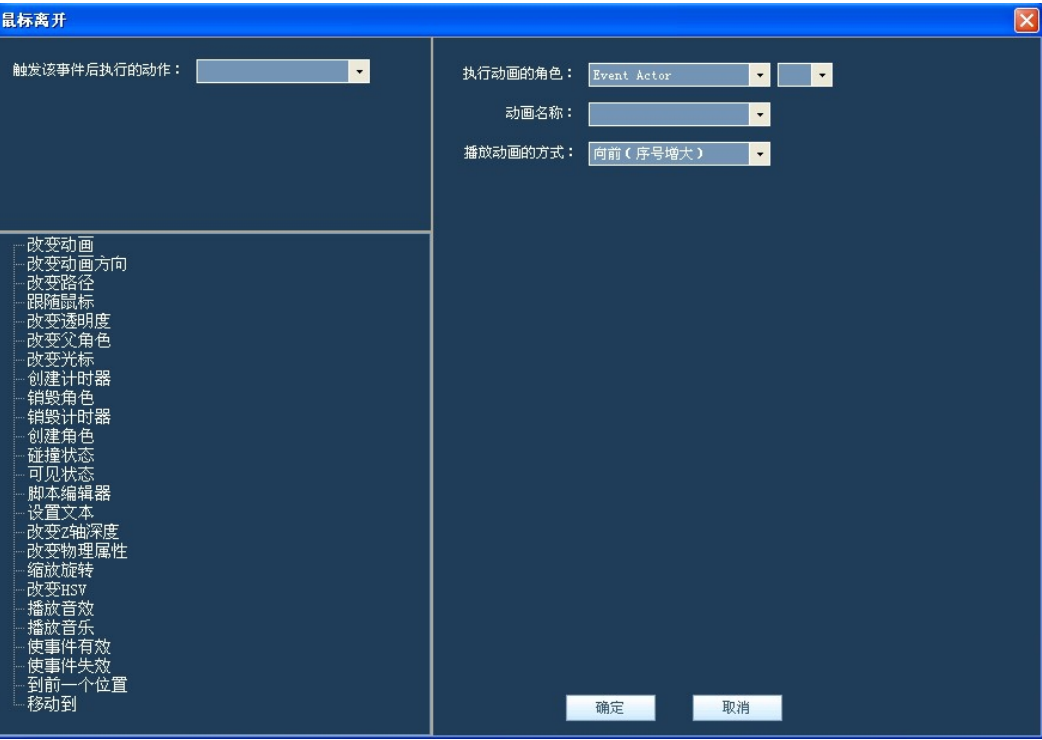


图 63

鼠标离开角色后产生

- 移动结束:

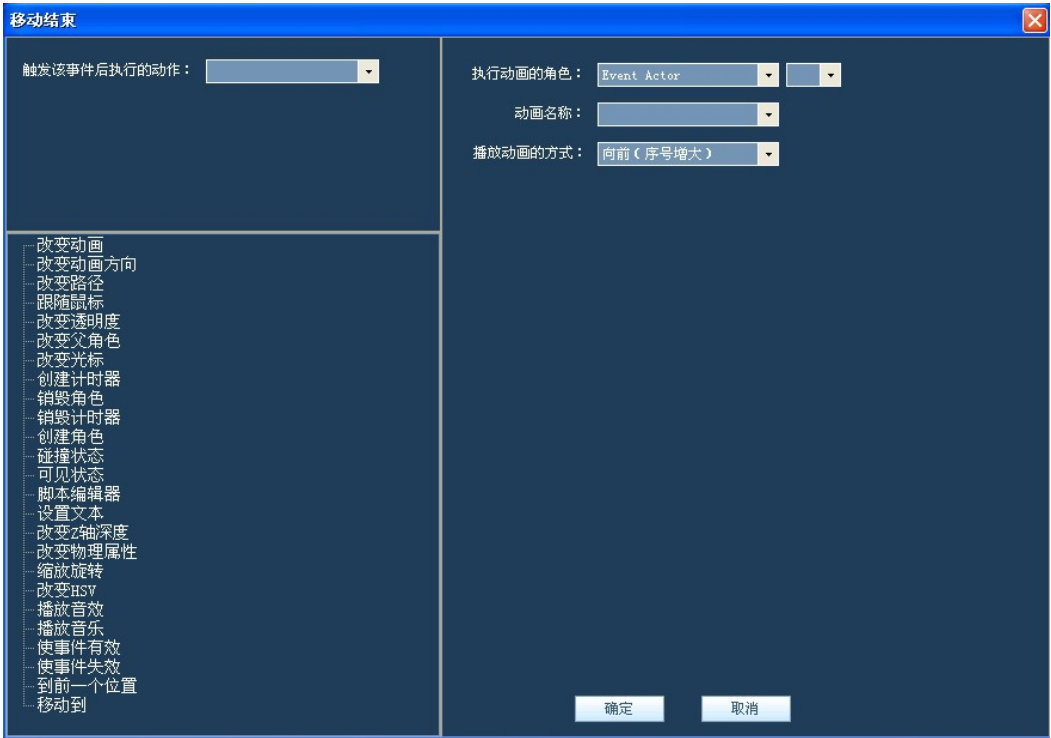


图 64

当使用“移动到”行为时，到达指定坐标后发生。

- 长按角色:

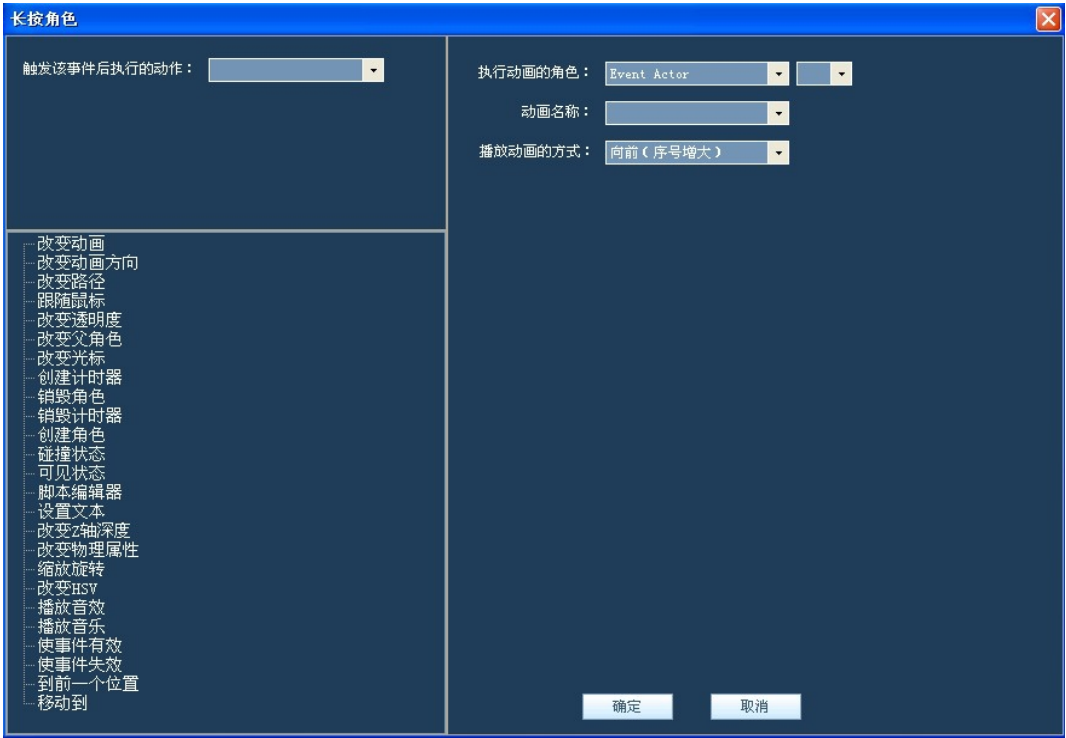


图 65

当鼠标键一直按下不放，或触屏时手按住某点不放开时，可以使用该事件执行相应行为。

- 多点触控：

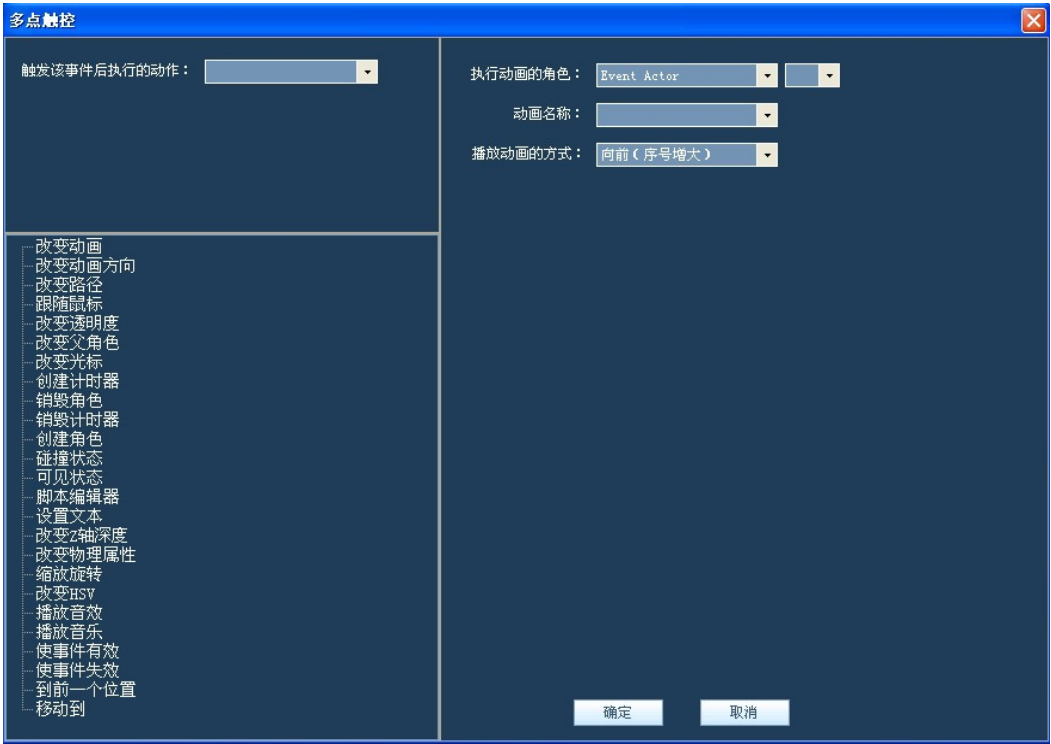


图 66

触屏中可以接受多个手指同时对屏幕进行操作的事件。

- 物理物体接触：

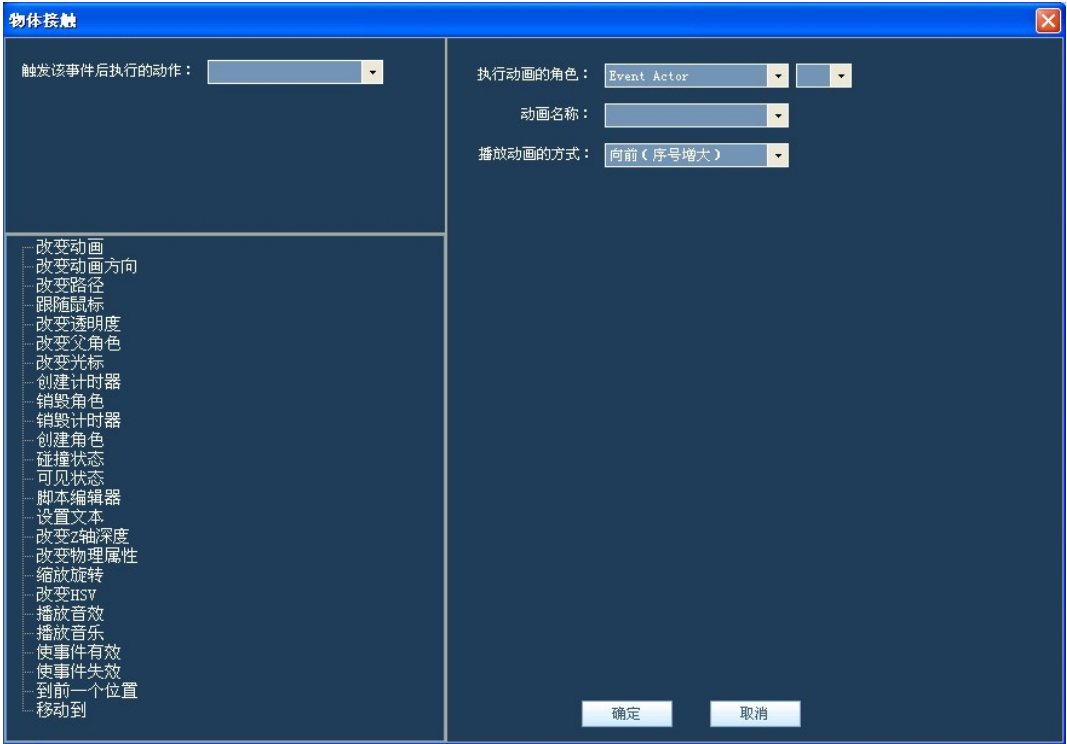
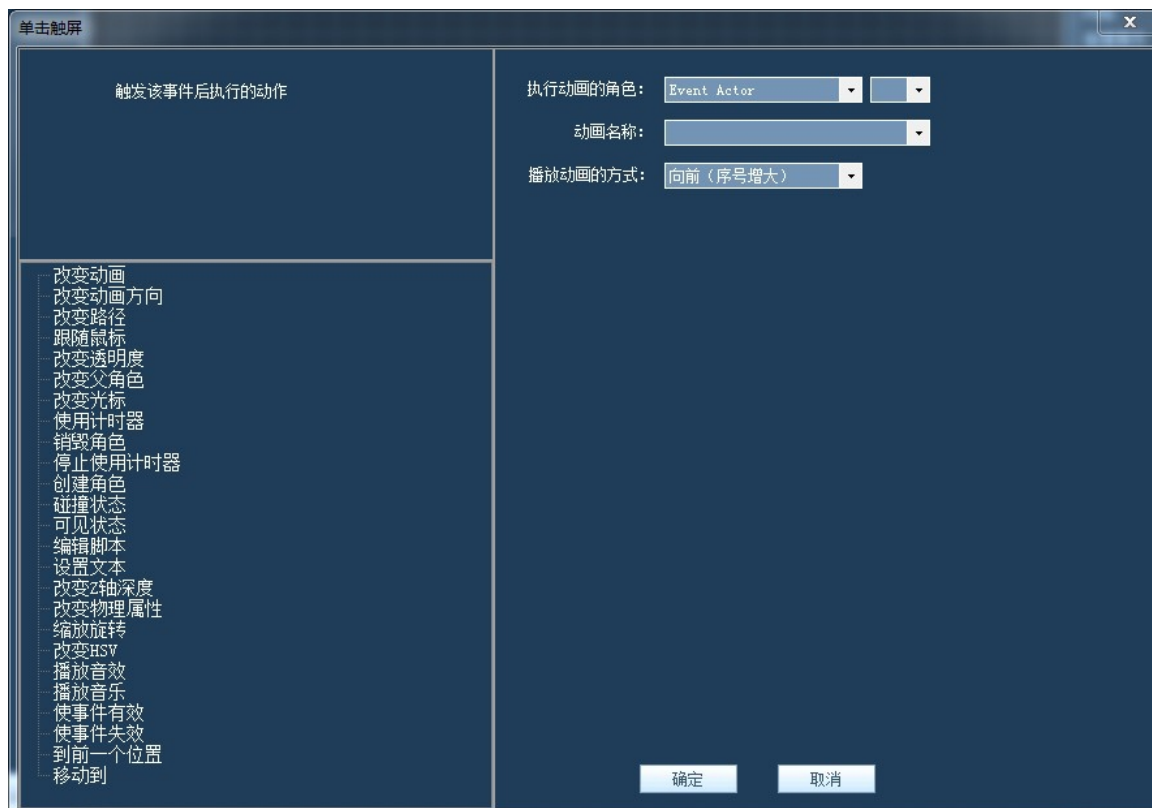


图 67

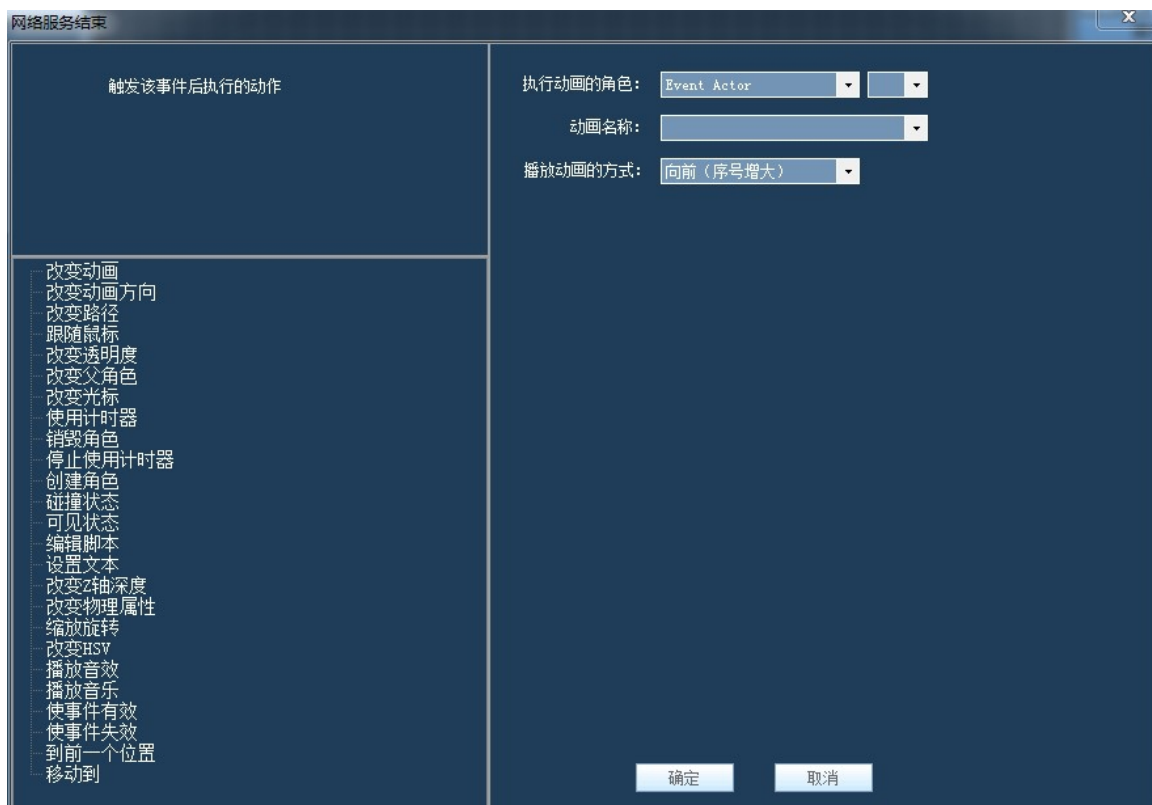
当需要模拟物理世界的效果时，可以使用该事件来执行相应的行为。

- 单击触屏：



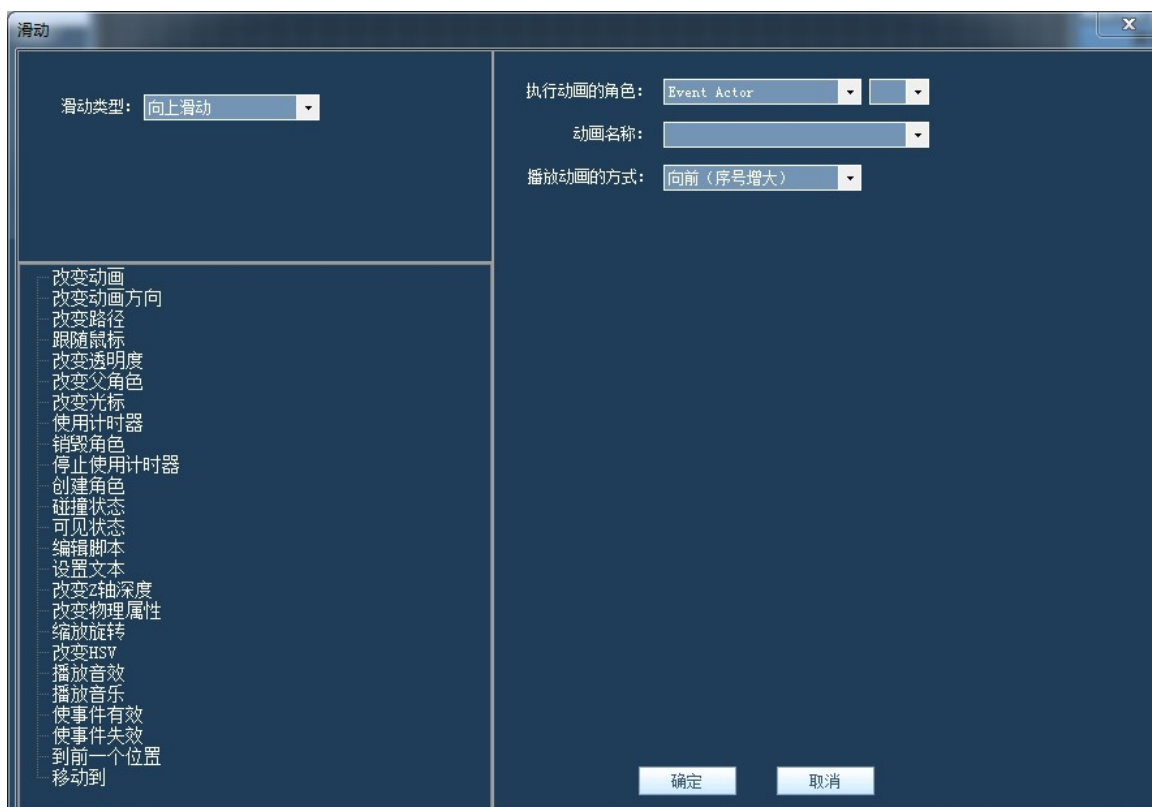
当角色被点击的时候触发。

- 异步调用结束：



当异步调用结束的时候（异步服务结束时）触发。

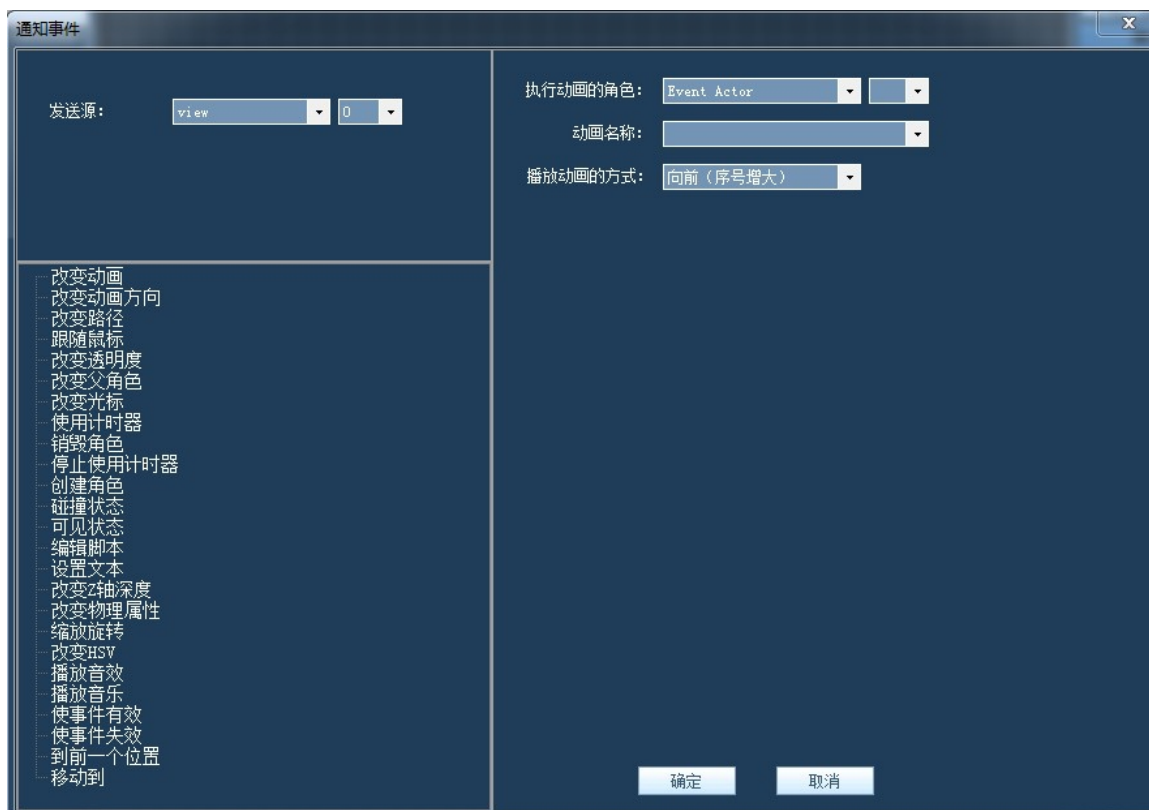
- 滑动:



当手指在屏幕上滑动的时候触发。

- **滑动类型：** 向上滑动、向下滑动、向左滑动、向右滑动

- **通知事件：**



接收通知事件。

- **发送源：** 发送触发信号的角色。

即触发发送源角色某个事件时，执行当前角色指定的行为。

## 行为

行为就是事件响应时应该发生什么。比如，当一个角色和火箭碰撞时就应该创建一个爆炸。

MC 有一套预定义的行为并且任何事件都能触发一个或多个行为。

行为在事件之后。如果在一个角色上定义鼠标按下的事件，行为在鼠标事件发生后产生。

- **MC 支持下列行为：**

|          |        |        |        |         |
|----------|--------|--------|--------|---------|
| 改变动画     | 改变动画方向 | 改变路径   | 跟随鼠标   | 改变透明度   |
| 改变父角色    | 改变光标   | 使用计时器  | 销毁角色   | 停止使用计时器 |
| 创建角色     | 碰撞状态   | 可见状态   | 脚本编辑器  | 设置文本    |
| 改变 Z 轴深度 | 改变物理属性 | 缩放旋转   | 改变 HSV | 播放音效    |
| 播放音乐     | 使事件有效  | 使事件失败效 | 到前一个位置 | 移动到     |

- **改变动画：**

改变当前角色的动画。

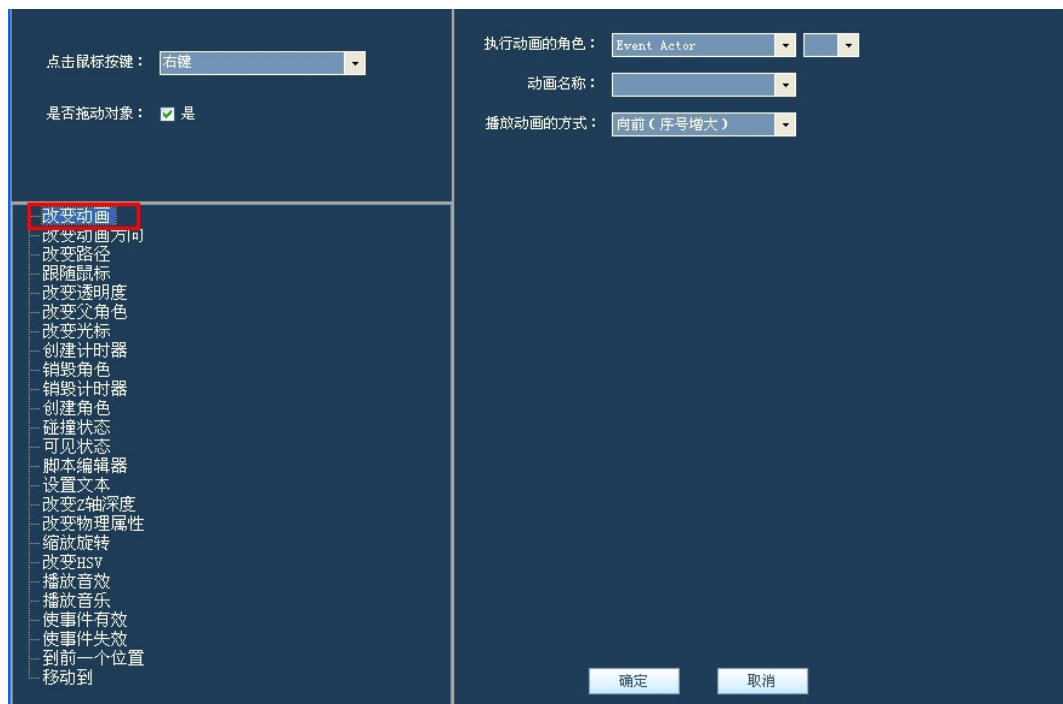


图 68

- **执行动画的角色：**

选择要改变的角色。

- **动画名称：**

选择角色中的某组动画。

- **播放动画的方式：**

指定动画的播放顺序分为向前、向后、停止、无变化模式，此行为会改变 animindex 变量，不能改变碰撞角色的动画。

- **改变动画方向：**

改变当前动画的播放顺序。

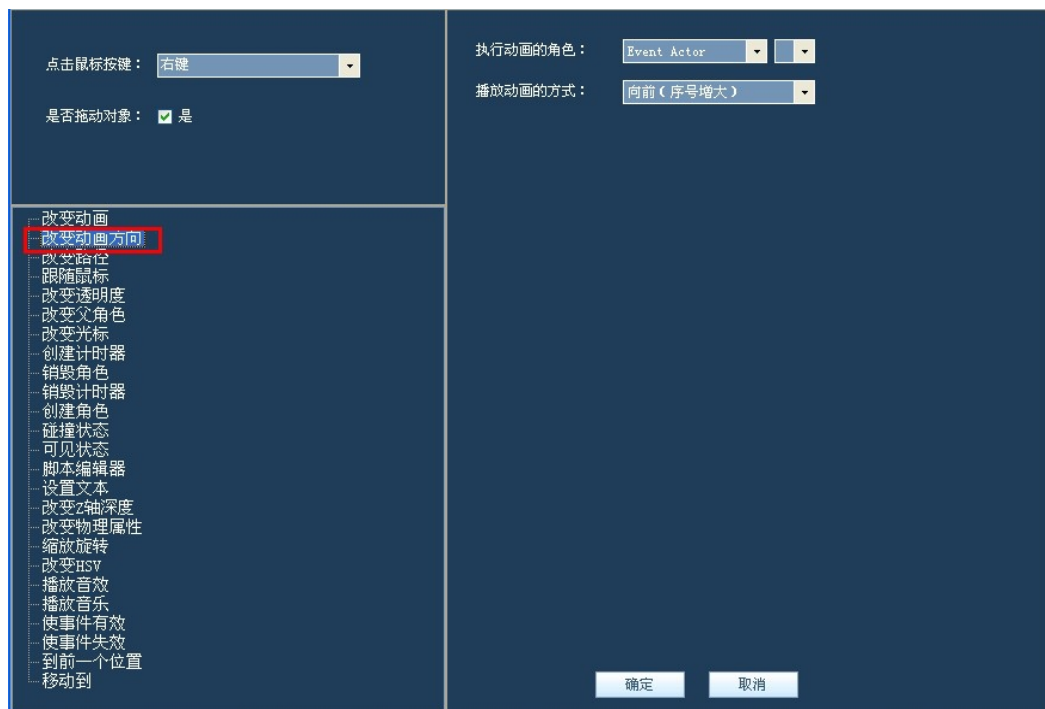


图 69

- **执行动画的角色：**

选择需要改变动画的角色。

- **播放动画的方式：**

指定动画的播放顺序分为向前、向后、停止、无变化模式。

- **改变路径：**

改变选择角色的路径。





图 70

- **角色名称：**  
选择要改变路径的角色。
  - **路径名称：**  
选择路径 (选项包括：之前创建的任意一个路径，无路径和随机路径)。
  - **影响的坐标轴：**  
指定该路径是否和 X 轴、Y 轴有关或都有关。
- **跟随鼠标：**  
让角色跟随鼠标移动。

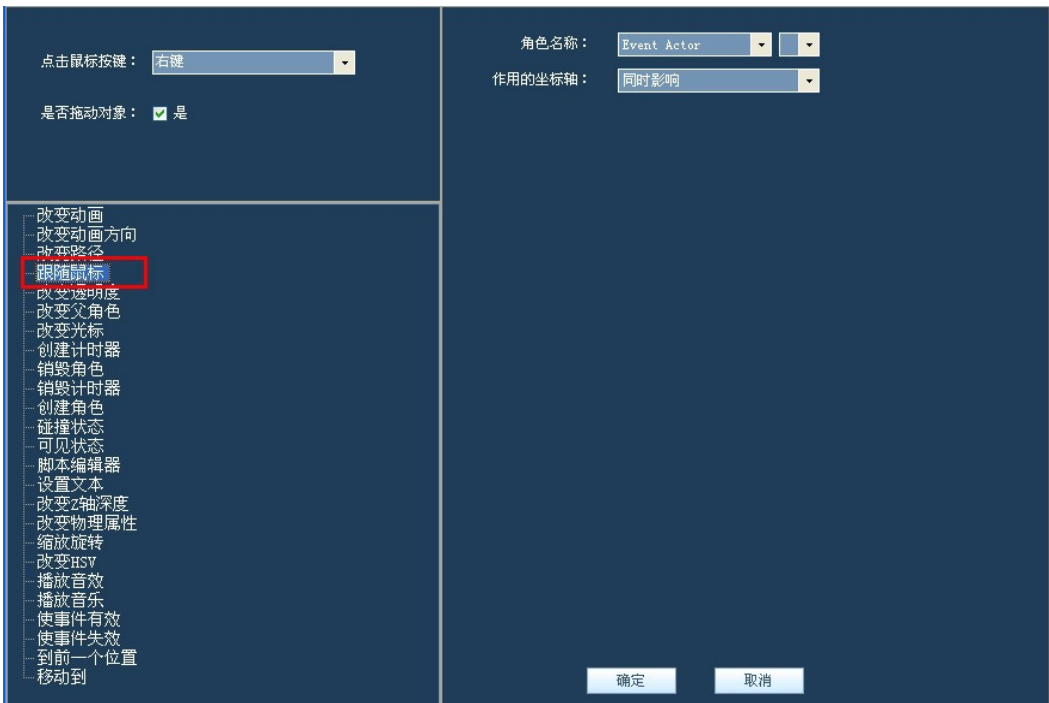


图 71

- **角色名称：**  
选择要跟随鼠标的角色。
  - **作用的坐标轴：**  
指定改路径跟随的范围 X 轴有效，Y 轴有效，都有效或都无效 (结束跟随)。
- **改变透明度：**  
改变选择角色的透明度。

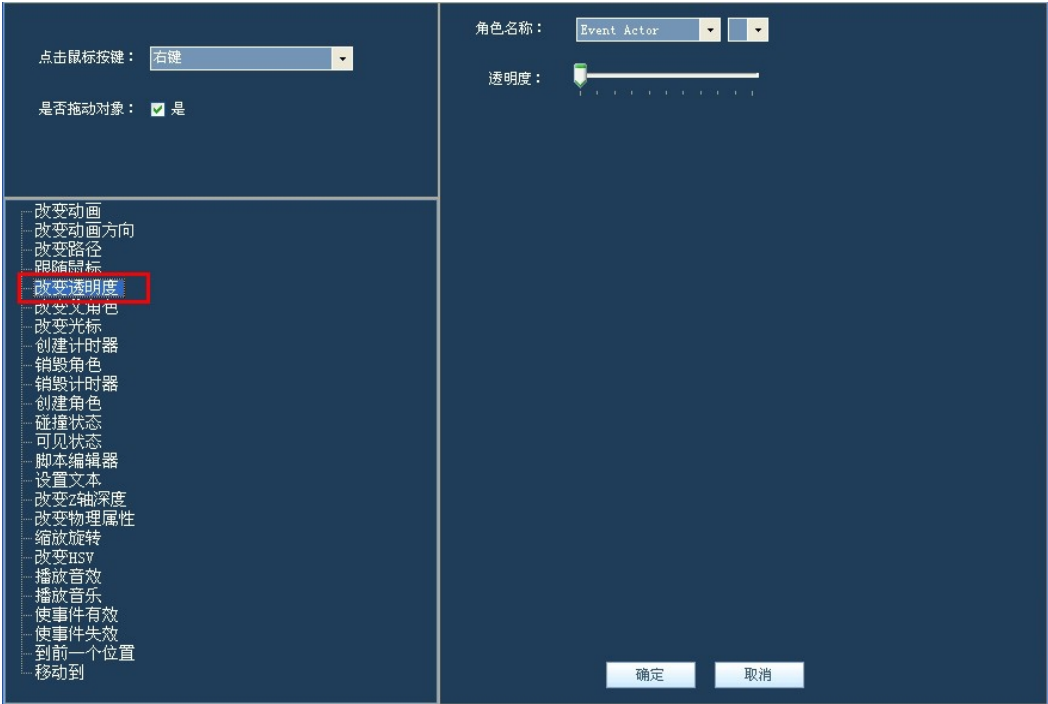


图 72

- **角色名称：**  
选择要改变透明度的角色。
- **透明度：**  
使用滑动条设置透明度，越往右越透明。
- **改变父角色：**  
改变选择的角色的父角色。

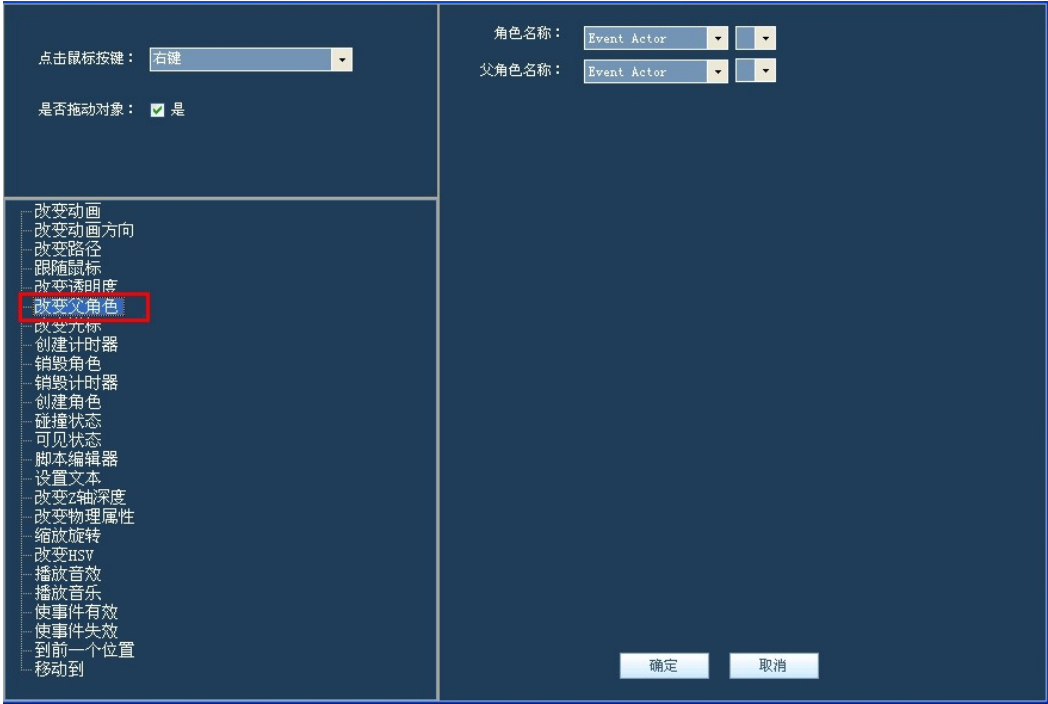


图 73

- **角色名称:**  
选择子类角色。
- **父角色名称:**  
选择父角色角色。
- **改变光标:**  
改变当前角色的鼠标光标。

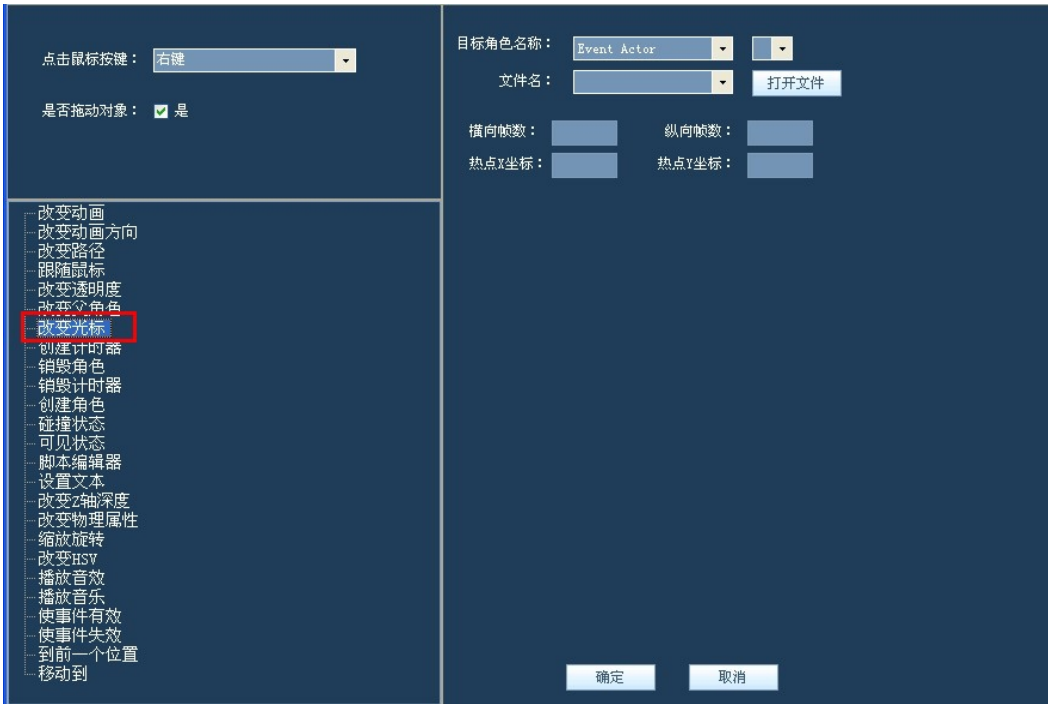


图 74

- **目标角色名称:**  
选择待改变光标的角色。
- **文件名:**  
选择光标图像文件。
- **横向帧数/纵向帧数:**  
分割光标图像文件。
- **热点 X/Y 坐标:**  
该光标响应鼠标事件的热点位置。
- **使用计时器:**  
为选择的角色使用一个计时器(timer)。

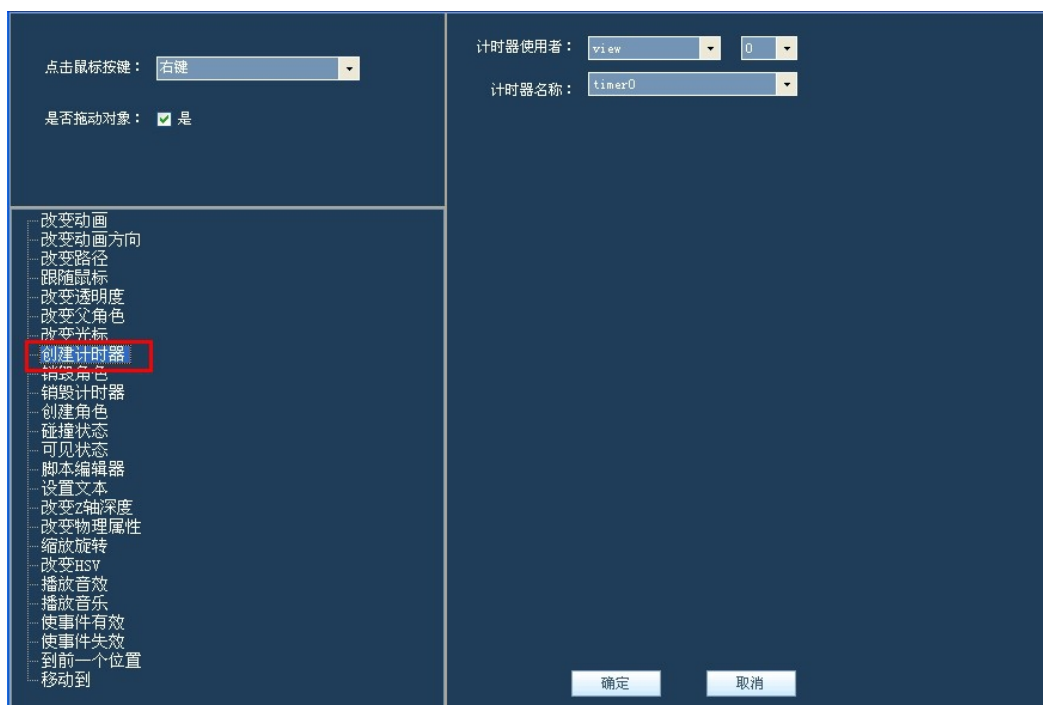


图 75

- **计时器的使用者：**

选择一个角色作为该计时器的使用者，可以是发生事件的当前角色，也可以是 view 或者别的指定角色。

- **计时器名称：**

选择一个已存在的计时器。

- **销毁角色：**

删除选择的角色。

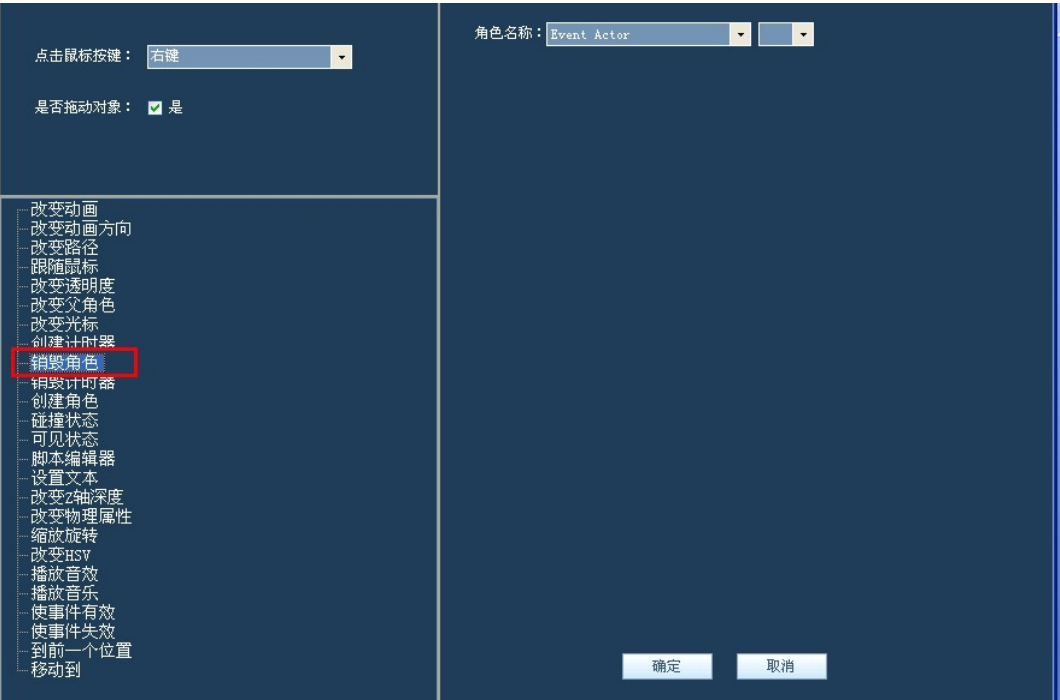


图 76

- **角色名称:**  
选择要删除的角色。
- **停止使用计时器:**  
停止使用的计时器，从而不再使用该计时器。该行为必须在角色有计时器的前提下。

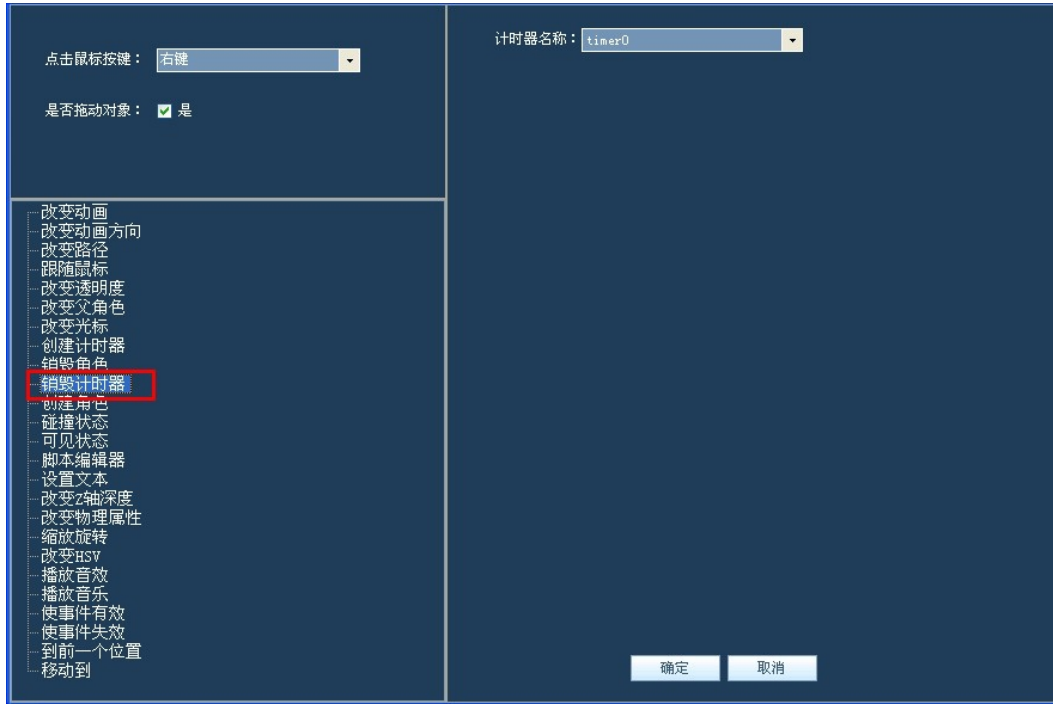


图 77

- **计时器名称:**

选择要停止使用的计时器。

- **创建角色：**

创建一个新角色。



图 78

- **被创建的角色：**

选择一个已设置好的角色进行创建。

- **动画名称：**

选择待创建角色的动画名。

- **父角色：**

选择该角色的父角色(选项包括已创建的角色，view 角色和无父角色)。

- **路径：**

选择一个路径(选项包括:已创建的任何路径，无路径和随机路径)。

- **初始化位置：**

设置角色的初始出现的坐标。

- **相对坐标的创建：**

设置待创建的角色是否和当前角色的坐标相关联。

- **碰撞状态：**

设置角色的碰撞状态为激活或者无效。

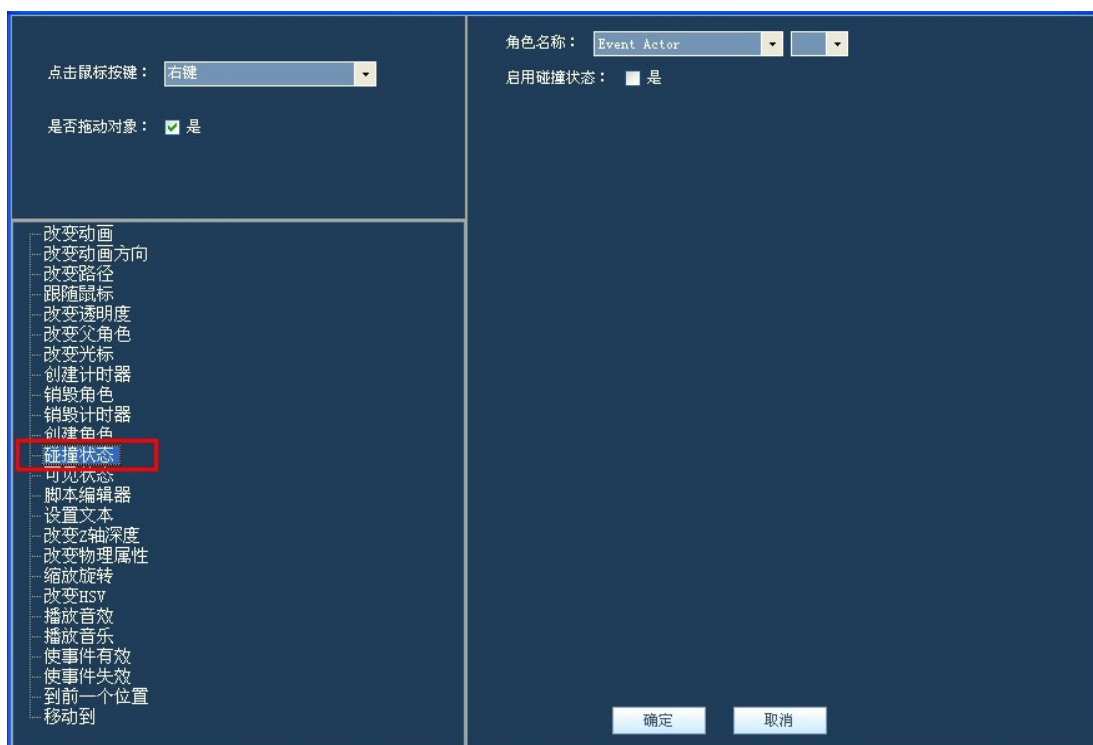


图 79

- 角色名称:

选择待设置碰撞状态的角色。

- 启动碰撞状态:

让碰撞状态激活或无效。

- 可见状态:

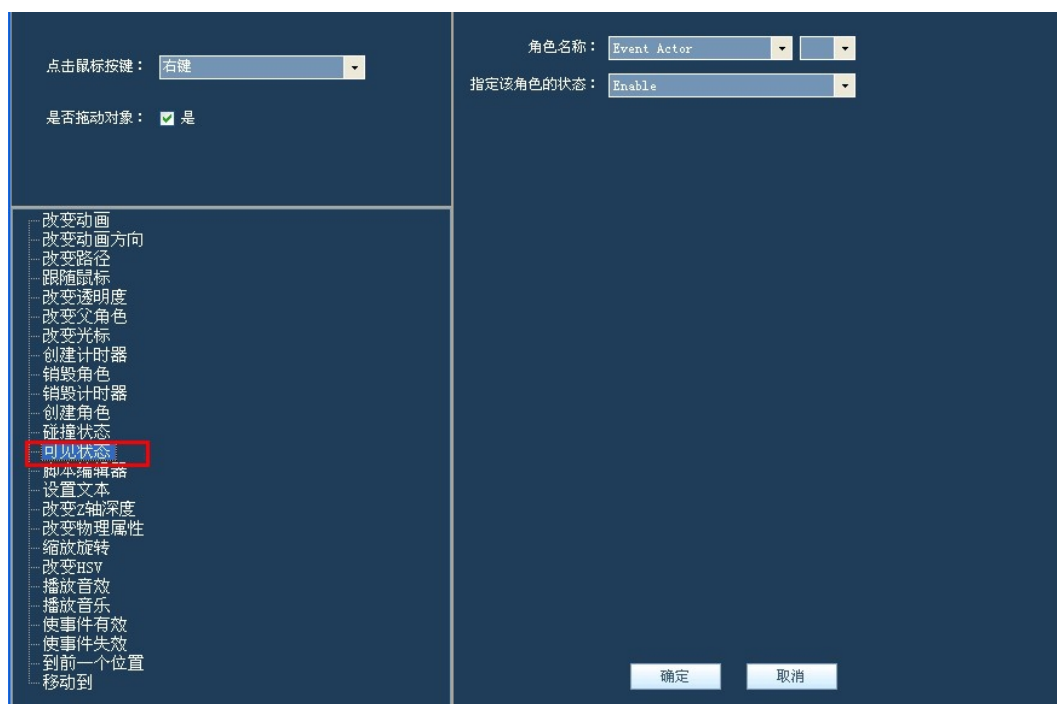


图 80

显示或隐藏选择的角色。

- **角色名称：**

选择一个待设置的角色。

- **指定该角色状态：**

“Enable”（显示角色），“Disable”（隐藏角色）和 “Don’ t draw, but allow events”（不绘制，但是接受事件）。

- **脚本编辑器：**

输入一组 C 语言脚本代码，在事件发生时执行，这是角色内部的脚本。如果需要更多的脚本编辑提示，可以点击右上角的“详细编辑”按钮。

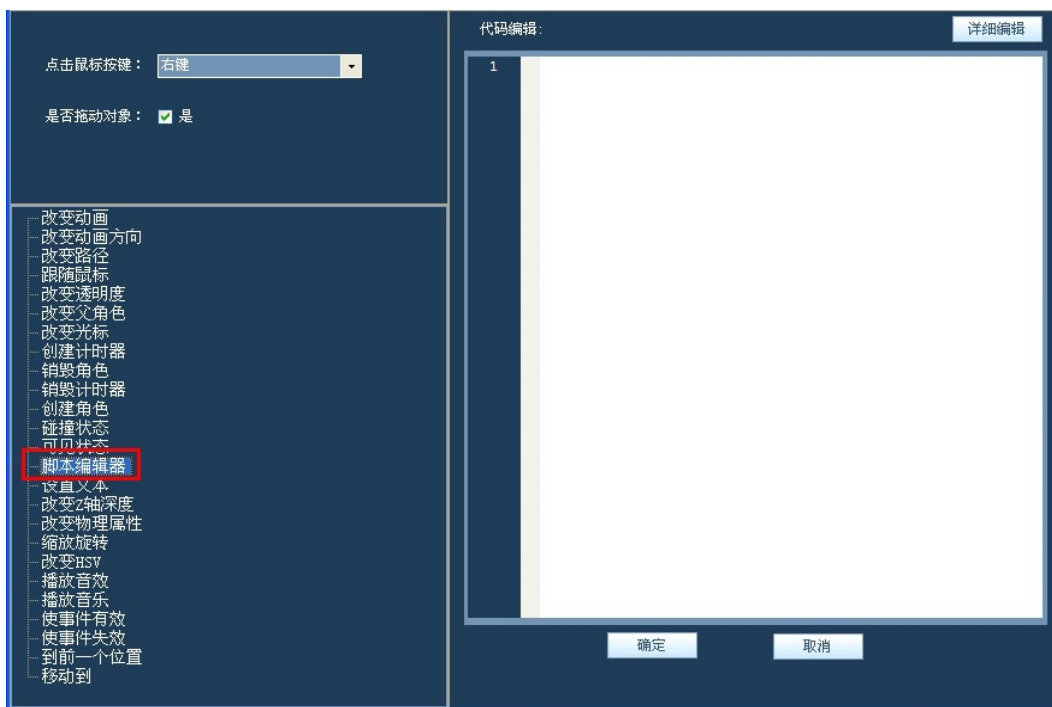


图 81

- **设置文本：**

设置或改变角色显示的文本。只有当前角色有文本内容后该行为才有效。



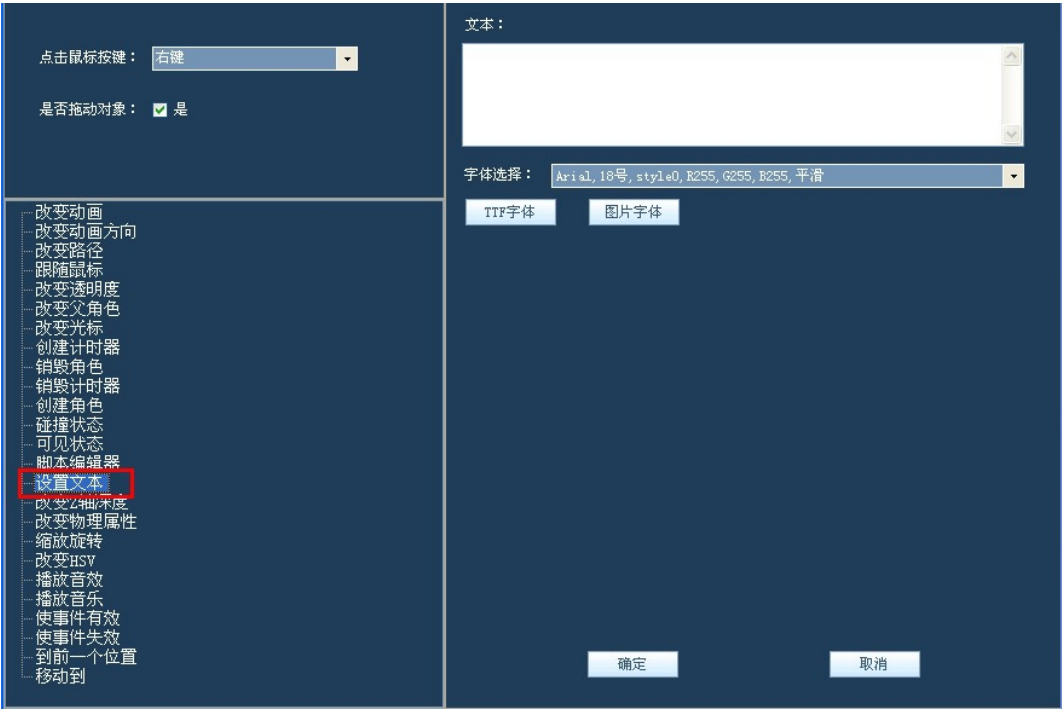


图 82

· 改变 Z 轴深度：

改变选择角色在游戏中所处的图层。

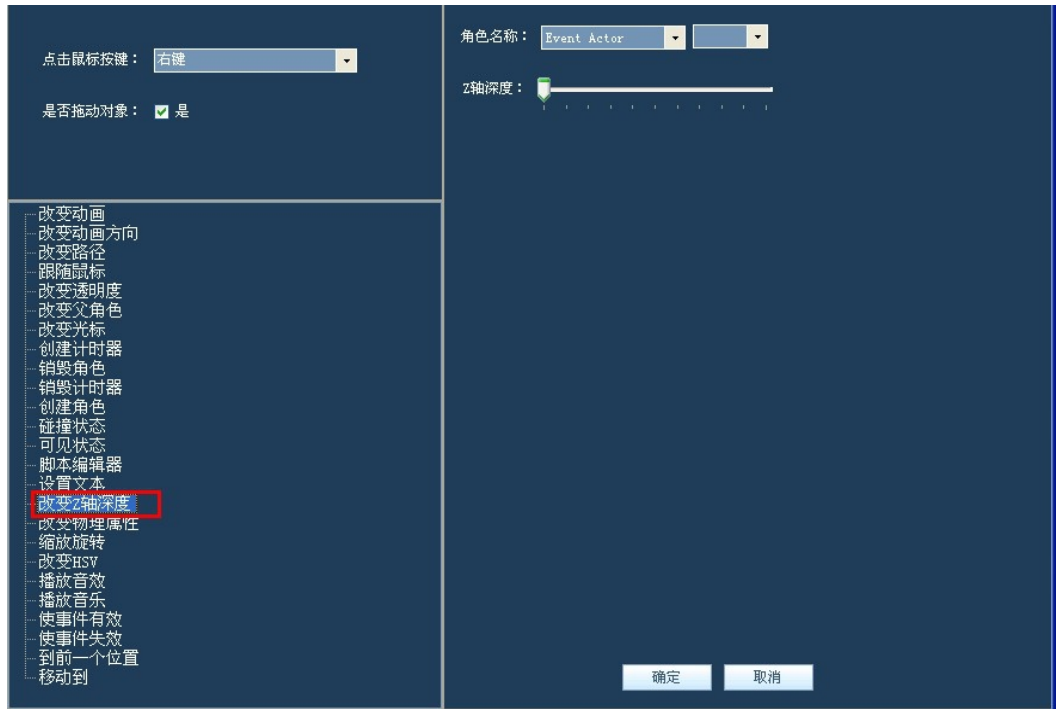


图 83

· 角色名称：

选择要改变图层深度的角色。

· Z 轴深度：

使用滑动条设置深度，越往右越高(图层越靠前)。

#### - 改变物理属性：

改变角色的物理属性。通常是配合物理世界使用。



图 84

#### • 角色名称：

选择要改变物理属性的角色。

#### • 是否开启物理属性：

当游戏世界需要模拟物理世界时，都需要勾选该项。

#### • 静态物体：

静止的物体，在物理世界中永远不会运动。静态物体可以接受所有物理世界中的事件。

#### • 矩形物体：

矩形、类似矩形的物体都可以勾选。

#### • 圆形物体：

圆形，类似圆形的物体都可以勾选。

#### • X、Y 轴的初始速度：

物理世界存在加速度，所以需要设置角色 X、Y 轴方向的初始速度。

#### • 密度：

物体的密度，用来替代设置物体的质量。密度越大，质量越大。

- **摩擦系数：**

两个物体相互摩擦力的大小，系数越大摩擦力越大。

- **弹力系数：**

两个物体相互碰撞产生的弹力的大小，弹力系数越大弹力也越大。

- **空气阻力：**

物体在上升或下降中会有空气阻力，阻力越大，物体的运动速度越来越慢。

- **旋转阻力：**

物体在做旋转运动时会有旋转阻力，阻力越大，旋转速度越慢。

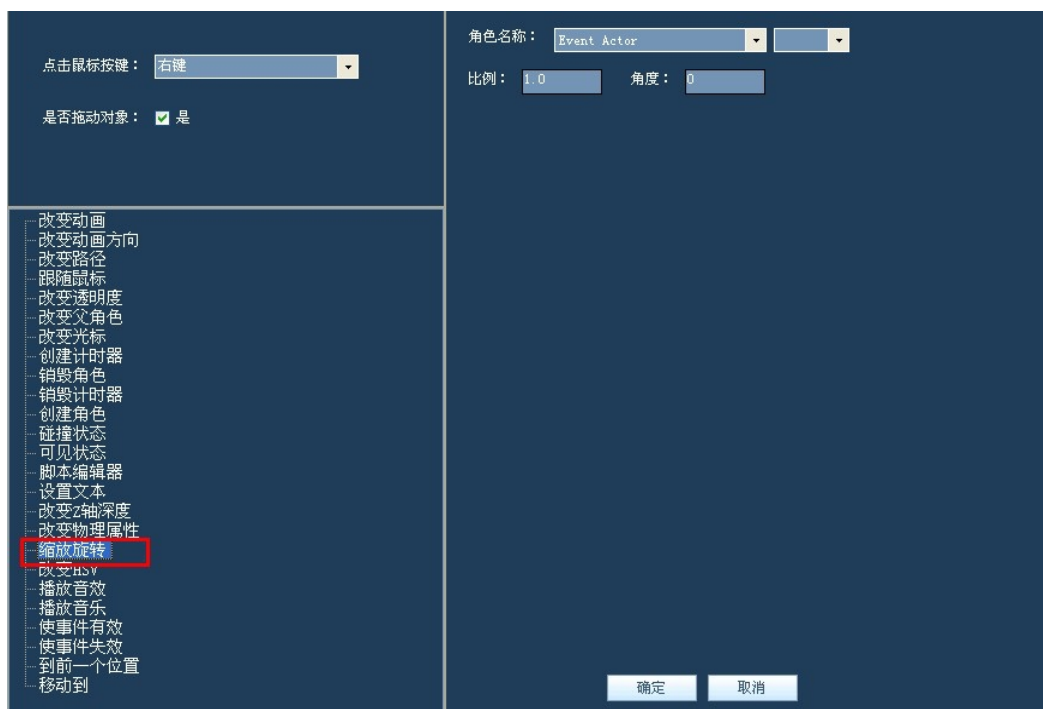


图 85

- **角色名称：**

选择要缩放旋转的角色。

- **比例：**

设置缩放的比例大小。

- **角度：**

设置旋转的角度

- **改变 HSV：**

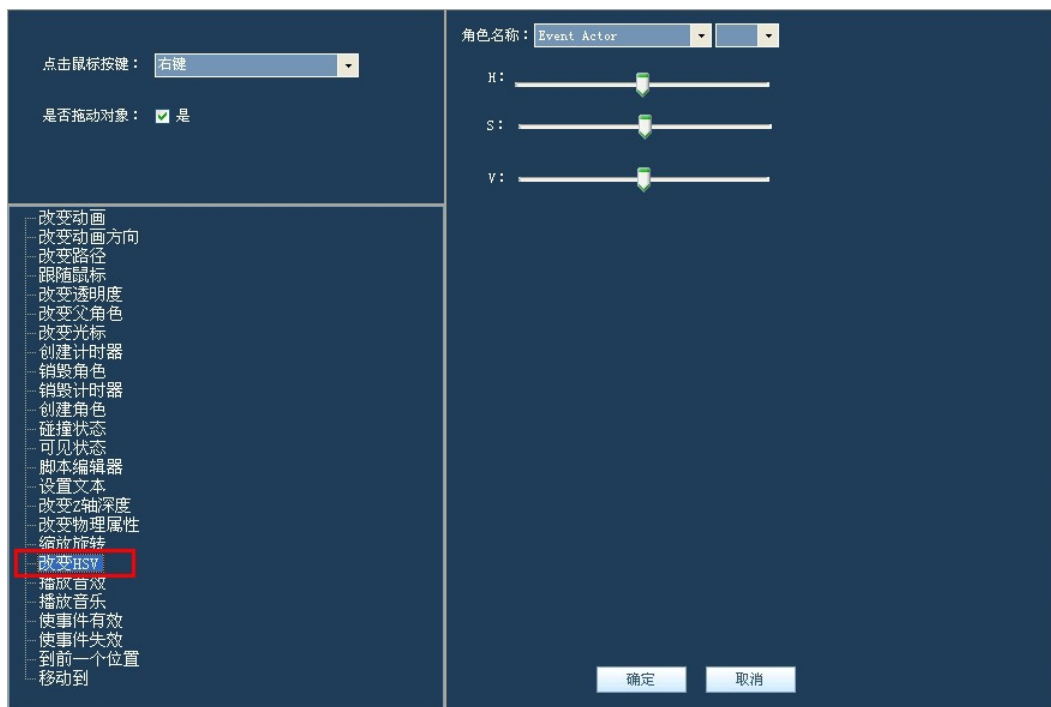


图 86

- **角色名称：**

选择要改变 HSV 的角色。

**H（色相）：**改变角色的色相。

**S（色调）：**改变角色的色调。

**V（饱和度）：**改变角色的饱和度。

- **播放音效：**

播放一个选择的的声音文件。

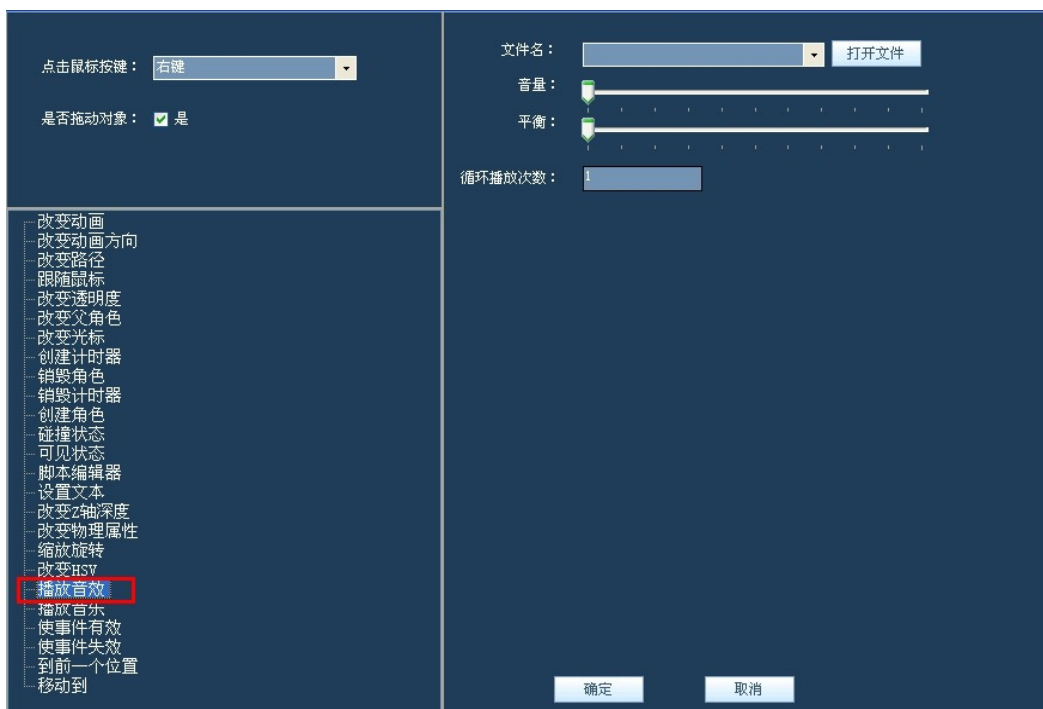


图 87

- **文件名：**  
选择一个声音文件（.wav，.ogg 格式）。
- **音量：**  
设置声音音量。
- **平衡：**  
设置声音的声像(立体声)。
- **循环播放次数：**  
设置循环次数(1 到 65000 次，“0”为一直循环播放)。
- **最大同时发声数：**  
可以在“游戏设置”中调整。
- **播放音乐：**  
播放一个选择的音乐文件。

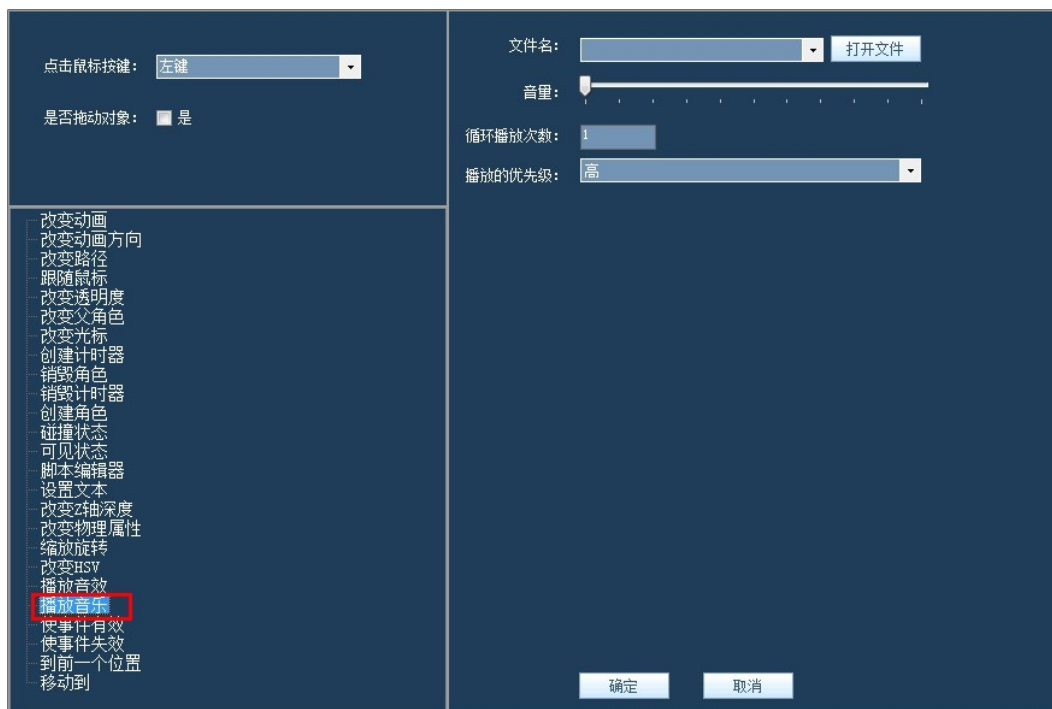


图 88

- **文件名：**

选择音乐文件（.wav, .ogg 格式）。

- **音量：**

设置音乐音量。

- **循环播放次数：**

设置循环次数（1 到 65000 次，“0”为一直循环播放）。

- **播放的优先级：**

在较慢的低端智能手机，音乐可能会读取得非常慢，设置“播放优先级”有助于满足速度要求。

- ✓ 高：在读取音乐文件时暂停游戏，这是最快的音乐模式。
- ✓ 中：读取音乐文件时游戏不停止，但此模式可能会较慢。
- ✓ 低：游戏会在正常速率下运行，但音乐文件读取得非常慢。

- **注意：**

- ✓ 音乐文件并没有完全读取进内存中。
- ✓ 一次只能播放一个音乐文件。

- **使事件有效：**

激活指定事件。

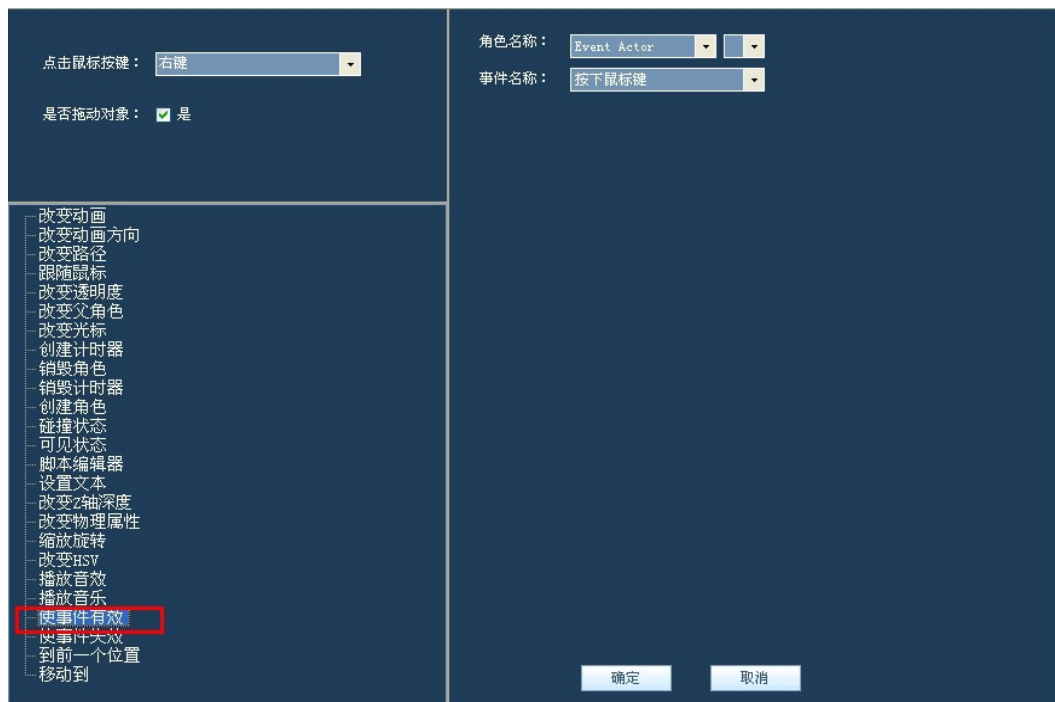


图 89

- **角色名称：**  
选择待激活事件的角色。
- **事件名称：**  
选择待激活的事件。
- **使事件无效：**  
设置指定事件无效。

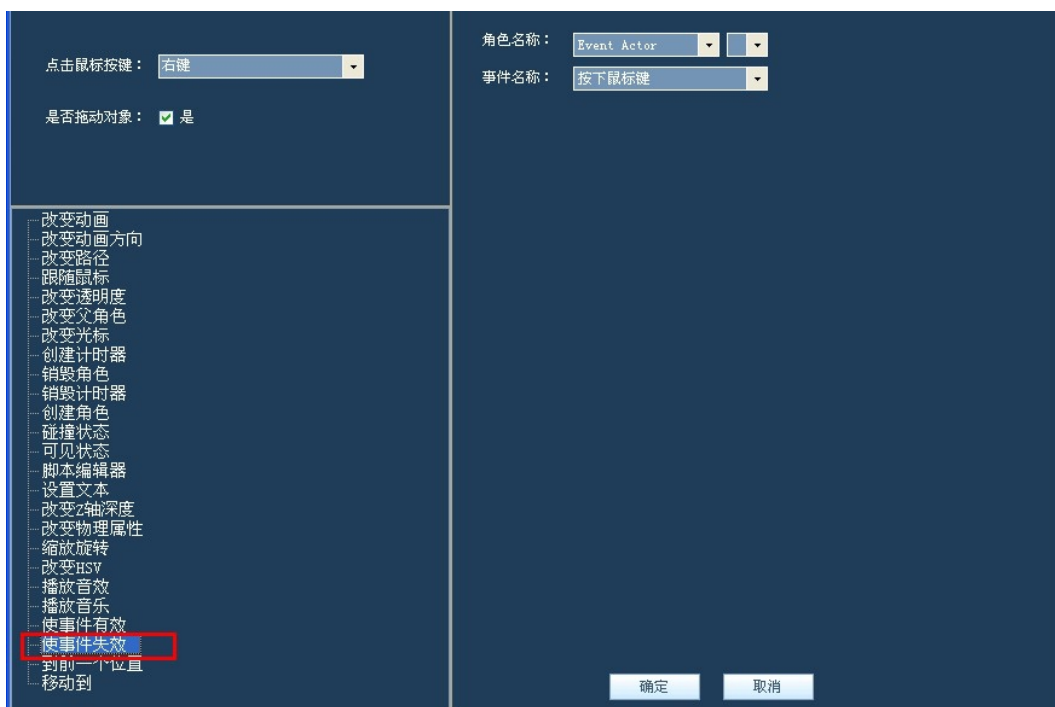


图 90

- **角色名称：**  
选择待失效事件的角色。
- **事件名称：**  
选择待失效的事件。
- **到前一个位置：**  
发送角色之前的帧位置。

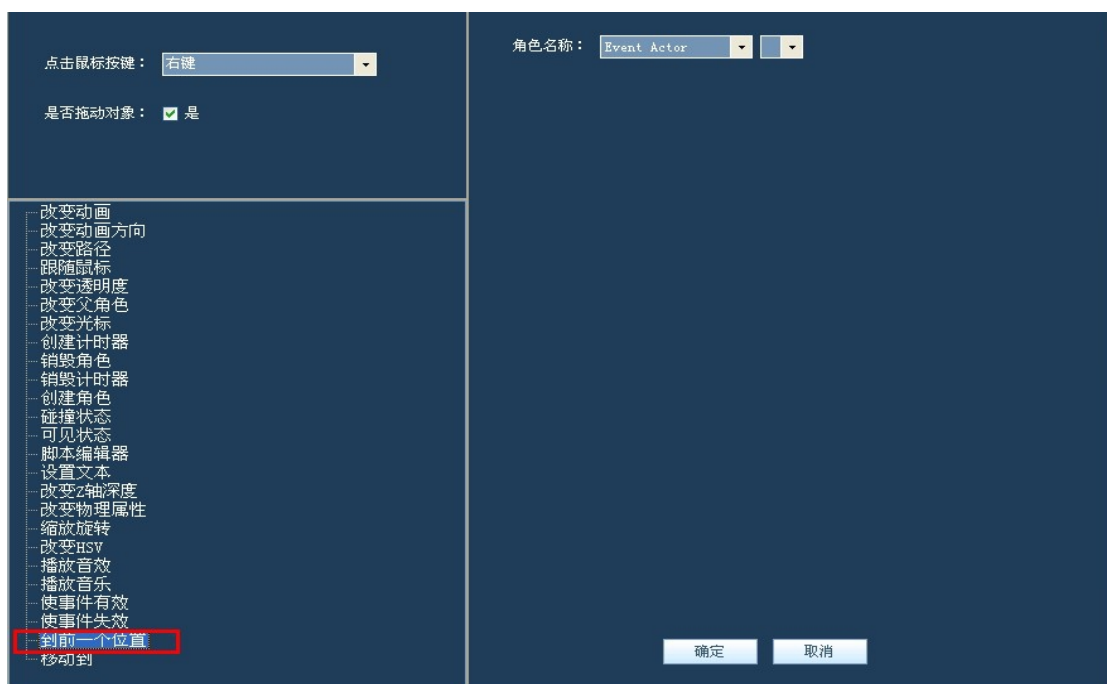


图 91

- **角色名称：**  
选择一个角色。
- **移动到：**  
在指定速率下移动角色到指定位置。角色到达指定位置就是“移动完成”事件的触发条件。



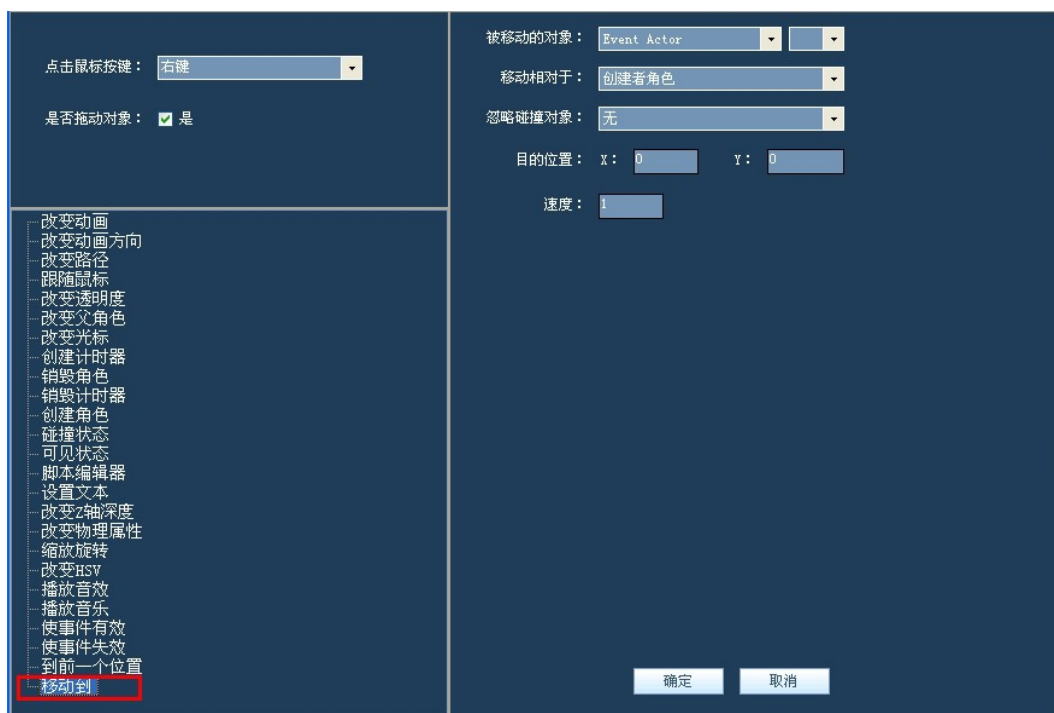


图 92

- **被移动的角色：**  
选择待移动角色。
  - **移动相对于：**  
角色能相对游戏中心移动，相对鼠标移动或者相对游戏里的任何角色。
  - **忽略碰撞角色：**  
设置角色在移动时避免碰撞到某一个角色。这通常用来避免和某个不需要碰撞计算的物体发生联系。
  - **目的位置：**  
设置目标位置的坐标。
  - **速度：**  
移动速率/速度。数字越大速度越快。
- ✓ 如果选择的是角色名而不是克隆名，则所有克隆的角色都会被认定。
  - ✓ 如果选择的是贴片类型的，则每个贴片的边界都会被认定。
  - ✓ 所有认定的都只是角色的边界。
  - ✓ 只有当行为执行时移动路径才会被计算，所以，如果你想避免和另一个移动的角色碰撞，必须再次调用“移动到”行为。

例子：

**1) 向右移动角色 10 个位置**

被移动的角色: "Event Actor"

移动相对于: "Event Actor"

忽略碰撞角色: (none)

X: 10

Y: 0

速度: 1000 (使用一个极大值来“瞬间”移动角色)

**2) 移动 view 到玩家的位置**

被移动的角色: "view"

移动相对于: "player"

忽略碰撞角色: (none)

X: 0

Y: 0

**3) 移动玩家到鼠标位置**

被移动的角色: "player"

移动相对于: "Mouse Position"

忽略碰撞角色: (none)

X: 0

Y: 0

**4) 移动玩家到鼠标的位置, 避免与地上的湖计算碰撞效果**

被移动的角色: "player"

移动相对于: "Mouse Position"

忽略碰撞角色: "lake"

X: 0

Y: 0

## 脚本

### 脚本编辑器

MC 脚本支持可实现使用兼容 C 语言的脚本语言，允许更高水平的开发，并且灵活方便。

在“脚本编辑器”行为里创建脚本，脚本编辑器支持多颜色的语法文字、自动缩进和语法提示功能来提高你的工作效率。对于常见的其他程序语言，脚本编辑器相当于一个集成开发环境。所有角色的行为都能被脚本调用。

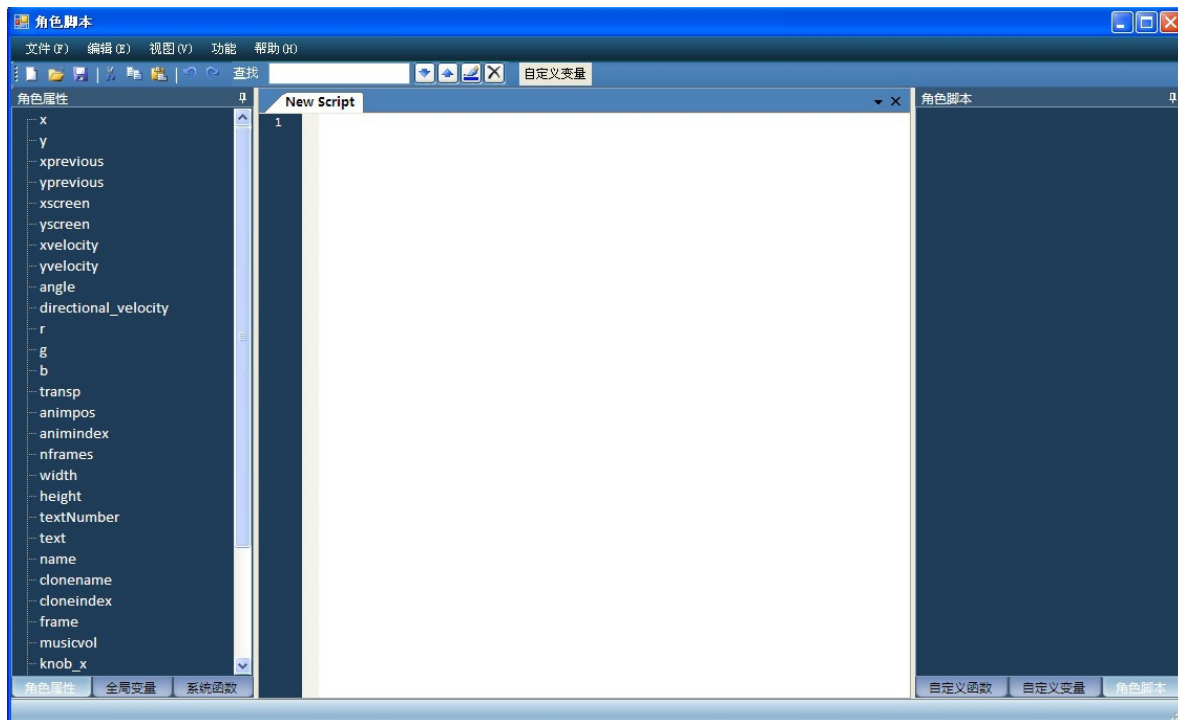


图 93

#### - 脚本编辑器输入框:

中心白色的矩形区域就是输入框，在此输入和调试脚本。

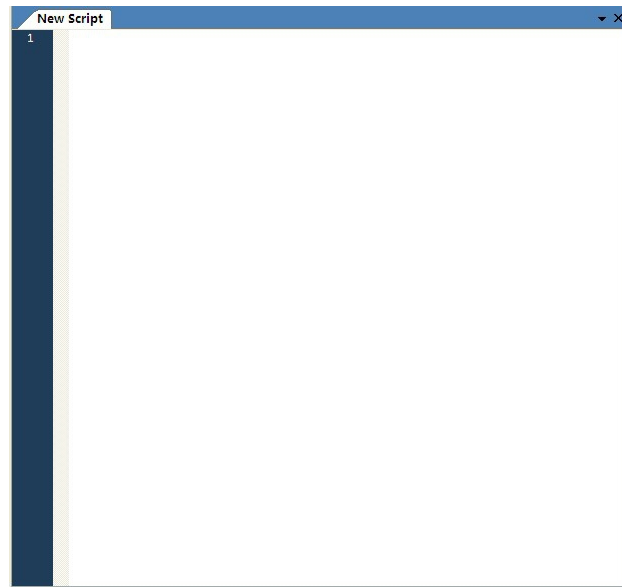


图 94

- **脚本编辑器工具栏:**

脚本编辑器工具栏位于脚本编辑器的左右两边。工具栏提供了一些选项和列表以有效访问各种功能。各个工具栏说明如下:

- **系统函数:**

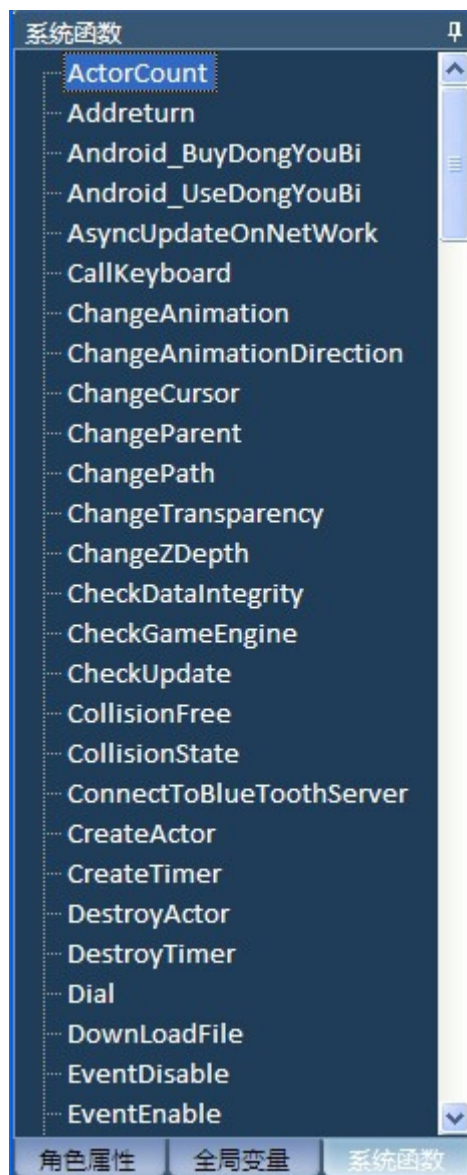


图 95

显示所有的系统预定义函数。为了使用方便，双击鼠标后将会把函数自动插入到输入框内。

- 角色属性:

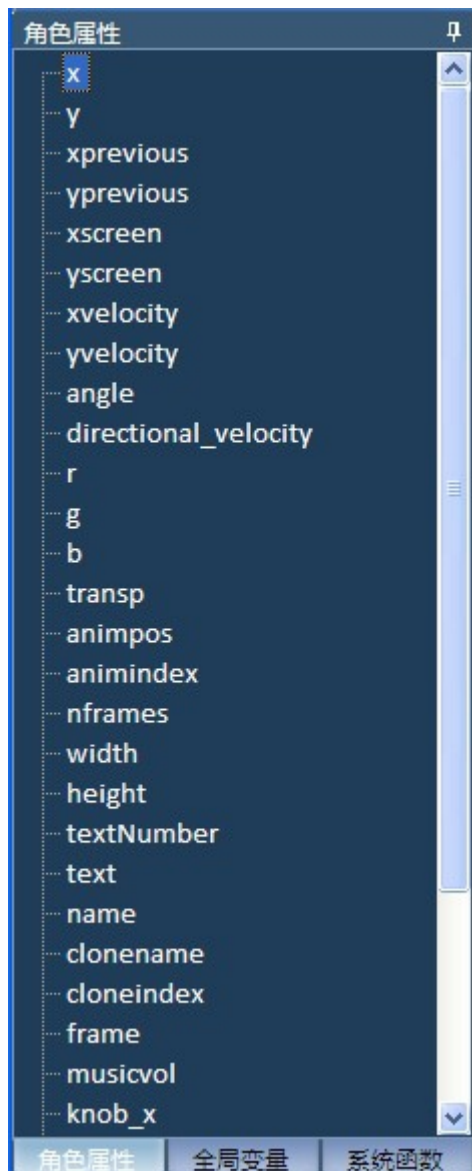


图 96

显示所有系统定义的角色属性变量。为了使用方便，双击鼠标后将会把变量自动插入到输入框内。

- **全局变量：**

显示用户在全局脚本中定义的全局变量。

- **自定义变量：**

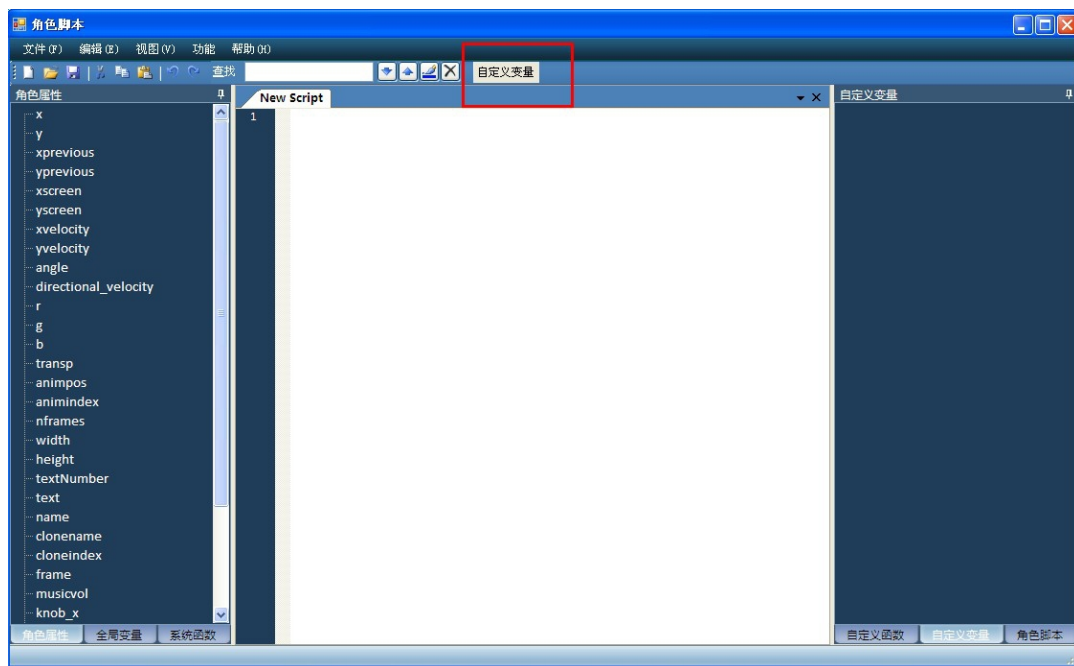


图 97

自定义变量就像一个储存值的容器，通常与 `saveVars()` / `loadVars()` 函数配合使用，所有此类变量将被保存或读取。

选择此按钮将会打开自定义变量面板以创建，编辑或删除单一变量或数组变量。变量的所有属性，除了变量名，其他都能编辑。如果要改名，需要删除该变量后重新定义。



图 98

✓ 添加新变量：

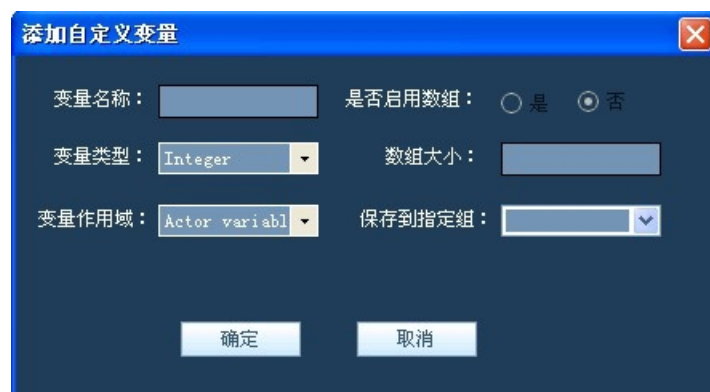


图 99

- **变量名称:**

变量名必须以字符开头，可包含字符，数字或下划线 ‘\_’；不能使用脚本的函数名和变量名。不能使用关键字命名（比如：“MyNewActor” 合法，“ CreateActor” 不合法）。

- **变量类型:**

整形（Integer），实型（real）或者字符型（string），最多只支持 255 个字符。

- **变量作用域:**

全局(Global)或者角色(Actor)变量。角色变量继承自角色结构，因此，角色变量能像内部角色变量一样以相同方法被访问。

比如，使用本地角色变量来做比如“玩家能量”之类的事。

- **是否启用数组:**

选择“是”创建全局数组变量。角色变量是不能指定数组的。

- **数组大小:**

范围可从 0 到 65000。

- **保存到指定组:**

如果你想使用“saveVars”和“loadVars”保存或读取变量，变量将会保存在设置的“组”里。

**注意：**改变任意角色内部的变量(x, y, xscreen, yscreen, ...)对全局变量没有影响。

- **全局脚本:**

使用全局变量编辑器自由添加（数组，结构体和函数），为了有效使用全局变量你需要有较好的 C 语言基础。任何脚本都能访问全局变量。



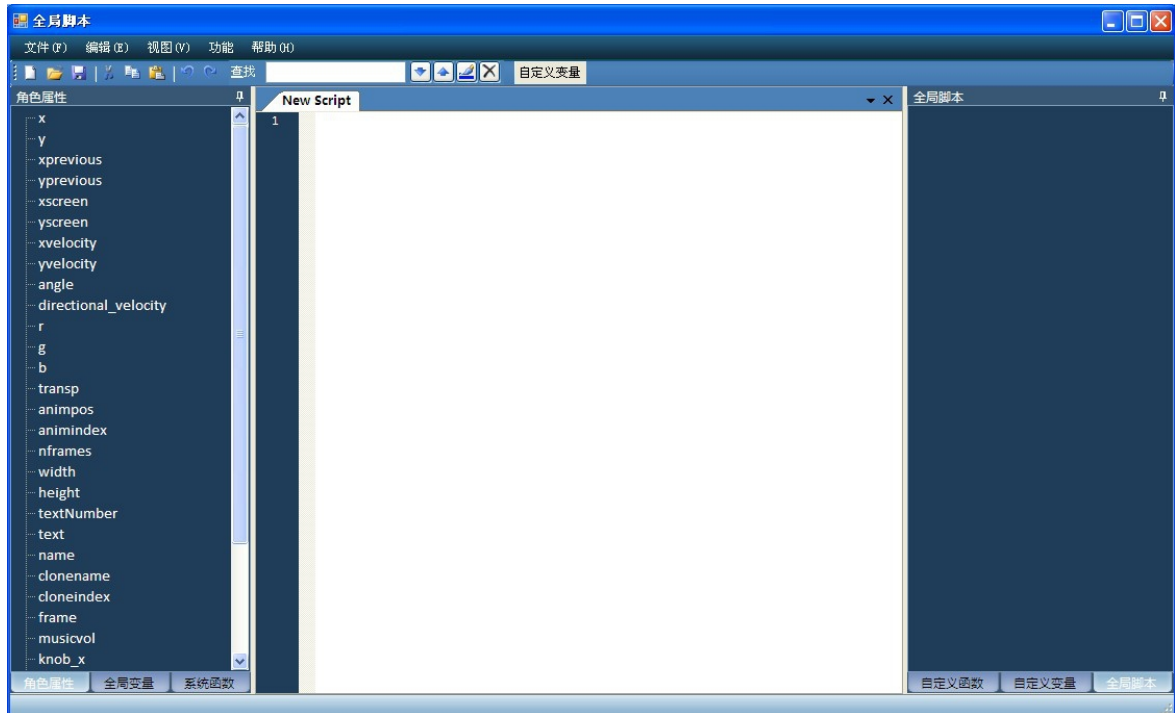


图 100

- **新建:**

输入完代码后，点击保存按钮就可以保存了。当然，别忘了输入一个有实际意义的脚本名。

- **编辑:**

双击右侧的“全局脚本”列表中的脚本名编辑代码，编辑需要修改的代码后点击“保存”按钮。

- **删除:**

在右侧的脚本名上点击鼠标右键，弹出菜单，选择“删除”。

- **导出:**

菜单“文件”→“导出当前脚本”可以将脚本存成文本文件格式，以便使用其他编辑工具进行编辑。

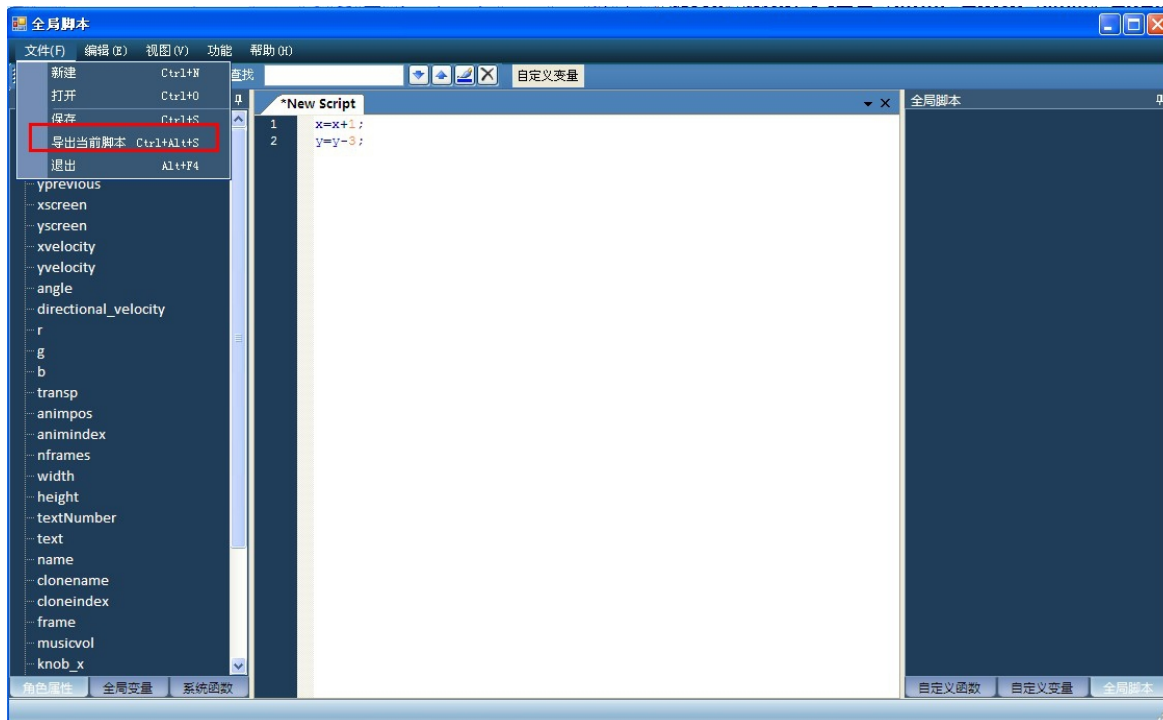


图 101

- 导入:

菜单“文件”→“打开”可以从外部文件读取之前保存的当前脚本或其他脚本。

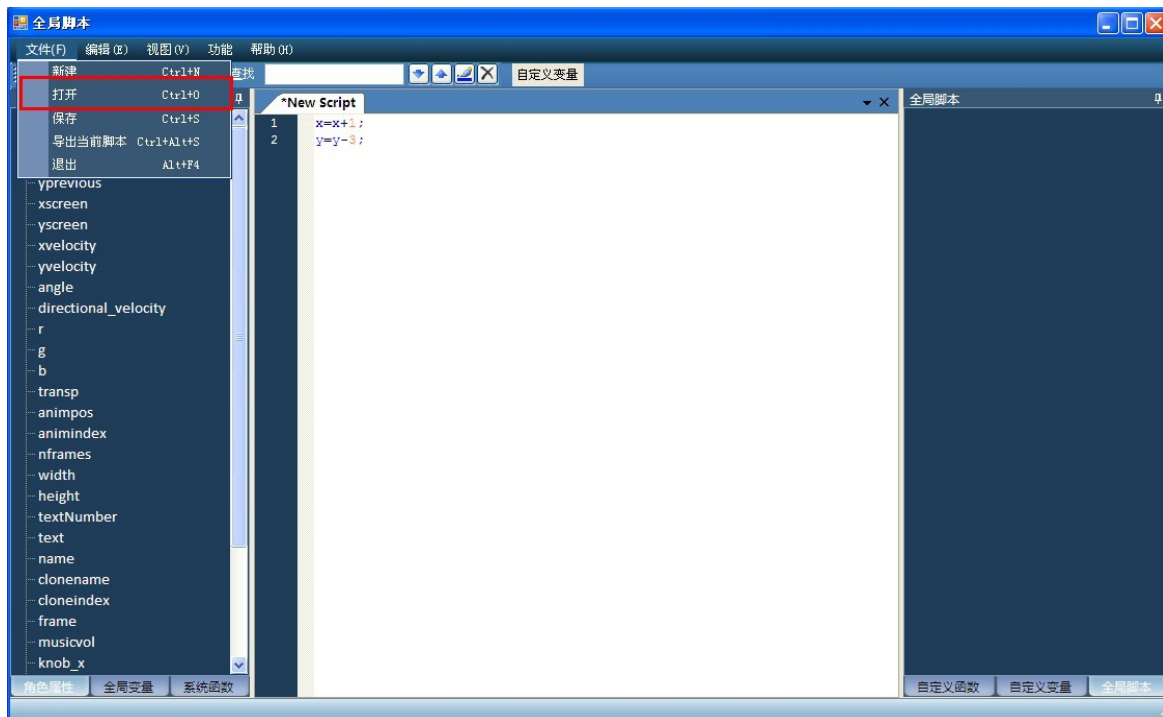


图 102

## 脚本参考

### 特殊角色

**parent**: 角色的父角色, 如果存在的话(在行为里的名字: "Parent Actor")

**creator**: 角色的创建者, 如果角色是在 "Create Actor" 行为里被创建的话(在行为里的名字: "Creator Actor")

## 角色属性

**x (double 类型)**:

x 是角色的水平坐标。X 的位置与父角色有关, 如果角色没有父角色, 则坐标与游戏坐标相关。

脚本语法:

在当前角色:

```
x = x + 5;
```

或

```
X += 5;
```

在其他角色:

```
MyActor.x += 5;
```

**y (double 类型)**:

y 是角色的竖直坐标。y 的位置与父角色有关, 如果角色没有父角色, 则坐标与游戏坐标相关。

脚本语法:

在 Key Down 事件中:

```
y = y + 5;
```

或

```
Y += 5;
```

**xprevious (double 类型)**:

角色上一帧的 x 坐标, 此变量是只读的。

**yprevious (double 类型)**:

角色上一帧的 y 坐标, 此变量是只读的。

**xscreen (double 类型)**:

角色在屏幕坐标中水平方向的位置

以 view 框左上角为 (0, 0) 点, 右下角为坐标为 (view.width, view.height)。

**yscreen (double 类型) :**

角色在屏幕坐标中竖直方向的位置

以 view 框左上角为 (0, 0) 点, 右下角为坐标为 (view.width, view.height)。

**xvelocity (double 类型) :**

角色的像素/帧 在水平方向上的速率

**yvelocity (double 类型) :**

角色的像素/帧 在竖直方向上的速率

**angle (double 类型) :**

角色移动时运动的角度(0 到 360, 沿着 X 轴逆时针方向, 注意不是旋转角度)

**directional\_velocity (double 类型) :**

角色旋转角度时的像素/帧速率(只有在使用旋转的时候才能设置此变量)

**r (double 类型) :**

红色 (0~255)

**g (double 类型) :**

绿色 (0~255)

**b (double 类型) :**

蓝色 (0~255)

**transp (double 类型) :**

角色的透明度设置参数 (0.0~1.0)

**animpos (double 类型) :**

角色的动画帧。Animpos 只会改变当前动画的帧。要想在不重置当前帧的情况下改变动画, 在 Change Animation 行为中使用” No change” 选项 (脚本里为” NO\_CHANGE”)。

**animindex (int 类型) :**

使用 animindex 搜索角色的当前实际动画(从 0 开始)。角色的每一个动画都有唯一的序号分配到 animindex 中(可以用此方法找到当前哪一组动画正在运行)

如果你的角色有三组动画

第一组动画的 animindex = 0

第二组动画的 animindex = 1

第一组动画的 animindex = 2

此变量是只读的

**nframes (int 类型) :**

当前动画的帧号, 此变量是只读的

**width (int 类型) :**

动画的实际宽度, 此变量是只读的

**height (int 类型) :**

动画的实际高度, 此变量是只读的

**textNumber (double 类型) :**

如果角色显示文本, 可设置文本号

脚本语法:

```
MyActor.textNumber=5;
```

**Text (char[1024]) :**

如果角色显示文本, 使用如下方法设置文本:

脚本语法:

```
strcpy (MyActor.text, "Hello World");
```

文本的最长字符数为 255

**name (char[33]) :**

角色名, 此变量只读

**clonename (char[42]) :**

克隆角色的名字(有原有的角色名加上克隆的号码组成)。此变量只读

**cloneindex (int 类型) :**

克隆角色的序列(从 0 开始)。如果有两个克隆角色, 序列为: myclone.0, myclone.1, ... 此变量只读

**frame (int 类型) :**

游戏帧计数器。此变量只读且全局。

**musicvol (double 类型) :**

音乐音量 (0.0~1.0) 此变量只读

脚本语法:

```
musicvol=.7;
```

**knob\_x (double 类型) :**

**knob\_y (double 类型) :**

**knob\_r (double 类型) :**

**knob\_ang (double 类型) :**

**knob\_rdc (double 类型) :**

这五个变量只有在多点触控的情况下才会发生变化，先来解释一下 MC 实现多点触控的原理：

1. 当多个手指头触碰到屏幕上的时候，我们去计算这所有点的坐标的平均值，从而得到取平均后的坐标 (**knob\_x**, **knob\_y**)。

2. 对于这些点，我们构建一个圆，这个圆的半径为这些点到圆心的平均值就是 **knob\_r**。

3. 在我们手指捏合的过程中，这个圆的半径在缩小，那么，当前这一帧的圆半径与上一帧的差，就是 **knob\_rdc**。

4. **knob\_ang** 是用来判断手指的旋转趋势的。当手指相对屏幕做顺时针运动的时候，**knob\_ang** 就为正值，当做逆时针运动的时候，**knob\_ang** 为负值。

**real\_fps (int 类型) :**

游戏的帧率(每秒帧数)此变量只读且全局

**xmouse (int 类型) :**

鼠标在屏幕的水平方向上的坐标。此变量只读且全局。

以 view 框左上角为 (0, 0) 点，右下角为坐标为 (view.width, view.height)。

脚本语法：

```
xscreen = xmouse;
```

**ymouse (int 类型) :**

鼠标在屏幕的竖直方向上的坐标。此变量只读且全局。

以 view 框左上角为 (0, 0) 点，右下角为坐标为 (view.width, view.height)。

脚本语法：

```
yscreen = ymouse;
```

**phy\_velocity (double 类型) :**

物理事件中物理物体的碰撞速度。此变量只读且全局。

**view\_scale (double 类型):**

在设备中 view 的缩放比例。

## 系统函数

### 一般函数

#### AddAnimation:

```
int AddAnimation(const char * actorName, const char * actionName, const char * imgPath,  
int isReplace);
```

功能：增加角色动画

参数：

actorName: 游戏里的任意角色名

actionName: 要添加的角色动画名

imgPath: 图片路径

isReplace: 是否强制替换（0 不强制替换，1 强制替换）

#### AddAction:

```
int AddAction(const char *actorName, const char *actionName, int fps, int bReplace);
```

功能：使用已处理过的动画图片（.ani 格式，暂时未提供转工具）替换目标角色动画，功能同 AddAnimation 函数

参数：

actorName: 游戏里的任意角色名

actionName: 要添加的角色动画名

fps: 动画帧速率

isReplace: 是否强制替换（0 不强制替换，1 强制替换）

#### Addreturn:

```
int Addreturn(const char *oldString, int lineWidth, char *newString, const char  
*ttfName);
```

功能：将字符串换行后给新的字符串

参数：

oldStr: 旧字符串变量

length: 每行的长度

newStr: 旧串换行后赋给的新串变量

ttfName: 字体名称

(根据字体的名称来定义可以在运行后的 test 文件夹中找到对应的字体名称, 如

"DROIDSansFallback.TTF0000182552552551", 说明使用的字体名称是

"DROIDSansFallback.TTF", 后面的"0000182552552551"可以拆分为"000"、"018"、"255255255"、

"1", 分别代表字体字形为常规 (0 常规、1 粗体、2 斜体、3 下划线)、大小为 18 号字体、rgb

值即字体颜色 (r=255, g=255, b=255)、是否平滑 (0 不平滑、1 平滑))

脚本语法:

```
Addreturn(strTemp, SD_MEDIUM_SCREEN, contActor->text, MSYH_13_WHITE);
```

### ChangeAnimation:

```
int ChangeAnimation(char *actorName, char *animationName, int state);
```

功能: 改变动画并设置起始状态 (FORWARD, BACKWARD, STOPPED, NO\_CHANGE)

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor": 接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在 "Create Actor" 行为里被创建的话
- 游戏里的任意角色

animationName: 合法的角色名

state: 动画状态 (FORWARD, BACKWARD, STOPPED, NO\_CHANGE)

脚本语法:

```
ChangeAnimation("Event Actor", "Animation", STOPPED);
```

### ChangeAnimationDirection:

```
int ChangeAnimationDirection(char *actorName, int state);
```

功能: 改变动画的状态 (FORWARD, BACKWARD, STOPPED).

成功返回 1, 否则返回 0



参数:

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

脚本语法:

```
ChangeAnimatonDirection("Ball", FORWARD);
```

### ChangeCursor:

```
int ChangeCursor(char *actorName, char *imgName, int nFramesH, int nFramesV, int hotSpotX, int hotSpotY);
```

功能: 改变角色的光标

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

imgName: 图像文件路径与游戏路径相关

nFramesH: 水平帧数

nFramesV: 竖直帧数

hotSpotX: 光标的 X 热点

hotSpotY: 光标的 Y 热点

脚本语法:

```
ChangeCursor("Event Actor", "Ball.png", 1, 1, 0, 0);
```

**ChangeParent:**

```
int ChangeParent(char *actorName, char *parentName);
```

功能: 改变角色的父角色

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

parentName:任意角色名(包括"Event Actor", "Parent Actor", "Creator Actor" and "no parent")

脚本语法:

```
ChangeParent("MyActor", "Event Actor");
```

**ChangePath:**

```
int ChangePath(char *actorName, char *pathName, int axis);
```

功能: 改变角色路径并设置轴向(X\_AXIS, Y\_AXIS, BOTH\_AXIS, NONE\_AXIS)

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

pathName:任意路径名(包括"no path"和"random path")

axis: 跟随路径方式(X\_AXIS, Y\_AXIS, BOTH\_AXIS, NONE\_AXIS)

脚本语法:

```
ChangePath("MyActor", "path1", BOTH_AXIS);
```

**ChangeTransparency:**

```
int ChangeTransparency(char *actorName, double transp);
```

功能：改变角色的透明度

成功返回 1，否则返回 0

参数：

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色，如果存在的话
- "Creator Actor": Event Actor 的创建者，如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

transp: 角色透明度，0.0~1.0 越来越透明

脚本语法：

```
ChangeTransparency("Event Actor", 0.64);
```

**ChangeZDepth:**

```
int ChangeZDepth(char *actorName, double zdepth);
```

功能：改变角色深度

成功返回 1，否则返回 0

参数：

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色，如果存在的话
- "Creator Actor": Event Actor 的创建者，如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

zdepth: 角色深度，0.0 到 1.0

脚本语法：

```
ChangeZDepth("My Actor", 0.70);
```

**CheckDataIntegrity**

```
int CheckDataIntegrity(const char *gameName);
```

功能：检测数据完整性。

参数：

gameName 更新包名

返回值：

- 0: 文件完整;
- 1: check.ini 不存在;
- 2: 缺少文件;
- 3: 本文件的长度和实际长度不一致。

**CheckGameEngine:**

```
int CheckGameEngine(int *engineSize);
```

功能：检测引擎是否有更新。

参数：

engineSize 引擎大小

**CheckUpdate:**

```
int CheckUpdate(const char *gameName);
```

功能：检测是否可以更新。

参数：

gameName 更新包名

返回值：

- 1: 有更新包;
- 0: 没有更新包;
- 1: SOAP 失败。

**CollisionResponse:**

```
int CollisionResponse(int moveType, int massType, double massEventActor, double  
massCollideActor, double eventVelocityMultiplier, double collideVelocityMultiplier);
```

功能：在对象碰撞期间使用物理碰撞移动（只能在碰撞及碰撞结束事件中使用）

成功返回 1，否则返回 0

参数：

moveType: MOVE\_EVENT\_ACTOR\_ONLY(只移动事件对象), MOVE\_COLLIDE\_ACTOR\_ONLY(只移动碰撞对象), MOVE\_BOTH\_ACTORS(都移动)

massType: USE\_CALCULATED\_MASS(使用预定义质量), USE\_SPECIFIED\_MASS(使用指定质量)

massEventActor, massCollideActor:

当 massType = USE\_CALCULATED\_MASS, 用此变量来控制质量的倍数

当 massType = USE\_SPECIFIED\_MASS, 此变量就是质量

(数值必须大于 0.0)

eventVelocityMultiplier, collideVelocityMultiplier:最终速率倍数

脚本语法：

```
CollisionResponse(MOVE_EVENT_ACTOR_ONLY, USE_CALCULATED_MASS, 1.000000, 1.000000, 1.000000, 1.000000);
```

**CollisionState:**

```
int CollisionState(const char *actorName, int state);
```

功能：激活或关闭碰撞

成功返回 1，否则返回 0

参数：

actorName:

- "Event Actor":接收当前事件的对象
- "Parent Actor": Event Actor 的父类, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话

- 游戏里的任意对象

state:激活(ENABLE)或关闭(DISABLE)

脚本语法：

```
CollisionState("Event Actor", DISABLE);
```

**ConnectToBlueToothServer:**

```
int ConnectToBlueToothServer(const char *ServerName);
```

功能：连接到蓝牙服务器中的某一个服务，成功返回 0，失败返回错误代码。

参数：

ServerName 连接的服务名

**CreateActor:**

```
Actor *CreateActor(const char *creatorName, const char *animationName, const char  
*parentName, const char *pathName, int xpos, int ypos, int absolutePosition);
```

功能：由脚本创建角色

成功返回对象，否则对象无效(cloneindex = -1 和 name = "")

参数：

creatorName:任意合法的对象名

animationName:合法的对象名或"no animation"

parentName:任意对象名(包括"Event Actor", "Parent Actor", "Creator Actor" 和 "no parent")

pathName:任意路径名(包括"no path" and "random path").

xpos, ypos:新对象的起始坐标

absolutePosition:是否绝对坐标(true 或 false)

脚本语法：

```
CreateActor("GamePaddle", "Paddle_Animation", "no parent", "no path", 0, 0,  
false);
```

**CreateTimer:**

```
int CreateTimer(char *actorName, char *timerName, int milliseconds)
```

功能：创建一个定时器，这个计时器必须是已存在并设置好的，它更像“使用计时器”。

成功返回 1，否则返回 0

参数：

actorName:

- "Event Actor":接收当前事件的对象

- "Parent Actor": Event Actor 的父类, 如果存在的话
  - "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
  - 游戏里的任意对象
- timerName: 定时器的名字(对象名必须在计时器面板定义过)
- milliseconds: 定时器的时间间隔

脚本语法:

```
CreateTimer("Event Actor", "MyTimer", 2000);
```

### DestroyActor:

```
int DestroyActor(const char *actorName);
```

功能: 删除角色

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor": 接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

脚本语法:

```
DestroyActor("Ball.2"); //删除 Ball.2 角色
```

```
DestroyActor("Ball"); //删除所有 ball 角色
```

### DestroyTimer:

```
int DestroyTimer(char *timerName);
```

功能: 删除已创建的计时器, 必须在有计时器的行为里调用

参数:

timerName: 计时器名字

成功返回 1, 否则返回 0

脚本语法:

```
DestroyTimer("MyTimer");
```

**Dial:**

```
void Dial (const char* phoneNumber) ;
```

参数:

phoneNumber: 要拨打的电话号码

脚本语法:

```
Dial ("10000") ;
```

**EventDisable:**

```
int EventDisable(char *actorName, unsigned long event);
```

功能: 使一个角色事件失效

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

event:

以下事件能用 EventDisable 设置成无效:

EVENTMOUSEBUTTONDOWN

EVENTMOUSEBUTTONUP

EVENTMOUSEENTER

EVENTMOUSELEAVE

EVENTANIMATION

EVENTANIMATIONFINISH

EVENTPATHFINISH



EVENTMOVEFINISH

EVENTKEYDOWN

EVENTKEYUP

EVENTTIMER

EVENTCOLLISION

EVENTCOLLISIONFINISH

EVENTCREATE

EVENTDESTROYACTOR

EVENTOUTOFVISION

EVENTALL

### 脚本语法:

```
EventDisable("Event Actor", EVENTMOUSEBUTTONDOWN);
```

### EventEnable:

```
int EventEnable(char *actorName, unsigned long event)
```

功能: 激活一个角色事件

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor": 接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在 "Create Actor"

行为里被创建的话

- 游戏里的任意角色

event:

以下事件能用 EventEnable 激活:

EVENTMOUSEBUTTONDOWN

EVENTMOUSEBUTTONUP

EVENTMOUSEENTER

EVENTMOUSELEAVE

EVENTANIMATION

EVENTANIMATIONFINISH

EVENTPATHFINISH

EVENTMOVEFINISH

EVENTKEYDOWN

EVENTKEYUP

EVENTTIMER

EVENTCOLLISION

EVENTCOLLISIONFINISH

EVENTCREATE

EVENTDESTROYACTOR

EVENTOUTOFVISION

EVENTACTIVATIONEVENT

EVENTALL

脚本语法:

```
EventEnable("Event Actor", EVENTMOUSEBUTTONDOWN);
```

### FollowMouse:

```
int FollowMouse(const char *actorName, int axis, int spotX, int spotY);
```

功能: 让角色跟随鼠标轴向 (X\_AXIS, Y\_AXIS, BOTH\_AXIS, NONE\_AXIS)

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor": 接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在 "Create Actor"

行为里被创建的话

- 游戏里的任意角色

axis: 鼠标轴向, 取值: X\_AXIS, Y\_AXIS, BOTH\_AXIS, NONE\_AXIS

spotX, spotY: 相对于当前角色位置的偏移量

脚本语法:

```
FollowMouse("GamePaddle", X_AXIS, 0, 0);
```

#### **getDimensions:**

```
void getDimensions(const char *inputText, const char *fontName, int *width, int *height);
```

功能: 获得文本角色的宽度和高度

参数:

inputText: 字符串变量

fontName: 字体文件名 (导出 dat 后可看到相关字体信息), 如果是图片字体则是图片的文件名

width: 行宽度

height: 行高度

(根据字体的名称来定义可以在运行后的 test 文件夹中找到对应的字体名称, 如

"DROIDSANSFALLBACK. TTF0000182552552551", 说明使用的字体名称是

"DROIDSANSFALLBACK. TTF", 后面的"0000182552552551"可以拆分为"000"、"018"、"255255255"、

"1", 分别代表字体字形为常规 (0 常规、1 粗体、2 斜体、3 下划线)、大小为 18 号字体、rgb

值即字体颜色 (r=255, g=255, b=255)、是否平滑 (0 不平滑、1 平滑))

脚本语法:

```
getDimensions(contActor->text, "DROIDSANSFALLBACK. TTF0010202552552551",  
&lineWidth, &lineHeight);
```

#### **GetImeWords:**

```
void GetImeWords(const char hzStr[], char *wordStr);
```

功能: 通过汉字得到词语。

参数:

hzStr[]: 当前选择的字

wordStr: 得到的联想的字

#### **GetParentActor:**

```
Actor *GetParentActor(const char *clonename);
```

功能：获得角色的父角色。成功返回该角色的父角色，失败返回无效角色（name = “Axis”，cloneindex = 0）。

参数：

clonename：角色名。

#### **GetRuntime:**

```
unsigned long GetRuntime();
```

功能：得到系统的运行时间，精确到毫秒。

#### **GetVersionInfo:**

```
int GetVersionInfo(const char *gameName);
```

功能：获取游戏的版本信息。

参数：

gameName：更新包名

#### **LocalUpdate:**

```
int LocalUpdate();
```

功能：本地更新。

返回值：

-1：打开更新目录失败；

0：没有更新文件；

1：更新成功

#### **MoveTo:**

```
int MoveTo(char *actorName, double x, double y, double velocity, char *relativeActor,  
char *avoidActor);
```

功能：在指定的速率（速度）下移动角色至指定地点

成功返回 1，否则返回 0

参数:

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

x, y: 需要移动到的相对 relativeActor 的坐标

relativeActor: 角色能相对屏幕中心移动(绝对坐标"Game Center"), 相对鼠标("Mouse Position")或任意角色移动

Valid values:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话

行为里被创建的话

- "Game Center":角色能相对屏幕中心移动
- "Mouse Position":角色能相对鼠标位置移动
- "Collide Actor":碰撞对象
- 游戏里的任意角色

avoidActor:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话

行为里被创建的话

- 游戏里的任意角色

示例:

1) 向右移动角色 10 个位置( $x = x + 10$ ):

```
MoveTo("Collide Actor", 10, 0, 1000, "Event Actor", "");
```

(使用高速来实现瞬时移动)

2) 移动 view 到 player 的位置( $view.x = player.x$ ;  $view.y = player.y$ ):

```
MoveTo("view", 0, 0, 1, "player", "");
```

3) 移动 player 到鼠标的位置:

```
MoveTo("player", 0, 0, 1, "Mouse Position", "");
```

4) 移动 player 到鼠标的位置, 并避开地上的湖:

```
MoveTo("player", 0, 0, 1, "Mouse Position", "lake");
```

#### **pyIme:**

```
void pyIme(const char *input, char *pingying);
```

功能: 实现拼音输入法。

参数:

input: 输入的拼音

pingying: 得到的存在的拼音组合

#### **RemoveVersion:**

```
int RemoveVersion(const char *gameName);
```

功能: 删除 version.ini 文件。

参数:

gameName: 更新包的名字

#### **RotoZoomActor:**

```
int RotoZoomActor(const char *actorName, double angle, double zoomx, double zoomy, int smooth);
```

功能: 角色的旋转和缩放。

参数:

actorName: 角色名

angle: 旋转角度

zoomx: 横向缩放比率

zoomy: 纵向缩放比率

smooth: 1 代表保存图片缓存, 0 代表不保存图片缓存

**sleep:**

```
void sleep(int ms);
```

功能：休眠，延迟

参数：

ms: 休眠的时长（毫秒）

**SetPixelClick:**

```
void SetPixelClick(const char *actorName, const char *aniName, int isSaveCache);
```

功能：是否使用图片实际的像素来触发点击类事件

参数：

actorName: 角色名

aniName: 动画名

isSaveCache: true 是、false 否

**tnineIme:**

```
int tnineIme(const char *input, void *pingying, void *buf, int pingyingsize, int  
bufsize);
```

功能：拼音输入法并返回字长

参数：

input: 输入的字

pingying: 拼音输入

buf: 缓冲用于储存拼音

pingyingsize: 拼音的长度

bufsize: 缓冲大小

**ToAnteriorPosition:**

```
int ToAnteriorPosition(char *actorName);
```

功能：将角色移动到指定位置并设置成上一帧

成功返回 1，否则返回 0

参数：

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

脚本语法:

```
ToAnteriorPosition("Event Actor");
```

### TranslateNumber:

```
void TranslateNumber(double number, int splitNum, char *finishStr);
```

功能: 将数字 987654321 改变为 987, 654, 321

参数:

number: 需要转换的数字

splitNum: 每间隔 splitNum 添加一个逗号

finishStr: 转换完成后的数字

脚本语法:

```
char str[256];  
  
TranslateNumber(1236456, 2, str);  
  
sprintf(text, "%s", str);
```

### VisibilityState:

```
int VisibilityState(char *actorName, int state)
```

功能: 设置角色的可视状态。

成功返回 1, 否则返回 0

参数:

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话



– 游戏里的任意角色

state: 可视状态, 值 ENABLE, DISABLE 和 DONT\_DRAW\_ONLY (不绘制, 但能访问事件)

脚本语法:

```
VisibilityState("GameGhost", ENABLE);
```

### ZoomActor:

```
int ZoomActor(const char *actorName, double zoom);
```

功能: 对目标角色进行缩放

返回 1 表示缩放成功, 返回 0 表示缩放失败。

参数:

actorName: 目标角色的角色名

zoom: 缩放倍数, 以 1 为基准点, 小于 1 则是缩小, 大于 1 则是放大。

### 绘制及截图函数

绘制函数必须在 canvas 角色的行为里使用, (x, y) 坐标相对于角色的左上角。

### adjustHsv:

```
void adjustHsv(const char *actorName, const char *aniName, double h, double s, double v);
```

功能: 更改角色中动画的 色彩 (H), 纯度 (S), 明度 (V)

参数:

actoName: 角色名称

aniName: 动画名称

h 色彩度: 范围-1.0 ~ 1.0

s 纯度: 范围-1.0 ~ 1.0

v 明度: 范围-1.0 ~ 1.0

### draw\_from:

```
void draw_from(const char *actorname, int x, int y, double scale)
```

功能: 在 canvas 角色上绘制当前角色的图像(tile 和文本角色无法使用)

参数:

actorname: 当前绘制在 canvas 角色上的帧的角色名

x: 图像的 X 坐标

y: 图像的 Y 坐标

scale: 绘制图像的尺寸 (大于 0.0)

脚本语法:

```
draw_from("player", 0, 1, 1.0);
```

### setpen:

功能: 定义事件角色的画笔

```
void setpen(int r, int g, int b, double transp, int pensize)
```

参数:

r: 红色部分 (0~255)

g: 绿色部分 (0~255)

b: 蓝色部分 (0~255)

transparency: 透明度 (0.0~1.0)

pensize: 画笔尺寸

脚本语法:

```
setpen(0, 255, 0, 0.0, 3); //设置画笔为绿色, 不透明, 画笔尺寸 3 个像素
```

### moveto:

```
void moveto(int x, int y);
```

功能: 移动事件角色的笔到 (x, y) 坐标

脚本语法/示例:

```
screen_to_actor(&xmouse, &ymouse); //获得鼠标坐标
```

```
moveto(xmouse, ymouse); //移动画笔到鼠标坐标
```

### lineto:

```
void lineto(int x, int y);
```

功能: 用画笔在事件角色上画一条线到 (x, y) 坐标

脚本语法:

```
setpen(255, 255, 255, 0, 3);
```

```
lineto (50, 50);
```

示例:

```
screen_to_actor(&xmouse, &ymouse); //获得鼠标坐标
```

```
lineto(xmouse, ymouse); //画一条线至鼠标坐标
```

#### **putpixel:**

```
void putpixel(int x, int y);
```

功能: 用画笔在事件角色上的(x, y)坐标画一个单一的像素

脚本语法:

```
putpixel(0, 0);
```

#### **erase:**

```
void erase(int r, int g, int b, double transp);
```

功能: 用指定颜色擦除所有角色

参数:

r: 红色部分(0~255)

g: 绿色部分(0~255)

b: 蓝色部分(0~255)

transparency: 透明度(0.0~1.0)

脚本语法:

```
erase(0, 0, 0, .5);
```

#### **screen\_to\_actor:**

```
void screen_to_actor(int *x, int *y);
```

功能: 屏幕坐标转换为角色坐标(使用 screen\_to\_actor(&xmouse, &ymouse);转换鼠标坐标为当前角色坐标。比如, 此函数能用来在 canvas 角色中在正确位置上画画)

脚本语法:

```
screen_to_actor(&xmouse, &ymouse); //转换鼠标坐标为当前角色坐标
```

**actor\_to\_screen:**

```
void actor_to_screen(int *x, int *y);
```

功能：转换角色坐标为屏幕坐标

参数：

x, y: 坐标

脚本语法：

```
actor_to_screen(&xmouse, &ymouse); //转换角色(此例该角色为鼠标)坐标为屏幕坐标
```

**actor\_to\_screen1:**

```
void actor_to_screen1(const char *actorCloneName, int *x, int *y);
```

功能：转换角色坐标为屏幕坐标

参数：

actorCloneName: 角色克隆名

x, y: 坐标

脚本语法：

```
actor_to_screen1("Actor0.0",&xmouse, &ymouse); //转换角色坐标为屏幕坐标
```

**saveActorFrame:**

```
void saveActorFrame(const char*actorName, const char *path,int x, int y, int width,  
int height,int round);
```

功能：将 actorName 角色中的动画以(x, y)为原点，width 为宽，height 为高的矩形范围截取图片保存到 path 路径中（支持圆形和矩形截图，旋转、缩放后的截图）

参数：

actorName: 角色名

path: 保存路径，如果填入"test.png",则生成的图片 PC 上的 Test 文件夹下，在手机上则在安装工程的根目录下。

x, y: 矩形范围的左上角坐标

width: 矩形范围的宽度

height: 矩形范围的高度

round: 0 为矩形，1 为圆形

**savecanvas:**

```
void savecanvas(const char *path, int width, int height, int round);
```

功能：将 canvas 角色的状态（以 canvas 角色的左上角为原点，width 为宽，height 为高的矩形范围）保存到 path 路径文件中（路径为空则保存到内存中），同时支持矩形和圆形截图

参数

path: 保存路径, 如果填入"test.png", 则生成的图片 PC 上的 Test 文件夹下, 在手机上则在安装工程的根目录下。

width: 矩形范围的宽度

height: 矩形范围的高度

round: 0 为矩形, 1 为圆形

**restorecanvas:**

```
void restorecanvas()
```

功能：从内存中还原 canvas 角色

脚本语法:

```
restorecanvas();
```

**音视频函数**

声音函数使用 PlayMusic, PlayMusic2, PlaySound, PlaySound2 返回的频道(channel)

**PlayMusic:**

```
int PlayMusic(char *soundPath, double volume, int loop);
```

功能：播放音乐文件

成功返回频道 1, 否则返回 0

参数:

soundPath: 相对游戏路径

volume: 0.0 到 1.0.

loop: 循环次数 (1 到 65000 或 "0" 表示无限循环)

脚本语法:

```
PlayMusic("data/MyMusic.wav", 1.0, 1);
```

**PlayMusic2:**

```
int PlayMusic2(char *soundPath, double volume, int loop, int priority);
```

功能：播放音乐文件并设置优先级

成功返回频道 1，否则返回 0

参数：

soundPath: 相对游戏路径

volume: 0.0 到 1.0.

loop: 循环次数 (1 到 65000 或 “0” 表示无限循环)

priority: 优先级, 值: HIGH\_PRIORITY\_MUSIC, MEDIUM\_PRIORITY\_MUSIC 或

LOW\_PRIORITY\_MUSIC

脚本语法:

```
PlayMusic2("C:/WINDOWS/Media/MyMusic2.wav", 1.000000, 1, HIGH_PRIORITY_MUSIC);
```

**PlaySound:**

```
int PlaySound(char *soundPath, double volume, int loop);
```

功能：播放声音文件

成功返回声音频道 (0、2-9)，否则返回 0

参数：

soundPath: 相对游戏路径

volume: 0.0 到 1.0.

loop: 循环次数 (1 到 65000 或 “0” 表示无限循环)

脚本语法:

```
PlaySound("data/tada.wav", 1.000000, 3);
```

**PlaySound2:**

```
int PlaySound2(char *soundPath, double volume, int loop, double pan);
```

功能：用双声道播放声音文件

成功返回声音频道 (0、2-9)，否则返回 0

参数：

soundPath: 相对游戏路径

volume: 0.0 到 1.0.

loop: 循环次数 (1 到 65000 或 “0” 表示无限循环)

pan: 声道, 值: -1.0(左声道)到 1.0(右声道)

脚本语法:

```
PlaySound2("data/tada.wav", 1.000000, 1, 0.000000);
```

### **PlayVideo:**

```
void PlayVideo(const char* Path);
```

功能: 播放视频

参数:

Path: 待播放的视频路径

### **setPan:**

```
void setPan(int channel, double pan);
```

功能: 设置声音的声道(音乐无效)

参数:

channel: 使用 0 则作用于所有频道(channel)

pan: 声道, 值: -1.0(左声道)到 1.0(右声道)

脚本语法:

```
setPan(channel, -1);
```

### **setVolume:**

```
void setVolume(int channel, double volume);
```

功能: 设置声音或音乐的音量

参数:

channel: 频道号 (0 为所有声音频道, 1 为音乐频道)

volume: 0.0 到 1.0

脚本语法:

```
setVolume(channel, .5);
```

**stopSound:**

```
void stopSound(int channel);
```

功能：在指定频道上停止播放音乐或声音文件

参数：

channel：使用 0 则停止所有声音, 使用 1 则停止所有音乐

– stopSound(0) 停止 PlaySound、PlaySound2 函数

– stopSound(1) 停止 PlayMusic、PlayMusic2 函数

脚本语法：

```
stopSound(channel);
```

**GetMusicTotalTime:**

```
double GetMusicTotalTime();
```

说明：获取当前 PlayMusic 函数或 PlayMusic2 函数播放的音频总时间，单位为秒

**SetMusicPosition:**

```
int SetMusicPosition(double position);
```

说明：设置当前 PlayMusic 函数或 PlayMusic2 函数播放的音频播放位置

参数：

position：单位秒

**PauseMusic:**

```
void PauseMusic();
```

说明：暂停当前 PlayMusic 函数或 PlayMusic2 函数播放的音频

**ResumeMusic:**

```
void ResumeMusic();
```

说明：继续播放当前 PlayMusic 函数或 PlayMusic2 函数播放的音频

**RewindMusic:**



```
void RewindMusic();
```

说明：重新播放当前 PlayMusic 函数或 PlayMusic2 函数播放的音频

### **Recoder\_OnPlay:**

```
int Recoder_OnPlay(int state, const char* path);
```

功能：播放音频文件，主要是播放 Recoder\_OnRecorder 函数所录的音频

返回值 1 代表成功。0 代表可能没有关闭之前的播放声音，如果播放完成会自动关闭，关闭使用 Recoder\_OnPlay(0, "");

参数：

state: 0 停止播放录音，1 开始播放录音

path: 文件路径

### **Recoder\_OnRecorder:**

```
int Recoder_OnRecorder(int state, const char* path);
```

功能：录音函数（仿微信方式录音，即无需弹界面录音）。

返回值 0 代表录音失败，可能是可以关闭，关闭使用 Recoder\_OnRecorder(0, "test.amr"); 函数，1 代表录音打开成功。

参数：

state: 0 停止录音，1 开始录音

path: 文件路径

## **文件/变量的基本操作函数**

### **Var\_OpenFile:**

```
void* Var_OpenFile(const char *fileName, char *mode);
```

功能：打开一个 Var 文件。打开成功返回该文件的指针，打开失败返回 0，如果模式为 r+ 且没有该文件，也同样返回 0

参数：

fileName: 要打开的文件名

mode: 模式与 C 语言的模式相同，但需要注意的是，只能使用二进制文件的读写，即如果

读文件，需要“rb”，写文件需要“wb”，读写文件，且不覆盖原文件的模式为“rb+”，读写文件，并覆盖原文件的模式为“wb+”文件。

#### **Var\_CloseFile:**

```
int Var_CloseFile(void *fp);
```

功能： 关闭一个 Var 文件，关闭成功返回 1，关闭失败返回 0

参数：

fileName: 要关闭的文件名

#### **Var\_GetOneDimArraySize:**

```
int Var_GetOneDimArraySize(void *fp, char *variableName, int *arraySize);
```

功能：从文件中获得名为 variableName 的一维数组的大小。返回 1 代表读取成功，返回 0 代表读取失败。

参数：

fp: 文件的指针。

variableName: 要读取的变量名

arraySize: 接收读取到的数组的大小

#### **Var\_GetTwoDimArraySize:**

```
int Var_GetTwoDimArraySize(void *fp, char *variableName, int *rows, int *cols);
```

功能：从文件中获得名为 variableName 的二维数组的行数和列数。返回 1 代表读取成功，返回 0 代表读取失败。

参数：

fp: 文件的指针。

variableName: 要读取的变量名。

rowSize: 接收该数组的行数。

colSize: 接受该数组的列数。

### **文件/变量的读写函数**

#### **Var\_ReadData:**

```
int Var_ReadData(void *fp, char *variableName, void *acceptVariable, int dataFlag);
```

功能：从文件中读取名为 variableName 的整数，或是一维整型数组。返回 1 代表读取成功，返回 0 代表读取失败。

参数：

fp：文件的指针

variableName：要读取的变量名

acceptVariable：接收读取到的数据

dataFlag：值为 0 表示 Int 型， 1 表示 Double 型， 2 表示 String 型， 3 表示 Char 型

#### **Var\_ReadPart:**

```
int Var_ReadPart(void *fp, char *variableName, void *acceptVariable, int startIndex,  
int endIndex, int dataFlag);
```

功能：从文件中读取名为 variableName 的数组中的部分数据。返回 1 代表读取成功，返回 0 代表读取失败。

参数：

fp：文件的指针

variableName：要读取的变量名

acceptVariable：接收读取到的数据

startIndex：文件中变量的开始下标

endIndex：文件中变量的结束下标

dataFlag：值为 0 表示 Int 型， 1 表示 Double 型， 2 表示 String 型， 3 表示 Char 型

#### **Var\_WriteData:**

```
int Var_WriteData(void *fp, char *variableName, char *groupName, void *writeVariable,  
int dataRows, int dataCols, int spaceRows, int spaceCols, int dataFlag);
```

功能：从文件中写入名为 variableName，组为 groupName 的整数，或是一维整型数组。返回 1 代表写入成功，返回 0 代表写入失败。（会先删除掉文件中已有的变量）

参数：

fp：文件的指针

variableName：要写入的变量名

groupName: 变量的组名

writeVariable: 要写入的数据

dataRows: 要写入数据的行数。若为单个数据或一维数组, 那么值为 1

dataCols: 要写入数据的列数, 若为单个数据, 那么值等于 1, 如果为一维数组那么值大于 1。

spaceRows: 数据要开辟的行数。若为单个数据或一维数组, 那么值为 1

spaceCols: 数据要开辟的列数。若为单个数据, 那么值等于 1, 如果为一维数组那么值大于 1

dataFlag: 值为 0 表示 Int 型, 1 表示 Double 型, 2 表示 String 型, 3 表示 Char 型

#### **Var\_WritePart:**

```
int Var_WritePart(void *fp, char *variableName, char *groupName, void *writeVariable,  
int startIndex, int endIndex, int dataFlag);
```

功能: 从文件中写入名为 variableName, 组为 groupName 的数组中的部分数据。返回 1 代表写入成功, 返回 0 代表写入失败。

参数:

fp: 文件的指针

variableName: 要写入的变量名

groupName: 变量的组名

writeVariable: 要写入的数据

startIndex: 文件中变量的开始下标

endIndex: 文件中变量的结束下标

dataFlag: 值为 0 表示 Int 型, 1 表示 Double 型, 2 表示 String 型, 3 表示 Char 型

#### **文件/变量的删除函数**

##### **Var\_DelateVariable:**

```
int Var_DelateVariable(void *fp, char *variableName);
```

功能: 从文件中删除变量 variableName, 返回 1 代表删除成功, 返回 0 代表删除失败。

参数:

fp: 文件的指针

variableName: 要删除的变量

#### Var\_DelateFile:

```
int Var_DelateFile(const char *FileName);
```

功能: 删除该文件。返回 1 代表删除成功, 返回 0 代表删除失败

参数:

FileName: 要删除的文件名

### 文件/变量保存函数

使用 saveVars 和 loadVars 保存和读取游戏内的任意变量, 比如最高分, 当前生命值等。

要想使用这些函数, 必须在“自定义变量”面板里使用“保存到指定组”选项。

#### saveVars:

```
void saveVars(char *file, char *group)
```

功能: 保存所有变量组里的变量到指定文件。文件将被保存在游戏目录。

脚本语法:

```
saveVars("game.sav", "High Score");
```

#### loadVars:

```
void loadVars(char *file, char *group);
```

功能: 从指定文件读取变量组里的所有变量。

脚本语法:

```
loadVars("game.sav", "High Score");
```

#### 示例:

1. 创建"score"变量并放进"High Score"组里
2. 创建两个变量:"lives"和"energy"。把这些变量放进"Actor State"组里

当角色死亡时, 使用 saveVars("game.sav", "High Score");在不保存角色状态的前提下保存玩家当前的最高分

当用户退出游戏时, 使用 saveVars("game.sav", "Actor State");保存角色状态(生命和能量)

注意: 不同的变量组可以保存在同一个文件(game.sav)。最终, 在编辑器适当的位置

```
loadVars("game.sav", "High Score");和 loadVars("game.sav", "Actor State");
```

## SQLite 数据库

### SQLite\_Open:

```
SQLiteDB* SQLite_Open(const char* path);
```

功能：打开 SQLite 文件, 与 SQLite\_Close 函数配合使用，具体参照 SQLite 相关内容

参数：

path: 文件路径

脚本语法：

```
SQLiteDB *db = SQLite_Open("test.db");

//todo

SQLite_Close(db);
```

### SQLite\_Close:

```
void SQLite_Close(SQLiteDB *db);
```

功能：关闭 SQLite 文件, 与 SQLite\_Open 函数配合使用

参数：

db: 要关闭的文件名

脚本语法：

```
SQLiteDB *db = SQLite_Open("test.db");

//todo

SQLite_Close(db);
```

### SQLite\_ExecDML:

```
int SQLite_ExecDML(SQLiteDB *db, const char* szSQL);
```

功能：对数据库进行基本操作（表的增、删、修改等操作），具体参照 SQLite 相关内容

参数：

db: 文件的指针

szSQL: 操作命令

脚本语法：

```
int nRows=SQLite_ExecDML(db,"create table emp(empno int, empname char(20));");
```

**SQLite\_ExecQuery:**

```
SQLiteQuery* SQLite_ExecQuery(SQLiteDB *db, const char* szSQL);
```

功能: SQLite 查询操作, 具体参照 SQLite 相关内容

参数:

db: 文件的指针

szSQL: 操作命令

脚本语法:

```
SQLiteQuery *dq = SQLite_ExecQuery(db, "select * from emp");
```

**SQLite\_ExecScalar:**

```
int SQLite_ExecScalar(SQLiteDB *db, const char* szSQL);
```

功能: SQLite 查询操作, 具体参照 SQLite 相关内容

参数:

db: 文件的指针

szSQL: 操作命令

脚本语法:

```
int nRows=SQLite_ExecScalar(db,"select count(*) from emp");
```

**SQLite\_TableExists:**

```
int SQLite_TableExists(SQLiteDB *db, const char* szTable);
```

功能: 检测 db 所指向的数据库文件中列名称为 szTable 中的内容的列是否存在, 具体参照 SQLite 相关内容

存在返回 1, 不存在返回 0

参数:

db: 文件的指针

szTable: 字符串指针 (列的名称)

**SQLite\_Query\_numfield:**

```
int SQLite_Query_numfield(SQLiteQuery *query);
```

功能：获取当前数据库中 query 所指向的表的列数，具体参照 SQLite 相关内容

参数：

query：数据指针

**SQLite\_Query\_descfield:**

```
char* SQLite_Query_descfield(SQLiteQuery *query, int nField);
```

功能：返回 query 所指向的第 nField 列的名称及数据类型，具体参照 SQLite 相关内容

参数：

query：数据指针

nField：数据表的列号

脚本语法：

```
char str[256] = "";  
strcpy(str, SQLite_Query_descfield(dq, 1));
```

**SQLite\_Query\_nextRow:**

```
void SQLite_Query_nextRow(SQLiteQuery *query);
```

功能：指向下一行数据库数据，具体参照 SQLite 相关内容

参数：

query：数据指针

**SQLite\_Query\_eof:**

```
int SQLite_Query_eof(SQLiteQuery *query);
```

功能：查询当前 query 所指向的数据库数据是否是文件结束标志，具体参照 SQLite 相关内容  
是返回 1，否返回 0

参数：

query：数据指针

脚本语法：

```
while(!SQLite_Query_eof(dq))  
{
```



```
        //todo

        SQLite_Query_nextRow(dq);
    }
}
```

**SQLite\_Query\_getIntfieldByID:**

```
int SQLite_Query_getIntfieldByID(void *query, int nField);
```

功能: 返回当前 query 所指向的第 nField 列数据内容 (int 型), 具体参照 SQLite 相关内容

参数:

query: 数据指针

nField: 数据表的列号

**SQLite\_Query\_getIntfieldByName:**

```
int SQLite_Query_getIntfieldByName(SQLiteQuery *query, const char* fileName);
```

功能: 返回当前 query 所指向的列名称为 fileName 的数据内容 (int 型), 具体参照 SQLite 相关内容

参数:

query: 数据指针

fileName: 数据表的列的名称

**SQLite\_Query\_getDoublefieldByID:**

```
double SQLite_Query_getDoublefieldByID(SQLiteQuery *query, int nField);
```

功能: 返回当前 query 所指向的第 nField 列数据内容 (double 型), 具体参照 SQLite 相关内容

参数:

query: 数据指针

nField: 数据表的列号

**SQLite\_Query\_getDoublefieldByName:**

```
double SQLite_Query_getDoublefieldByName(SQLiteQuery *query, const char* fileName);
```

功能: 返回当前 query 所指向的列名称为 fileName 的数据内容 (double 型), 具体参照 SQLite

相关内容

参数:

query: 数据指针

fileName: 数据表的列的名称

#### SQLite\_Query\_getStringfieldByID:

```
void SQLite_Query_getStringfieldByID(SQLiteQuery *query, int nField, char* Rstr);
```

功能: 返回当前 query 所指向的第 nField 列数据内容 (String 型) 到 Rstr 所指向的字符串,

具体参照 SQLite 相关内容

参数:

query: 数据指针

nField: 数据表的列号

Rstr: 字符串指针

#### SQLite\_Query\_getStringfieldByName:

```
void SQLite_Query_getStringfieldByName(SQLiteQuery *query, const char* fieldName,  
char* Rstr);
```

功能: 返回当前 query 所指向的列名称为 fileName 的数据内容 (String 型) 到 Rstr 所指向的字符串, 具体参照 SQLite 相关内容

参数:

query: 数据指针

nField: 数据表的列号

Rstr: 字符串指针

#### SQLite 数据库示例:

```
int nRows;
```

```
int nfield;
```

```
int i;
```

```
int exist;
```

```
SQLiteQuery *dq;
```

```
char test1[256];

char test2[256];

SQLiteDB *db = SQLite_Open("test.db");

nRows=SQLite_ExecDML(db,"create table emp(empno int, empname char(20),ID float);");

nRows=SQLite_ExecDML(db,"begin transaction;");

nRows=SQLite_ExecDML(db,"insert into emp values (7, '甲方',20001);");

nRows=SQLite_ExecDML(db,"insert into emp values (9, '乙方',20002);");

nRows=SQLite_ExecDML(db,"insert into emp values (11, '丙方',20003);");

SQLite_ExecDML(db,"commit transaction;");


nRows=SQLite_ExecScalar(db,"select count(*) from emp");

logoutf("nRows:%d",nRows);

//检测表是否存在

exist = SQLite_TableExists(db, "emp");

logoutf("exist:%d",exist);


dq = SQLite_ExecQuery(db, "select * from emp");

if(dq != NULL)

{

    nfield= SQLite_Query_numfield(dq);

    for(i = 0; i< nfield; i++)

    {

        //获取列数据名称及数据类型

        strcpy(test1,SQLite_Query_descfield(dq, i));

        logoutf("%s \n",test1);

    }

}


while(!SQLite_Query_eof(dq))
```

```
{  
    //根据名称获取当前行的整型数据  
    int nint = SQLite_Query_getIntfieldByName(dq, "Mapno");  
    //根据列号获取当前行的整型数据  
    int idint = SQLite_Query_getIntfieldByID(dq, 0);  
    //根据名称获取当前行的浮点数据  
    double ac = SQLite_Query_getDoublefieldByName(dq, "ID");  
    //根据列号获取当前行的浮点数据  
    double idreal = SQLite_Query_getDoublefieldByID(dq, 0);  
    //根据名称获取当前行的字符串数据  
    SQLite_Query_getStringfieldByName(dq, "Mapname", test1);  
    //根据列号获取当前行的字符串数据  
    SQLite_Query_getStringfieldByID(dq, 1, test2);  
    logoutf("%d %d %lf %lf %s %s", nint, idint, ac, idreal, test1, test2);  
  
    //跳到下一行  
    SQLite_Query_nextRow(dq);  
}  
}  
SQLite_Close(db);
```

## 游戏控制器

### LoadGame:

```
int LoadGame(const char* path);
```

功能： 载入游戏，同时可以载入 mcd 文件和导出的 dat 文件

返回值 1 成功 0 失败

参数：

Path: 游戏路径,

例如：

想载入 test.mcd

```
LoadGame(“test.mcd”); //支持相对路径和绝对路径。
```

或者是载入导出的 dat 文件，载入的是导出时候的文件名称，比如是 test.dat，

```
LoadGame(“test.dat”); //支持相对路径和绝对路径。
```

### **ExitGame:**

```
int ExitGame();
```

功能：结束游戏并返回系统

成功返回 1，否则返回 0

脚本语法：

```
ExitGame();
```

### **SuspendGame:**

```
void SuspendGame();
```

功能：暂停游戏并停止接收任何事件

脚本语法：

```
SuspendGame();
```

在掌上电脑上显示任务栏。

当游戏获得焦点时游戏继续(在掌上电脑，手提电脑和智能手机上，当用户点击标题栏窗口或按下 Alt+Tab 或用户点击[Continue]选项)

### **PauseGameOff:**

```
void PauseGameOff();
```

功能：当游戏在 PauseGameOn() 函数作用下暂停后，此函数可继续游戏

下面是一个使用 PauseGameOn 后的例子

脚本语法/示例代码：

1) 当用户要暂停游戏时，创建你的” pauseActor”：

```
“Paused - Click To Resume”
```

2) 在 Create Actor 事件的” pauseActor” 里添加如下代码至脚本编辑器行为：

```
PauseGameOn();
```

3) 在 Mouse Button Down 事件里, 添加如下代码至脚本编辑器行为:

```
PauseGameOff();

DestroyActor("Event Actor");
```

#### PauseGameOn:

```
void PauseGameOn();
```

功能: 暂停游戏但继续接收键盘和鼠标事件。该行为让你可以在键盘或鼠标事件里调用

PauseGameOff()。

脚本语法:

```
PauseGameOn();
```

#### 键盘函数

##### getKeyText:

```
char *getKeyText(int key);
```

功能: 返回按键种类

脚本语法:

```
int keyCode = getLastKey(); //获得最后按下的按键

strcpy(myactor.text, getKeyText(keyCode)); //获取按键文本(getKeyText)并复制
(strcpy) 到 myactor
```

##### getLastKey:

```
int getLastKey();
```

功能: 返回最后按下的按键

脚本语法:

```
int selectedKey = getLastKey(); // "selectedKey"是 getLastKey 函数中用来储存最后
一个按下的按键的变量, 任何变量名都能用
```

注意: getLastKet() 函数返回的是重新映射后的最后一个按下的按键。

所以, 如果你按下 T 按键并且在之前把 T 映射到右方向键, 则函数返回的是右方向键。

为了避免这种状况, 在此例中, 你必须用如下方法删除映射:

```
remapKey(KEY_t, KEY_t);
```

##### remapKey:

```
void remapKey(int fromKey, int toKey);
```

功能：重定向 fromKey 到 toKey（用户配置按键）。用此函数可重定向任意按键

当使用 LoadGame() 后所有的重定向都会被删除

脚本语法：

例如：

一个角色用 KEY\_RCTRL 按键射击

如果你想让你的用户能够自己改变按键设置，就要使用 KeyDown 并添加如下代码进脚本编辑器：

```
remapKey(getLastKey(), KEY_RCTRL); // KEY_RCTRL 是需要重新映射的按键
```

现在，当用户按下按键将会重定向 KEY\_CTRL。remapKey() 函数允许你保存默认按键

### GetKeyState:

```
char *GetKeyState();
```

功能：返回键盘状态的数组

脚本语法：

```
char *key = GetKeyState(); // "key" 用来储存 GetKeyState() 函数。可以是任意名字
```

示例：

在 "Draw Actor" 事件里当左或右按键被按下时，以下代码将会向右或向左移动角色

- 1) 创建一个角色
- 2) 添加 "Draw actor" 事件
- 3) 添加脚本代码

代码：

```
char *key = GetKeyState(); //获取键盘状态

if(key[KEY_RIGHT] == 1) //测试是否按了“右方向键”
{
    x = x + 5; //把角色右移 5 个像素
}

if(key[KEY_LEFT] == 1) //测试是否按了“左方向键”
{
    x = x - 5; //把角色左移 5 个像素
}
```

有效的按键记录:

KEY\_BACKSPACE

KEY\_TAB

KEY\_CLEAR

KEY\_RETURN

KEY\_PAUSE

KEY\_ESCAPE

KEY\_SPACE

KEY\_EXCLAIM

KEY\_QUOTEDBL

KEY\_HASH

KEY\_DOLLAR

KEY\_AMPERSAND

KEY\_QUOTE

KEY\_LEFTPAREN

KEY\_RIGHTPAREN

KEY\_ASTERISK

KEY\_PLUS

KEY\_COMMA

KEY\_MINUS

KEY\_PERIOD

KEY\_SLASH

KEY\_0

KEY\_1

KEY\_2

KEY\_3

KEY\_4

KEY\_5

KEY\_6

KEY\_7



KEY\_8

KEY\_9

KEY\_COLON

KEY\_SEMICOLON

KEY\_LESS

KEY\_EQUALS

KEY\_GREATER

KEY\_QUESTION

KEY\_AT

KEY\_LEFTBRACKET

KEY\_BACKSLASH

KEY\_RIGHTBRACKET

KEY\_CARET

KEY\_UNDERSCORE

KEY\_BACKQUOTE

KEY\_a

KEY\_b

KEY\_c

KEY\_d

KEY\_e

KEY\_f

KEY\_g

KEY\_h

KEY\_i

KEY\_j

KEY\_k

KEY\_l

KEY\_m

KEY\_n

KEY\_o

KEY\_p

KEY\_q

KEY\_r

KEY\_s

KEY\_t

KEY\_u

KEY\_v

KEY\_w

KEY\_x

KEY\_y

KEY\_z

KEY\_PAD\_0

KEY\_PAD\_1

KEY\_PAD\_2

KEY\_PAD\_3

KEY\_PAD\_4

KEY\_PAD\_5

KEY\_PAD\_6

KEY\_PAD\_7

KEY\_PAD\_8

KEY\_PAD\_9

KEY\_PAD\_PERIOD

KEY\_PAD\_DIVIDE

KEY\_PAD\_MULTIPLY

KEY\_PAD\_MINUS

KEY\_PAD\_PLUS

KEY\_PAD\_ENTER

KEY\_PAD\_EQUALS

KEY\_UP

KEY\_DOWN

KEY\_RIGHT

KEY\_LEFT

KEY\_INSERT

KEY\_HOME

KEY\_END

KEY\_PAGEUP

KEY\_PAGEDOWN

KEY\_F1

KEY\_F2

KEY\_F3

KEY\_F4

KEY\_F5

KEY\_F6

KEY\_F7

KEY\_F8

KEY\_F9

KEY\_F10

KEY\_F11

KEY\_F12

KEY\_F13

KEY\_F14

KEY\_F15

KEY\_NUMLOCK

KEY\_CAPSLOCK

KEY\_SCROLLLOCK

KEY\_RSHIFT

KEY\_LSHIFT

KEY\_RCTRL

KEY\_LCTRL

KEY\_RALT

KEY\_LALT

KEY\_RMETA

KEY\_LMETA

KEY\_LWINDOWS

KEY\_RWINDOWS

KEY\_ALT\_GR

KEY\_HELP

KEY\_PRINT

KEY\_SYSREQ

KEY\_BREAK

KEY\_MENU

KEY\_MAC\_POWER

KEY\_EURO

KEY\_POCKET\_UP

KEY\_POCKET\_DOWN

KEY\_POCKET\_LEFT

KEY\_POCKET\_RIGHT

KEY\_POCKET\_A

KEY\_POCKET\_B

KEY\_POCKET\_C

KEY\_POCKET\_START

KEY\_POCKET\_AUX1

KEY\_POCKET\_AUX2

KEY\_POCKET\_AUX3

KEY\_POCKET\_AUX4

KEY\_POCKET\_AUX5

KEY\_POCKET\_AUX6

KEY\_POCKET\_AUX7

KEY\_POCKET\_AUX8

KEY\_GP2X\_BUTTON\_UP

KEY\_GP2X\_BUTTON\_DOWN  
KEY\_GP2X\_BUTTON\_LEFT  
KEY\_GP2X\_BUTTON\_RIGHT  
KEY\_GP2X\_BUTTON\_UPLEFT  
KEY\_GP2X\_BUTTON\_UPRIGHT  
KEY\_GP2X\_BUTTON\_DOWNLEFT  
KEY\_GP2X\_BUTTON\_DOWNRIGHT  
KEY\_GP2X\_BUTTON\_CLICK  
KEY\_GP2X\_BUTTON\_A  
KEY\_GP2X\_BUTTON\_B  
KEY\_GP2X\_BUTTON\_X  
KEY\_GP2X\_BUTTON\_Y  
KEY\_GP2X\_BUTTON\_L  
KEY\_GP2X\_BUTTON\_R  
KEY\_GP2X\_BUTTON\_START  
KEY\_GP2X\_BUTTON\_SELECT  
KEY\_GP2X\_BUTTON\_VOLUP  
KEY\_GP2X\_BUTTON\_VOLDOWN

**GetJoystick1Axis:**

```
int GetJoystick1Axis(int axis);
```

功能：获取手柄中方向按键在当前设备中的对应值

参数：

axis: 方向按钮

**GetJoystick1Button:**

```
int GetJoystick1Button(int num);
```

功能：获取手柄中按钮按键在当前设备中的对应值

参数：

num: 按钮

## 网络相关函数

（注：异步服务 ID 大于 5000 的会排到小于 5000 的异步任务前面）

### AsyncUpdateOnNetWork:

```
int AsyncUpdateOnNetWork(const char *gameName, int *response, int *dataSize);
```

功能：异步更新。从 FTP 服务器下载打包文件，并解压成文件夹。

参数：

gameName：下载的文件名

response：响应

dataSize：下载的文件大小

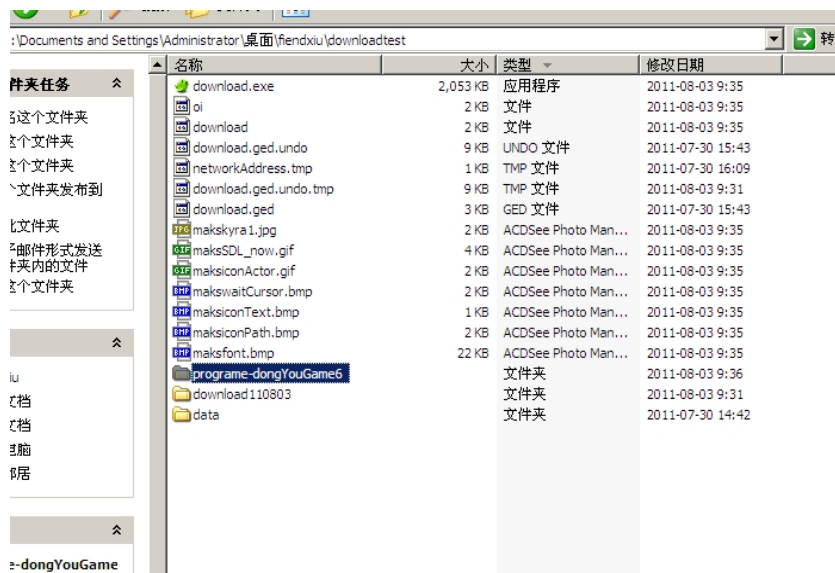
脚本语法：

```
int *size;
```

```
int a = 4791854;
```

```
size = &a;
```

```
AsyncUpdateOnNetWork("progame-dongYouGame6", 0, size);
```



### UpdateOnNetWork:

```
int UpdateOnNetWork(const char *gameName);
```

功能：修改服务的返回值，添加一个最新版本的返回值，通过当前版本和最新版本的比较，判断是否为最新版本。

参数：

gameName:更新包名

返回值:

100: 表示更新完成;

-1: 访问网成功, 但是以“wb”的方式打开文件失败;

-2: 网络访问失败。

#### **AsyncCallGsoapByJson:**

```
int AsyncCallGsoapByJson(char *parameters, char *format, char *serverName, const char *endPoint, int asyncNetId);
```

功能: 使用异步传输访问网络, 从服务器上将所需的数据通过 Webservice 服务传输并写到指定好的自定义变量里。

参数:

parameters: 参数

format: 参数格式

serverName: 服务名

endPoint: 服务器地址

asyncNetId: 异步网络 ID(用于识别)

#### **callGsoapByJson:**

```
int callGsoapByJson(char *parameters, char *format, char *serverName, const char *endPoint);
```

功能: 访问网络, 从服务器上将所需的数据通过 Webservice 服务传输并写到指定好的自定义变量里。

参数:

parameters: 参数

format: 参数格式

serverName: 服务名

endPoint: 服务器地址

#### **callGsoapService:**

```
int callGsoapService(char *parameters, char *format, char *serverName, void  
*resp_result, char *resp_format, int string_size);
```

功能：访问 Webservice 服务，并且将数据写进指定的自定义变量中。

参数：

parameters: 参数  
format: 参数格式  
serverName: 服务名  
resp\_result: 响应结果  
resp\_format: 响应格式  
string\_size: 字符串长度

#### **AsyncFtpUpload:**

```
int AsyncFtpUpload(const char * host, const char * userName, const char * pwd, const  
char * filePath, int asyncNetId);
```

功能：使用 ftp 服务异步上传，从服务器上将所需的数据通过 Webservice 服务传输并写到指定好的自定义变量里。

参数：

host: 服务器 IP 地址及端口，如 192.168.1.1:80  
userName: ftp 服务器用户名  
pwd: ftp 服务器密码  
filePath: 文件路径  
asyncNetId: 异步网络 ID(用于识别)

#### **FtpUpload:**

```
int FtpUpload(const char * host, const char * userName, const char * pwd, const char  
* filePath, const char * saveName);
```

功能：使用 ftp 服务上传文件

参数：

host: 服务器 IP 地址及端口，如 192.168.1.1:80  
userName: ftp 服务器用户名



pwd: ftp 服务器密码

filePath: 文件路径

saveName: 保存到 ftp 服务器上的文件名称

### **AsyncPostUpload:**

```
int AsyncPostUpload(const char *url, const char *key, const char *file, int  
asyncNetId);
```

功能: 异步文件上传服务

参数:

url: 上传地址

key: 关键字名称

file: 上传文件路径

asyncNetId: 异步 ID

### **PostUpload:**

```
int PostUpload(const char *url, const char *key, const char *file);
```

功能: 文件上传服务

参数:

url: 上传地址

key: 关键字名称

file: 上传文件路径

### **SendNetDataf:**

```
void SendNetDataf (const char * fmt, ...);
```

功能: 发送网络数据。输入格式可以参考 c 语言 printf 函数

脚本语法:

```
SendNetDataf ("%d,%d", xmouse, ymouse); 发送鼠标坐标
```

### **GetNetData:**

```
void GetNetData (char* data) ;
```

功能：接收网络数据。

参数：

data: 接收的数据的缓存。

**GetError:**

```
void GetError(char *data);
```

功能：获取网络通信中的错误，有错误则返回字符串。

参数：

data: 接收的数据的缓存。

**HttpPostSetUrl:**

```
int HttpPostSetUrl(const char * url);
```

功能：使用 HttpPost 服务前，设置发送的网络地址，必须和 HttpPostAddParam

(HttpPostAddParamBase64、HttpPostAddParamMd5) 及 HttpPostSend (AsyncHttpPostSend)

函数配合使用

参数：

url: 网络地址

**HttpPostAddParam:**

```
int HttpPostAddParam(const char *key, const char *value);
```

功能:添加HttpPost 服务的数据参数,与 HttpPostSetUrl 及 HttpPostSend(AsyncHttpPostSend)

配合使用, 同 HttpPostAddParamBase64 及 HttpPostAddParamMd5 函数

参数:

key: 关键字名称

value: 变量值

**HttpPostAddParamBase64:**

```
int HttpPostAddParamBase64(const char *key, const char *filePath);
```

功能：添加 HttpPost 服务的数据参数（Base64 加密，可传图片），与 HttpPostSetUrl 及

HttpPostSend (AsyncHttpPostSend) 配合使用, 同 HttpPostAddParam 及 HttpPostAddParamMd5 函数

参数:

key: 关键字名称

value: 文件路径

#### **HttpPostAddParamMd5:**

```
int HttpPostAddParamMd5(const char *key, const char *value);
```

功能: 添加 HttpPost 服务的数据参数 (md5 加密), 与 HttpPostSetUrl 及 HttpPostSend (AsyncHttpPostSend) 配合使用, 同 HttpPostAddParamBase64 及 HttpPostAddParam 函数

参数:

key: 关键字名称

value: 变量值

#### **HttpPostAddUploadFile:**

```
int HttpPostAddUploadFile(const char *key, const char *filePath);
```

功能: 添加 HttpPost 服务的数据参数 (将本地文件上传至服务器), 与 HttpPostSetUrl 及 HttpPostSend (AsyncHttpPostSend) 配合使用, 同 HttpPostAddParamBase64 及 HttpPostAddParam 函数

参数:

key: 关键字名称

filePath: 上传文件路径

#### **AsyncHttpPostSend:**

```
int AsyncHttpPostSend(int asyncNetId);
```

功能: 使用 Http 异步传输网络数据, 必须和 HttpPostSetUrl 及 HttpPostAddParam (HttpPostAddParamBase64、HttpPostAddParamMd5) 配合使用, 同 HttpPostSend 函数。

参数:

asyncNetId: 异步网络 ID(用于识别)

**HttpPostSend:**

```
int HttpPostSend();
```

功能：发送网络数据（同步），必须和 HttpPostSetUrl 及 HttpPostAddParam

（HttpPostAddParamBase64、HttpPostAddParamMd5）配合使用，同 AsyncHttpPostSend 函数（异步）。

**AsyncHttpDownload:**

```
int AsyncHttpDownload(const char * httpAddr, const char * saveName, int asyncNetId);
```

功能：使用 Http 异步下载文件

参数：

httpAddr: 下载文件的网址

saveName: 保存到本地的文件名

asyncNetId: 异步网络 ID(用于识别)

**HttpDownload:**

```
int HttpDownload(const char * httpAddr, const char * saveName);
```

功能：使用 Http 服务下载文件

参数：

httpAddr: 下载文件的网址

saveName: 保存到本地的文件名

**AsyncSendHttpRequest:**

```
int AsyncSendHttpRequest(const char *serverAddr, const char *request, int asyncNetId);
```

功能：使用异步传输功能接受网络数据，从服务器上将所需的数据通过 http 服务传输并写到指定好的自定义变量里。

参数：

serverAddr: 服务器网址首地址

request: 请求数据

asyncNetId: 异步网络 ID(用于识别)

**sendHttpRequest:**

```
int sendHttpRequest(const char *serverAddr, const char *request);
```

功能：接收网络数据，从服务器上将所需的数据通过 http 服务传输并写到指定好的自定义变量里。

参数：

serverAddr: 服务器网址首地址

request: 请求数据格式

**httpRequest:**

```
void httpRequest(const char *serverAddr, const char *request, char *response);
```

功能：接收网络数据，从服务器上将所需的数据通过 http 服务传输并复制到 response 所指向的数组。

参数：

serverAddr: 服务器网址首地址

request: 请求数据格式

response: 接收到的数据指针

**getAsyncNetId:**

```
int getAsyncNetId();
```

功能：在“异步调用结束”事件中使用，获取异步通行结束时服务的网络 ID

**getAsyncNetResponse:**

```
int getAsyncNetResponse();
```

功能：在“异步调用结束”事件中使用，获取异步通行结束时服务的响应

返回值：0 成功、其他值为失败

**getDoubleVar:**

```
double getDoubleVar(const char *key, int index);
```

功能：获取网络服务中关键字为 key 中网络数据的内容（必须在网络请求到数据时使用，不然

可能会出现异常错误)

参数:

key: 关键字名称

index: 数组下标, 一个变量则为 0

#### **getDoubleVarSize:**

```
int getDoubleVarSize(const char *key);
```

功能: 获取网络服务中关键字为 key 中网络数据的内容的数组大小

参数:

key: 关键字名称

#### **getIntVar:**

```
int getIntVar(const char *key, int index);
```

功能: 获取网络服务中关键字为 key 中网络数据的内容 (必须在网络请求到数据时使用, 不然可能会出现异常错误)

(注: 在自定义变量中不能声明的变量名称时使用, 如 r)

参数:

key: 关键字名称

index: 数组下标, 一个变量则为 0

#### **getIntVarSize:**

```
int getIntVarSize(const char *key);
```

功能: 获取网络服务中关键字为 key 中网络数据的内容的数组大小

参数:

key: 关键字名称

#### **getStringVar:**

```
void getStringVar(const char *key, char *str, int index);
```

功能: 将网络服务中关键字为 key 中网络数据的内容复制到 str 所指向的数组 (必须在网络请求到数据时使用, 不然可能会出现异常错误)

(注：在自定义变量中不能申明的变量名称时使用，如 r)

参数：

key: 关键字名称

str: 返回参数指针

index: 数组下标，一个变量则为 0

#### **getStringVarSize:**

```
int getStringVarSize(const char *key);
```

功能：获取网络服务中关键字为 key 中网络数据的内容的数组大小

参数：

key: 关键字名称

#### **GetDownloadFileLength:**

```
long GetDownloadFileLength(const char * httpAddr);
```

功能：获取下载地址 httpAddr 中文件的大小

参数：

httpAddr: 网络下载路径地址

#### **QueryDownloadInfo:**

```
int QueryDownloadInfo(const char * httpAddr, long *curLen, int *state);
```

功能：获取下载地址 httpAddr 中正在下载的文件的大小（无需使用网络），配合

AsyncHttpDownload 函数使用

参数：

httpAddr: 网络下载路径地址

curLen: 当前正在下载文件大小

state: 状态值（预留）

#### **GetDownLoadSpeed:**

```
double GetDownLoadSpeed(const char * httpAddr);
```

功能：获取当前下载地址 httpAddr 中正在下载的文件的下下载速度（单位 KB/S）（无需使用网络），

配合 AsyncHttpDownload 函数使用

参数:

httpAddr: 网络下载路径地址

#### **PauseAsyncHttpDownload:**

```
void PauseAsyncHttpDownload(const char * httpAddr);
```

功能: 暂停下载地址为 httpAddr 的 AsyncHttpDownload 函数的进程

参数:

httpAddr: 网络下载路径地址

#### **SendMsg:**

```
void SendMsg (const char* phoneNumber, const char* msgContext);
```

功能: 向手机上发短信。

参数:

phoneNumber: 目标手机的号码。

msgContext: 发送的短信的内容。

### **HTTP 对照变量列表网络函数**

#### **AddJsonVarTable:**

```
int AddJsonVarTable(const char *serviceName, const char *key, const char *value);
```

功能: 将声明网络服务标识名称serviceName的服务中的变量key中的值存放到自定义变量value中。

参数:

serviceName: 网络服务标识名称

key: http返回参数的关键字

value: 自定义变量中定义的变量

#### **sendHttpRequest3:**

```
int sendHttpRequest3(const char *serverAddr, const char *request, const char  
*serviceName);
```

功能: 接收网络数据, 从服务器上将所需的数据通过 http 服务传输并写到指定好的自定义变量



里。

参数:

serverAddr: 服务器网址首地址  
request: 请求数据格式  
serviceName: 网络服务标识名称

#### **AsyncSendHttpRequest5:**

```
int AsyncSendHttpRequest5(const char *cloneName, const char *serverAddr, const char  
*request, const char *serviceName, int asyncNetId);
```

功能: 使用异步传输功能接受网络数据, 从服务器上将所需的数据通过 http 服务传输并写到指定好的自定义变量里。

参数:

cloneName: 服务回调将要返回的角色克隆名 (调用 异步调用结束 事件)  
serverAddr: 服务器网址首地址  
request: 请求数据  
serviceName: 网络服务标识名称  
asyncNetId: 异步网络 ID(用于识别)

#### **HttpPostSend1:**

```
int HttpPostSend1(const char *serviceName);
```

功能: 发送网络数据 (同步), 必须和HttpPostSetUrl及HttpPostAddParam

(HttpPostAddParamBase64、HttpPostAddParamMd5) 配合使用, 同AsyncHttpPostSend函数 (异步)。

参数:

serviceName: 网络服务标识名称

#### **AsyncHttpPostSend3:**

```
int AsyncHttpPostSend3(const char *cloneName, const char *serviceName, int  
asyncNetId);
```

功能: 使用 Http 异步传输网络数据, 必须和 HttpPostSetUrl 及 HttpPostAddParam

(HttpPostAddParamBase64、HttpPostAddParamMd5) 配合使用, 同 HttpPostSend 函数。

参数:

cloneName: 服务回调将要返回的角色克隆名 (调用 异步调用结束 事件)

serviceName: 网络服务标识名称

asyncNetId: 异步网络 ID(用于识别)

## 物理世界函数

### phyCreateWorld:

```
int phyCreateWorld(double gravity_x, double gravity_y, intpositionY);
```

功能: 创建一个物理世界。

参数:

gravity\_x : x 轴方向的重力加速度, 正数向右, 负数向左。(double 类型数值)

gravity\_y : y 轴方向的重力加速度, 正数向下, 负数向上。(double 类型数值)

position : 设置水平线的位置。(int 类型数值)

### phyWorldStep:

```
void phyWorldStep(double timeStep, intvelocityIterations, intpositionIterations);
```

功能: 模拟物理世界。

参数:

timeStep : 时间一般都是 1.0 除以帧数, 一般是 60 帧。

velocityIterations : 速度迭代, 迭代次数越多, 速度就越接近现实, 但是对性能要求就越高, 一般设置成 8。

positionIterations : 位置迭代, 迭代次数越多, 位置就越接近现实, 但是对性能要求就越高, 一般设置成 3。

### phyActorUpdate:

```
void phyActorUpdate();
```

功能: 更新物理物体状态。

无参数, 无返回值。

**phyIsAwake:**

```
int phyIsAwake();
```

功能：判断物理物体是否停下来。

返回值：

0: 为真，物体停下。

1: 为假，物体还在运动中。

**createRopeJoint:**

```
int createRopeJoint(const char *cloneNameA, const char *cloneNameB, const char  
*ropeCloneName, int ropeSize);
```

功能：将角色 cloneNameA 和角色 cloneNameB 用绳子连起来，绳子用 ropeCloneName 的角色来体现，与 CutRope 函数配合使用

参数

cloneNameA: 角色名

cloneNameB: 角色名

ropeCloneName: 绳子角色名

ropeSize: 绳子的长度（预留接口）

脚本语法：

```
createRopeJoint(bodyA.clonename, bodyB.clonename, rope.clonename, 0);
```

**CutRope:**

```
int CutRope(const char *cloneName);
```

功能：将物理事件中的绳子切断，与 createRopeJoint 函数配合使用

参数：

cloneName: 角色克隆名

**phyCreateAutoDynamicBody:**

```
void phyCreateAutoDynamicBody(const char *actorName, float density, float friction,  
float restitution);
```

功能：

将指定的对象创建成物理世界的动态物体，会根据图片的形状，自动创建出符合图片

形状的物理物体。

参数：

actorName:对象名

Density: 物体的密度

Friction: 物体的摩擦力

Restitution: 物体的弹力系数

#### **phyCreateAutoStaticBody:**

```
void phyCreateAutoStaticBody(const char *actorName, float density, float friction, float restitution);
```

功能：

将指定的对象创建成物理世界的静态物体，会根据图片的形状，自动创建出符合图片形状的物理物体。

参数：

actorName: 对象名

Density: 物体的密度

Friction: 物体的摩擦力

Restitution: 物体的弹力系数

#### **phyCreateDynamicBody:**

```
void phyCreateDynamicBody(double density, double friction, double restitution, double linearDamping, double angularDamping);
```

功能：创建矩形动态物理物体

参数：

density : 设置物体的密度，以 kg/m<sup>3</sup> 为单位。(double 类型数值)

friction : 设置物体的摩擦系数，范围[0.0-1.0]。(double 类型数值)

restitution : 设置物体的弹力系数，范围[0.0-1.0]。(double 类型数值)

linearDamping : 设置物体的空气阻力，范围[0.0-1.0]。(double 类型数值)

angularDamping : 设置物体的旋转阻力，范围[0.0-1.0]。(double 类型数值)

#### **phyCreateStaticBody:**

```
void phyCreateStaticBody(double density, double friction, double restitution, double linearDamping, double angularDamping);
```

功能：创建矩形静态物理物体

参数：

density : 设置物体的密度，以 kg/m<sup>3</sup> 为单位。(double 类型数值)

friction : 设置物体的摩擦系数，范围[0.0-1.0]。(double 类型数值)

restitution : 设置物体的弹力系数，范围[0.0-1.0]。(double 类型数值)

linearDamping : 设置物体的空气阻力，范围[0.0-1.0]。(double 类型数值)

angularDamping : 设置物体的旋转阻力，范围[0.0-1.0]。(double 类型数值)

#### **phyCreateDynamicCircle:**

```
void phyCreateDynamicCircle(double density, double friction, double restitution, double linearDamping, double angularDamping);
```

功能：创建圆形动态物理物体

参数：

density : 设置物体的密度，以 kg/m<sup>3</sup> 为单位。(double 类型数值)

friction : 设置物体的摩擦系数，范围[0.0-1.0]。(double 类型数值)

restitution : 设置物体的弹力系数，范围[0.0-1.0]。(double 类型数值)

linearDamping : 设置物体的空气阻力，范围[0.0-1.0]。(double 类型数值)

angularDamping : 设置物体的旋转阻力，范围[0.0-1.0]。(double 类型数值)

#### **phyCreateStaticCircle:**

```
void phyCreateStaticCircle(double density, double friction, double restitution, double linearDamping, double angularDamping);
```

功能：创建圆形静态物理物体

参数：

density : 设置物体的密度，以 kg/m<sup>3</sup> 为单位。(double 类型数值)

friction : 设置物体的摩擦系数，范围[0.0-1.0]。(double 类型数值)

restitution : 设置物体的弹力系数，范围[0.0-1.0]。(double 类型数值)

linearDamping : 设置物体的空气阻力，范围[0.0-1.0]。(double 类型数值)

angularDamping : 设置物体的旋转阻力, 范围[0.0-1.0]。(double 类型数值)

### phyCreateDynamicPolygon3:

```
void phyCreateDynamicPolygon3(double *xarray, double *yarray, int arraySize);
```

功能: 创建动态多边形物理物体 (最多支持 8 个点), 同 phyCreateDynamicPolygon8 函数但物理属性为默认值

参数:

xarray: x 数组坐标

yarray: y 数组坐标

arraySize: 实际点的个数

脚本语法:

```
//三角形变量
double xarray[3];
double yarray[3];
//三个顶点的坐标
xarray[0] = -24;
yarray[0] = 24;
xarray[1] = -24;
yarray[1] = -24;
xarray[2] = 24;
yarray[2] = 24;
//创建三角形动态物体
phyCreateDynamicPolygon3(xarray, yarray, 3);
```

### phyCreateDynamicPolygon8:

```
void phyCreateDynamicPolygon8(double *xarray, double *yarray, int arraySize, double
density, double friction, double restitution, double linearDamping, double
angularDamping);
```

功能: 创建动态多边形物理物体 (最多支持 8 个点) 并设置物理属性, 同 phyCreateDynamicPolygon3 函数但物理属性可以自己设定

参数:

xarray: x 数组坐标

yarray: y 数组坐标

arraySize: 实际点的个数

density : 设置物体的密度, 以 kg/m<sup>3</sup> 为单位。(double 类型数值)

friction : 设置物体的摩擦系数, 范围[0.0-1.0]。(double 类型数值)

restitution : 设置物体的弹力系数, 范围[0.0-1.0]。(double 类型数值)

linearDamping : 设置物体的空气阻力, 范围[0.0-1.0]。(double 类型数值)

angularDamping : 设置物体的旋转阻力, 范围[0.0-1.0]。(double 类型数值)

脚本语法:

```
//三角形变量
```

```
double xarray[3];
```

```
double yarray[3];
```

```
//三个顶点的坐标
```

```
xarray[0] = -24;
```

```
yarray[0] = 24;
```

```
xarray[1] = -24;
```

```
yarray[1] = -24;
```

```
xarray[2] = 24;
```

```
yarray[2] = 24;
```

```
//创建三角形动态物体
```

```
phyCreateDynamicPolygon8(xarray, yarray, 3, 10, 0, 0, 0, 0);
```

### **phyCreateStaticPolygon3:**

```
void phyCreateStaticPolygon3(double *xarray, double *yarray, int arraySize);
```

功能: 创建静态多边形物理物体 (最多支持 8 个点), 同 phyCreateStaticPolygon8 函数但物理属性为默认值

参数:

xarray: x 数组坐标

yarray: y 数组坐标

arraySize: 实际点的个数

脚本语法:

```
//三角形变量
double xarray[3];
double yarray[3];
//三个顶点的坐标
xarray[0] = -24;
yarray[0] = 24;
xarray[1] = -24;
yarray[1] = -24;
xarray[2] = 24;
yarray[2] = 24;
//创建三角形静态物体
phyCreateStaticPolygon3(xarray, yarray, 3);
```

#### **phyCreateStaticPolygon8:**

```
void phyCreateStaticPolygon8(double *xarray, double *yarray, int arraySize, double
density, double friction, double restitution, double linearDamping, double
angularDamping);
```

功能: 创建静态多边形物理物体 (最多支持 8 个点) 并设置物理属性, 同

phyCreateStaticPolygon3 函数但物理属性可以自己设定

参数:

xarray: x 数组坐标

yarray: y 数组坐标

arraySize: 实际点的个数

density : 设置物体的密度, 以 kg/m<sup>3</sup> 为单位。(double 类型数值)

friction : 设置物体的摩擦系数, 范围[0.0-1.0]。(double 类型数值)

restitution : 设置物体的弹力系数, 范围[0.0-1.0]。(double 类型数值)

linearDamping : 设置物体的空气阻力, 范围[0.0-1.0]。(double 类型数值)

angularDamping : 设置物体的旋转阻力, 范围[0.0-1.0]。(double 类型数值)



脚本语法:

```
//三角形变量
double xarray[3];
double yarray[3];
//三个顶点的坐标
xarray[0] = -24;
yarray[0] = 24;
xarray[1] = -24;
yarray[1] = -24;
xarray[2] = 24;
yarray[2] = 24;
//创建三角形静态物体
phyCreateStaticPolygon8(xarray, yarray, 3, 10, 0, 0, 0, 0);
```

#### **getcontactActor:**

```
Actor *getcontactActor();
```

功能: 获取与当前物理物体角色相接触的物理物体指针, 配合“物体接触”事件使用

#### **phyExplosions:**

```
int phyExplosions(double x, double y);
```

功能: 将一个物理世界的物体变成碎片

参数:

x:x 坐标值

y:y 坐标值

xy 坐标确定一个点, 从这个点将物体切成碎片

#### **phyGetDensity:**

```
double phyGetDensity(const char *actorName);
```

功能: 获取角色名为 actorName 的物理物体的密度

参数:

actorName: 角色名

#### **phyGetFriction:**

```
double phyGetFriction(const char *actorName);
```

功能: 获取角色名为 actorName 的物理物体的摩擦系数

参数:

actorName: 角色名

#### **phyGetMass:**

```
double phyGetMass(const char *actorName);
```

功能: 获取角色名为 actorName 的物理物体的质量

参数:

actorName: 角色名

#### **phyGetRestitution:**

```
double phyGetRestitution(const char *actorName);
```

功能: 获取角色名为 actorName 的物理物体的弹力系数

参数:

actorName: 角色名

#### **phySetLinearVelocity:**

```
void phySetLinearVelocity(float xVelocity, float yVelocity);
```

功能: 设置当前物理物体的初始速度

参数:

xVelocity: x 轴的速度, >0 代表向右的速度, <0 代表向左的速度。(float 类型数值)

yVelocity: y 轴的速度, >0 代表向下的速度, <0 代表向上的速度。(float 类型数值)

#### **phyGetLinearVelocityX:**

```
double phyGetLinearVelocityX(const char *actorName);
```

功能: 获取角色名为 actorName 的物理物体的 x 轴上的线速度

参数:

actorName: 角色名

#### **phyGetLinearVelocityY:**

```
double phyGetLinearVelocityY(const char *actorName);
```

功能: 获取角色名为 actorName 的物理物体的 y 轴上的线速度

参数:

actorName: 角色名

#### **手势支持函数**

##### **getOnFlingVXY:**

```
int getOnFlingVXY (float *x, float *y) ;
```

功能: 用户按下触摸屏、快速移动后松开, 返回值存入这里的\*x, \*y, 代表的是手指离开屏幕时候的瞬时速度。

脚本语法:

```
float i, j;  
getOnFlingVXY (&i, &j) ;  
sprintf (text, "%.6f, %.6f", i, j) ;
```

##### **getOnScrollDXY:**

```
int getOnScrollDXY (float *x, float *y) ;
```

功能: 用户拖动屏幕的时候, 返回值 x, y 是与上一帧的坐标的差值, 即位移。

脚本语法:

```
float i, j;  
getOnScrollDXY (&i, &j) ;  
sprintf (text, "%.6f, %.6f", i, j) ;
```

##### **getPressType:**

```
void getPressType(int * pressType);
```

功能: 获得按下屏幕的类型。

参数:

pressType: 用于存储按键类型, 0 表示按下不放, 1 表示长按, 2 表示短按

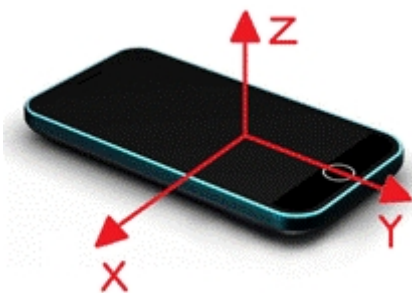
脚本语法:

```
int pressType;
getPressType(&pressType);
if(pressType == 0)
{
    sprintf(txt1.text, "OnDown");
}
else if(pressType == 1)
{
    sprintf(txt1.text, " LongDown");
}
else if(pressType == 2)
{
    sprintf(txt1.text, " ShortDown");
}
```

**getAccel:**

```
void getAccel(float *x, float *y, float *z);
```

功能: 检测手机的放置状态, 原理是将轴加速计三个轴的参数存入这里的  $x, y, z$ 。这三个参数构成的方向向量  $(x, y, z)$  指示的就是相对于地平面的手机的方向向量, 其坐标系如下图所示



手机的方然后根据  $x, y, z$  三个参数来判断手机的放置状态当手机处于缓慢移动的时候,  $x, y, z$  的范围是 $-10 \sim 0$ 、 $0 \sim 10$ 。这个时候, 其他外力大小可以忽略不计, 那么各个分量方向的加速度大小都不会大于重力加速度。而当我们疯狂甩动手机的时候, 手机受到除了重力以外的外力, 它的分量方向加速度就可能会大于 10。然后我们来分析手机的放置状态: ①当  $x=y=0$  时, 手机处于水平放置状态。②当  $x=0$  并且  $y>0$  时, 手机顶部的水平位置要大于底部, 也就是一般接听电

话时手机所处的状态。③当  $x=0$  并且  $y<0$  时，手机顶部的水平位置要小于底部。手机一般很少处于这种状态。④当  $y=0$  并且  $x>0$  时，手机右侧的水平位置要大于左侧，也就是右侧被抬起。⑤当  $y=0$  并且  $x<0$  时，手机右侧的水平位置要小于左侧，也就是左侧被抬起。⑥当  $z=0$  时，手机平面与水平面垂直。⑦当  $z>0$  时，手机屏幕朝上。⑧当  $z<0$  时，手机屏幕朝下。然而，上面的数据那只是理想状态下的情况，也就是说，手机水平放置的时候可以很准确的让  $x, y$  等于 0。可是实际情况下，手机的感应器并没有那么精确，它会有一点点误差，所以实际编程中，我们可以将水平放置情况下的取值范围适当扩大一点，一般情况下，判断手机水平放置的取值范围应当扩大到 -0.2 到 0.2

## 输入框函数

### GetFocus:

```
void GetFocus(const char * _cloneName);
```

功能：输入框获得光标

参数：

    \_cloneName: 输入框角色的克隆名

### LostFocus:

```
void LostFocus(const char * _cloneName);
```

功能：输入框失去光标

参数：

    \_cloneName: 输入框角色的克隆名

### GetLastFocus:

```
void GetLastFocus(char * _cloneName);
```

功能：获得最后一个失去光标的输入框名称

参数：

    \_cloneName: 返回的角色名称

## 运营充值函数

### Android\_BuyDongYouBi:

```
int Android_BuyDongYouBi(int count);
```

功能：用于用户购买动友币，暂时只能用于安卓系统。一般与 callGsoapByJson 函数一起使用。

参数:

count: 购买动友币的数量。

#### **Android\_UseDongYouBi:**

```
void Android_BuyDongYouBi(int count);
```

功能: 用于用户使用动友币, 暂时只能用于安卓系统。一般与 callGsoapByJson 函数一起使用。

参数:

count: 使用动友币的数量。

#### **Alipay:**

```
int Alipay(const char *product, const char* detail, const char* price);
```

说明: 调用支付宝进行支付

参数:

product: 物品名称

detail: 物品说明

price: 物品价格, 精确到: 0.01 元

返回值:

//9000 操作成功

//4000 系统异常

//4001 数据格式不正确

//4003 该用户绑定的支付宝账户被冻结或不允许支付

//4004 该用户已解除绑定

//4005 绑定失败或没有绑定

//4006 订单支付失败

//4010 重新绑定账户。

//6000 支付服务正在进行升级操作。

//6001 用户中途取消支付操作。

//6002 网络连接异常。

**EgameSMSCheckFee:**

```
int EgameSMSCheckFee(const char *feeName, const char* feecode, const char* tipInfo,  
const char* okinfo, int repeat);
```

说明：具体信息请参考论坛中的帖子说明

<http://www.dongyo.cn/bbs/forum.php?mod=viewthread&tid=3636>

参数说明：

feeName: 物件的名称。

Feecode : 物件的代码

Tipinfo: 确认的提示消息

Oninfo: 如果支付成功，弹出的信息

Repeat: 是否可以重复购买。(如果设置为 0: 购买一次后，再次玩就无需购买。如果设置为 1: 每一次使用该物件都需要重新购买)

返回值 int: 返回状态 (可以查看状态列表)

**EgameWebInit:**

```
void EgameWebInit(const char *gameid, const char *cpcode, const char *serviceCode,  
const char *goldResponseurl);
```

说明：

使用网络付费平台，必须在开始的时候初始化网络付费引擎，不然可能导致不可想象的后果。

具体信息请参考论坛中的帖子说明

<http://www.dongyo.cn/bbs/forum.php?mod=viewthread&tid=3636>

参数：

gameid 爱游戏平台提供的游戏 ID

cpcode 爱游戏平台提供的游戏 cpcode

serviceCode 爱游戏平台提供的游戏 serviceCode

goldResponseurl 获取金币的返回地址

以上 4 个参数目前无效，因为主配置文件编译后不可更改，所以以上的参数只是保留参数，如果要修改以上的参数，要提交到动友公司进行修改。

**EgameWebPayBySMS:**

```
Int EgameWebPayBySMS(int userID, int money);
```

说明:

通过短信的方式支付, 使用前必须初始化 EgameWebInit,

如果为非电信卡, 会直接转到充值界面进行充值后支持, 如果为电信卡用户, 可以直接通过短信支付。

具体信息请参考论坛中的帖子说明

<http://www.dongyo.cn/bbs/forum.php?mod=viewthread&tid=3636>

参数:

userID: 用户的 ID , 由用户自己定义 (1-268435455)。

money: 充值的钱 (单位元)

使用示例:

```
int result = EgameWebPayBySMS(123456, 2);  
sprintf(test.text, "bysms result = %d", result); //输出返回值
```

### **EgameWebPay:**

```
int EgameWebPay(int userID, int money);
```

说明:

通过弹出充值界面的方式进行充值钱币, 购买物品。使用前必须初始化 EgameWebInit。

具体信息请参考论坛中的帖子说明

<http://www.dongyo.cn/bbs/forum.php?mod=viewthread&tid=3636>

参数:

1.userID: 用户的 ID , 由用户自己定义 (1-268435455)。

2.money: 充值的钱 (单位元)

使用示例:

```
int result = EgameWebPay (123456, 2);  
sprintf(test.text, " result = %d", result); //输出返回值
```

### **部分绘制函数**

#### **SetDisplayRect:**

```
int SetDisplayRect(const char *cloneName, double x1, double y1, double x2, double y2);
```



说明：部分绘制函数，绘制角色以（x1, y1）为左上角、（x2, y2）为右下角的矩形区域，可用于绘制进度条等

参数：

cloneName: 被绘制角色克隆名

x1, y1: 左上角坐标

x2, y2: 右下角坐标

### **ClearDisplayRect:**

```
int ClearDisplayRect(const char *cloneName);
```

说明：清除部分绘制函数，清除 SetDisplayRect 绘制函数的绘制效果

参数：

cloneName: 被绘制角色克隆名

## **定位函数**

### **Location\_InitEngine:**

```
int Location_InitEngine(int scanTime, int pro, const char *format);
```

说明：启动定位引擎

参数：

scanTime: 扫描时间

pro: 优先级别，暂时为 0

format: 保留默认为空

返回值：

成功 1 否则 0

### **Location\_SwitchLocationState:**

```
void Location_SwitchLocationState()
```

说明：切换定位的开关，首次执行位打开定位，再次调用的时候关闭定位。

### **Location\_GetLocation:**

```
void Location_GetLocation(double *la, double *lo);
```

说明：获取经纬度。

参数：

la: 纬度

lo: 经度

## HTML 函数

### AddBrower:

```
void AddBrower(const char* url, int width, int height, int leftMargin, int topMargin, int  
rightMargin, int bottomMargin);
```

功能：调用系统网页浏览界面，可以支持调用本地 html 文件等

参数：

url: 网址或本地路径，调用本地文件时需加/，如："/traffic.html"

width: 设定网页的宽度，为 0 时自适应

height: 设定网页的高度，为 0 时自适应

leftMargin: 左边预留的空间

topMargin: 顶部预留的空间

rightMargin: 右边预留的空间

bottomMargin: 底部预留的空间

### RemoveBrower:

```
void RemoveBrower();
```

功能：移除已调用的网页浏览界面。

## 帮助函数

### ActorCount:

```
int ActorCount(const char *actorName);
```

功能：返回角色数量

参数：

actorName:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

脚本语法:

```
ActorCount("metalBlock1") == 1
```

### CollisionFree:

```
int CollisionFree(char *actorName, int x, int y);
```

功能: 检测(x,y)的位置是否和 "ActorName" 碰撞, 如果没有碰撞返回 1 否则返回 0.

参数:

Actor name:

- "Event Actor":接收当前事件的角色
- "Parent Actor": Event Actor 的父角色, 如果存在的话
- "Creator Actor": Event Actor 的创建者, 如果 Event Actor 是在"Create Actor"行为里被创建的话
- 游戏里的任意角色

脚本语法:

```
if (CollisionFree("Event Actor", x, y+5))  
{  
    y = y + 5;  
} //在 KEY_DOWN 的 KeyDown 事件脚本中, 检测在向下移动 5 个像素之前是否有碰撞。
```

### ConvertToDate:

```
void ConvertToDate(int intTime, int *tm_year, int *tm_mon, int *tm_mday, int *tm_hour,  
int *tm_min, int *tm_sec);
```

功能: 将 unix 时间戳转化为对应时间格式。

参数:

intTime: unix 时间戳

tm\_year: 年

tm\_mon: 月

tm\_mday: 日

tm\_hour: 时

tm\_min: 分

tm\_sec: 秒

使用方法:

```
ConvertToDate(intTime, &tm_year, &tm_mon, &tm_mday, &tm_hour, &tm_min, &tm_sec);
```

```
sprintf(Actor1.text, "%d年%d月%d日%d时%d分%d秒", tm_year, tm_mon, tm_mday, tm_hour,
tm_min, tm_sec);
```

### SendActivationEvent:

```
int SendActivationEvent(char *cloneName);
```

功能: 发送一个 Activation 事件给某个角色的克隆角色

参数:

cloneName: nameactor.cloneindex (比如: ship.1, ship.2, ...)

成功返回 1, 否则返回 0

脚本示例:

```
SendActivationEvent("ship.1"); //发送 Activation 事件给编号为 1 的克隆 ship 角色
```

### radtodeg:

```
double radtodeg(double angle);
```

功能: 把 angle 从弧度转换成角度

### degtorad:

```
double degtorad(double angle);
```

功能: 把 angle 从角度转换成弧度

脚本语法:

```
myactor.textNumber = degtorad(angle);
```

### replace\_all:

```
int replace_all(char *output, const char *input, const char *oldStr, const char
*newStr);
```

功能：将 input 所指向的数据中与 oldStr 所指向的数据匹配，如果与 oldStr 所指向的数据匹配则相应的匹配数据替换成 newStr 中的数据并将最后的数据复制给 output

参数：

output: 替换匹配数据后的字符串

input: 输入的字符串数据

oldStr: 要替换掉的数据

newStr: 新的数据

脚本语法：

```
char output[256] = "";
```

```
char input[256] = "[a]获得 158 金币, 5 经验";
```

```
replace_all(output, input, "[a]", "玩家 A");//执行后 output 的内容为"玩家 A 获得  
158 金币, 5 经验"
```

**direction:**

```
double direction(double x1, double y1, double x2, double y2);
```

功能：返回 (x1, y1) 到 (x2, y2) 的角度（沿 X 轴逆时针方向选择的角度）

脚本语法：

```
textNumber = direction(x, y, player.x, player.y);
```

**distance:**

```
double distance(double x1, double y1, double x2, double y2);
```

功能：返回 (x1, y1) (x2, y2) 两点的距离

脚本语法：

```
textNumber = distance(x, y, player.x, player.y);
```

**getAllActorsInCollision:**

```
Actor *getAllActorsInCollision(char *cloneName, int *nActors);
```

功能：检测所有角色是否和 "cloneName" 碰撞，返回角色数组。

参数：

nActors: 角色计算结果会被保存到 nActors。

cloneName: nameactor.cloneindex (比如: ship.1, ship.2, ...), "Event Actor",  
"Collide Actor", "Parent Actor" 或 "Creator Actor"

返回的数组在下次调用 `getAllActorsInCollision` 前都有效，返回的数组只读。

脚本语法：

```
getAllActorsInCollision("Event Actor",&num);
```

比如，让 "Event Actor" 作为所有碰撞角色的父角色：

```
int n;
Actor *actors;
actors = getAllActorsInCollision("Event Actor", &n);
if(actors)
{
    int i;
    for(i = 0; i < n; i++)
    {
        ChangeParent(actors[i].clonename, "Event Actor");
    }
}
```

#### **getAnimIndex:**

```
int getAnimIndex(char *animName);
```

功能：获取事件角色的动画序列

参数：

animName：当前角色的有效动画名。

成功返回动画序列，否则返回-1

脚本语法：

```
getAnimIndex("palette");
```

示例：

1. 创建两个角色："number" 和 "myImages"；
2. 分配文本给"number"角色，文本输入 0；
3. 分配 3 个动画给 "myImages" 角色；
4. 在"myImages"角色上，Create Actor >脚本编辑器；
5. 输入以下代码：

```
number.textNumber = getAnimIndex("myAnimation2");
```

**getActorAnimIndex:**

```
int getActorAnimIndex(const char *clonename, const char *animName);
```

功能: 获取角色特定动画的动画帧

返回: 特定动画帧的帧数

参数:

clonename: 角色克隆名

animIndex: 事件角色里的动画名。

**getAnimName:**

```
char *getAnimName(int animIndex);
```

功能: 获取事件角色的动画名

参数:

animIndex: 事件角色里的动画序列。

成功返回角色名, 否则返回"" 空

脚本语法:

```
strcpy(text, getAnimName(animindex));
```

**getActorAnimName:**

```
char *getActorAnimName(const char *clonename, int animIndex);
```

功能: 获取指定角色特定动画帧的动画名

返回: 角色动画名

参数:

clonename: 角色克隆名

animIndex: 事件角色里的动画序列。

**IsAnimationExists:**

```
int IsAnimationExists(const char * actorName, const char * aniName);
```

功能: 确认对象是否存在指定的对画

参数:

actorName: 对象名称

aniName:动画名称

返回值:

如果动画存在, 返回 1, 否则返回 0

**getactor:**

```
Actor *getactor(int x, int y);
```

功能: 返回(x,y)坐标(绝对坐标)上的角色

成功返回角色, 否则返回无效角色(cloneindex = -1 且 name = "")

脚本语法:

```
Actor *MyActor;
```

```
MyActor=getactor(x,y);
```

**getclone:**

```
Actor *getclone(char *cloneName);
```

功能: 返回指定克隆角色的指针, 允许你读取和改变克隆角色的各项变量, 而这通常是无法做到的。

参数:

cloneName: 克隆名 nameactor.cloneindex (比如: ship.1, ship.2, ...)

成功返回角色, 否则返回无效角色(cloneindex = -1 且 name = "")

**getTime:**

```
stTime getTime();
```

功能: 获取系统日期和时间

stTime 为如下结构体:

sec: Seconds after minute (0 - 59)

min: Minutes after hour (0 - 59)

hour: Hours since midnight (0 - 23)

mday: Day of month (1 - 31)

mon: Month (1 - 12; 一月 = 1)

year: Year (当前年份)



wday: Day of week (0 - 6; Sunday = 0)

yday: Day of year (0 - 365)

sec\_utc: Number of seconds elapsed since midnight (00:00:00), January 1, 1970  
(coordinated universal time)

脚本语法:

```
stTime t = getTime();
```

```
textNumber = t.sec; //显示秒数
```

### GetFileSize:

```
int GetFileSize(const char *file);
```

功能: 获取文件的文件大小。

参数:

file: 文件路径

返回值:

文件大小单位为 B (1GB=1024MB, 1MB=1024KB, 1KB=1024B)

当返回值为-1时表示文件不存在

### GetScreenWidth:

```
int GetScreenWidth();
```

功能: 获取屏幕的原本宽度, 这个一般是用来获取手机的屏幕分辨率。

返回值:

屏幕的原本宽度

### GetScreenHeight:

```
int GetScreenHeight();
```

功能: 获取屏幕的原本高度, 这个一般是用来获取手机的屏幕分辨率。

返回值

屏幕的原本高度

### setScreenScale:

```
void setScreenScale(double scale);
```

功能: 设置屏幕的缩放比例, 可与 getViewScale 函数配合使用

参数

scale: 缩放比例

**getViewScale:**

```
double getViewScale();
```

功能: 获取 view 框的缩放比例

**GetAngle:**

```
double GetAngle(const char *cloneName);
```

功能: 获取指定对象的旋转角度

参数:

cloneName: 对象的克隆名

返回值: 返回指定对象的旋转角度

**GetScale:**

```
double GetScale(const char *cloneName);
```

功能: 获取指定对象的缩放比例

参数:

cloneName: 对象的克隆名

返回值: 返回指定对象的缩放比例

**SetRelativeCoor:**

```
void SetRelativeCoor(const char* clonename, const char* parentname, int x, int y);
```

功能: 设置以(parentname)角色左上角为原点的坐标系, (clonename)角色的左上角相对这个坐标系原点的相对偏移量 x, y, 与 GetRelativeCoor 函数配合使用

参数:

clonename: 角色克隆名

parentname: 拿来当参照物的角色名

x, y: 相对于参照物左上角为原点的偏移量

**GetRelativeCoor:**

```
void GetRelativeCoor(const char* clonename, const char* parentname, int* x, int* y);
```

功能：获取以 (parentname) 角色左上角为原点的坐标系，(clonename) 角色的左上角相对这个坐标系原点的相对偏移量 x, y，与 SetRelativeCoor 函数配合使用

参数：

clonename: 角色克隆名

parentname: 拿来当参照物的角色名

x, y: 相对于参照物左上角为原点的偏移量

**CallKeyboard:**

```
void CallKeyboard(const char *dialogTitle, const char *TextTitle, char *showKeyBoard);
```

功能：用于控制虚拟键盘的呼出（单行输入和多行输入角色可以自动呼出虚拟键盘）

参数：

dialogTitle: 预留接口

TextTitle: 预留接口

showKeyBoard: 0 不呼出虚拟键盘，1 呼出虚拟键盘

**KeyboardHeight:**

```
int KeyboardHeight();
```

功能：当手持设备上弹出虚拟键盘时获取虚拟键盘的高度值

**KeyboardVisible:**

```
int KeyboardVisible();
```

功能：当手持设备上弹出虚拟键盘时获取虚拟键盘的状态值（0 为未呼出，1 为呼出）

**AddObserverActor:**

```
void AddObserverActor(const char *cloneName);
```

功能：将对象添加对观察列表中

参数：

cloneName: 对象的克隆名

**RemoveObserverActor:**

```
void RemoveObserverActor(const char *cloneName);
```

功能：将对象从观察列表中移除

参数：

cloneName: 对象的克隆名

**ClearObserverActor:**

```
void ClearObserverActor();
```

功能：清楚观察列表中的所有对象

**SetObserverRegion:**

```
void SetObserverRegion(int minX, int minY, int maxX, int maxY);
```

功能：

设置观察区域

参数：

minX: 左上角 X 坐标的值

minY: 左上角 Y 坐标的值

maxX: 右下角 X 坐标的值

maxY: 右下角 Y 坐标的值

**ShareFuns:**

```
void ShareFuns(const char *appkey, const char* content);
```

说明：实现分享功能，其中 android 上面使用的是 umeng 的，ios 使用的是 shareSDK 的分享集成包。

参数

appkey: umeng 或者 shareSDK 分享网站上所申请的 appkey

content: 要分享的内容

**AsyncUnzip:**

```
int AsyncUnzip(const char *unzipFile, const char *unzipDir, const char *clonename, int  
asyncNetId);
```

说明：实现文件异步解压的功能，两个路径都是相对于资源文件的目录（解压结束时会创建相应包名的.unzip 文件，字节数为 1 为成功，0 为失败）

参数：

unzipFile: zip 文件所在的路径

unzipDir: 解压的路径 如果是 "" 空字符串，则表示解压在资源文件目录下。

clonename: 事件回调将要返回的角色克隆名（调用 异步调用结束 事件）

asyncNetId: 异步 ID

返回值：

成功为 1 否则 0

### Unzip:

```
int unzip(const char *zipFile, const char *path);
```

说明：实现文件解压的功能，两个路径都是相对于资源文件的目录

参数：

zipFile: zip 文件所在的路径

path: 解压的路径 如果是 "" 空字符串，则表示解压在资源文件目录下。

返回值：

成功为 1 否则 0

### logout:

```
void logout(char *msg);
```

功能：将字符串和时间信息写入到 log.txt 文本文件中。

参数：

msg: 字符串。

脚本语法：

```
logout("123");
```

**logoutf:**

```
void logoutf(const char * fmt, ...);
```

功能：将字符串写入 log.txt 文本文件中。写入方法可以参考 sprintf()。

参数：

fmt: 字符串格式

脚本语法：

```
logoutf("view.x = %f, view.y = %f", view.x, view.y);
```

**openUrl:**

```
void openUrl(const char *url, int bExitGame);
```

功能：打开网页

参数：

Url: 网地

bExitGame: 打开网页时是否退出游戏

脚本语法：

```
openUrl("http://www.dongyo.cn", 0);
```

**round:**

```
double round(double arg);
```

功能：返回 arg 四舍五入后最接近的整数的值，无论怎样返回的数字都是浮点型。该值精确到两个值中间，比如 3.5 会被上调

**md5:**

```
void md5(const char * key, char * md5value);
```

功能：将 key 中的字符串内容经过 md5 算法加密后复制到 md5value 所指向的数组中

参数：

key: 要加密的内容

md5value: 加密后的内容

**max:**

```
double max(double a, double b);
```

功能：返回两者的最大值

**min:**

```
double min(double a, double b);
```

功能：返回两者的最小值

**rand:**

```
double rand (double max);
```

功能：产生一组伪随机数字，每次调用的时候都会返回 0 到最大值之间的数

**vectoradd:**

```
void vectoradd(double *angle1, double *magnitude1, double angle2, double magnitude2);
```

功能：添加向量 2(angle2, magnitude2) 给向量 1 (angle1, magnitude1) 并返回向量 1 里的结果

**设备鉴定函数****getHardwareID:**

```
char *getHardwareID();
```

功能：返回能鉴定设备的字符串，否则在桌面返回空字符串或者发生其他错误。

使用此函数获得的字符串可鉴定掌上电脑，手提电脑或智能手机设备，此字符串能被用在某些注册的服务来保护你的游戏不被盗版，某些受保护的智能手机要求你的游戏要签名，可用此函数。

脚本语法：

```
strcpy(text, getHardwareID());
```

**getOwner:**

```
char *getOwner();
```

功能：成功返回设备所有者名字，否则在桌面返回空字符串或者发生其他错误。

使用此函数返回掌上电脑，手提电脑或智能手机设备的拥有者名字。

脚本语法:

```
strcpy(text, getOwner());
```

#### **GetDeviceID:**

```
void GetDeviceID(char *DeviceID);
```

功能: 获取设备的 IMEI

#### **GetDeviceModel:**

```
void GetDeviceModel(char* outModel);
```

功能: 获取设备的型号

#### **GetDevicePhoneNumber:**

```
void GetDevicePhoneNumber(char* outPhoneNumber);
```

功能: 获取设备的手机号码, 无返回 0

#### **GetBarcode:**

```
void GetBarcode(char *str)
```

说明: 扫描二维码功能

参数:

str: 扫描成功后返回的结果字符串

#### **getSys:**

```
int getSys(void);
```

功能: 获取当前平台

返回值:

1: Android 系统

2: iOS 系统

5: html5

0: 其他系统

#### **其他设备相关函数**



**Bluetooth\_Close:**

```
void Bluetooth_Close();
```

功能：关闭蓝牙。

**Bluetooth\_GetList:**

```
BluetoothInfo Bluetooth_GetList();
```

功能：获取蓝牙列表信息。

使用方法：

```
int i_num = 0;

BluetoothInfo info;

info = Bluetooth_GetList();

for(i_num = 0; i_num < info.count; ++i_num)
{
    logoutf("Name:%s", info.NAME[i_num]);
    logoutf("RSSI:%s", info.RSSI[i_num]);
    logoutf("MAC:%s", info.MAC[i_num]);
}
```

**Bluetooth\_StartScan:**

```
void Bluetooth_StartScan();
```

功能：开启蓝牙搜索功能。

**ShakeDevice:**

```
void ShakeDevice();
```

功能：使设备振动。

**GetRecorder:**

```
void GetRecorder(char *path);
```

说明：打开录音功能（调用系统录音功能）

参数：

path: 录音完成后返回的音频文件的位置

#### **GetLocalPic:**

```
void GetLocalPic(char *path);
```

功能: 选择本地图片并获取路径 (调用系统照相机查看图片功能)

#### **WIFI\_GetList:**

```
WiFiInfo WIFI_GetList();
```

功能: 获取 **wifi** 列表信息

使用说明:

```
int i_num = 0;

WiFiInfo info;

info = WIFI_GetList();

for(i_num = 0; i_num < info.count; ++i_num)
{
    logoutf("SSID:%s", info.SSID[i_num]);
    logoutf("LEVEL:%s", info.LEVEL[i_num]);
    logoutf("MAC:%s", info.MAC[i_num]);
}
```

#### **标准 C 函数**

##### **abs:**

```
double abs(double num);
```

功能: 返回数字的绝对值

脚本语法:

```
int num =5;

myactor.textNumber=abs(num);
```

##### **acos:**

```
double acos (double arg);
```

功能: 返回 *arg* 的反余弦 (弧度), *arg* 的值为余弦值

注意: 参数必须在 -1 到 1 之间; 否则会发生 domain 错误

**asin:**

```
double asin (double arg);
```

功能: 返回 *arg* 的反正弦 (弧度), *arg* 的值为正弦值

如: `radtodeg(asin(0.5))` 值为 30 度

注意: 参数必须在 -1 到 1 之间; 否则会发生 domain 错误

**atan:**

```
double atan (double arg);
```

功能: 返回 *arg* 的反正切 (弧度), *arg* 的值为正切值。

如: `radtodeg(atan(1))` 值为 45 度

**atan2:**

```
double atan2(double y, double x);
```

功能: 函数计算 *y/x* 的反正切值, 返回 *arg* 的反正切 (弧度), 按照参数的符号计算所在的象限。

如: `radtodeg(atan2(1, -1))` 反正切值为 135 度

**atof:**

```
double atof (const char *str);
```

功能: 把 *str* 指向的字符串转换成 double 型

字符串必须包含浮点型数字。如果没有, 则返回的值是未定义的。该数字可以以一个字符结尾, 但不能是浮点数的一部分。包括空格、标点(除了句号)和字符(除了 E 或 e)。意思就是比如当 "300.00 MobileCreator is great." 调用 `atof()` 时, 返回值是 300.00。

**atoi:**

```
int atoi (const char * str);
```

功能: 把 *str* 指向的字符串转换成 int 型

字符串必须包含整型数字。如果没有, 则返回的值是未定义的。该数字可以以一个字符结

尾，但不能是整型数的一部分。包括空格、标点和字符。比如当”456.78”调用 `atoi()` 则返回值是 456，且”.78”被忽略

**atol:**

```
long atol (const char * str);
```

功能：把 `str` 指向的字符串转换成 `long` 型

字符串必须包含整型型数字。如果没有，则返回的值是未定义的。该数字可以以一个字符结尾，但不能是整型数的一部分。包括空格、标点和字符。比如当”456.78”调用 `atoi()` 则返回值是 456L，”.78”被忽略

**ceil:**

```
double ceil (double num);
```

功能：返回不小于 `num` 的最小整数(表现为浮点型)

比如，给定 3.02，`ceil()` 返回 4.0，给定-3.02，返回-3

脚本语法：

```
myactor.textNumber = ceil(3.02);
```

**cos:**

```
double cos (double arg);
```

功能：返回 `arg` 的余弦，`arg` 的值必须为弧度。

如：`cos(degtorad(60))` 余弦值为 0.5

**cosh:**

```
double cosh (double arg);
```

功能：返回 `arg` 的双曲线余弦值

**fclose:**

```
int fclose(FILE * stream);
```

功能：关闭与 `stream` 关联的文件

若成功，返回 0，否则返回 EOF。尝试关闭已经关闭的文件会产生错误。在关闭文件之前删

除媒体存储器也会产生错误，因为缺少足够的磁盘空间

#### **feof:**

```
int feof(FILE *stream);
```

功能：检测流上的文件结束符。

如果文件指示符在文件末端则返回非 0 值，否则返回 0。一旦到达文件末端，后继的读取操作会返回 EOF 直到 `rewind()` 被调用或使用 `fseek()` 移动文件指示符。`feof()` 在使用二进制操作时特别有用，因为文件结束符也是二进制的整数。

#### **fgetc:**

```
int fgetc(FILE *stream);
```

功能：返回指定输入流的下个字节并增加文件位置指示符。

读取的字符为无符号字符。如果读取操作到达文件末端，`fgetc()` 返回 EOF。无论如何，在 EOF 是一个有效整数前，使用二进制文件时必须使用 `feof()` 来检查文件末端。如果 `fgetc()` 遇到错误也会返回 EOF。使用二进制文件，必须用 `ferror()` 检查文件错误

#### **fgets:**

```
char *fgets(char *str, int num, FILE *stream);
```

功能：从流中读取长度为 `num-1` 的字符串并储存在指向 `str` 的字符数组中。

如果成功, 返回 `str`, 失败返回空指针

字符会一直读取到换行、接收到错误或到达指定界限才停下。字符读取完成后，会在最后一个字符后面储存一个空字符，并保留换行符。

#### **floor:**

```
double floor(double num);
```

功能：返回不大于 `num` 的最大整数(表现为浮点型)

比如给定 3.02，返回 3.0，给定 -3.02，返回 -4.0

脚本语法：

```
myactor.textNumber = floor(3.02);
```

**fmod:**

```
double fmod(double a, double b);
```

功能：返回 a/b 的余数

**fopen:**

```
FILE *fopen(const char *fname, const char *mode)
```

功能：打开 fname 指向的文件并与 stream 关联后返回。其路径在游戏相对目录下。

Mode 代表 stream 形态。文件名必须是操作系统上有效的文件名。

脚本语法：

从文件中读取文档的脚本：

```
char textArray[10][256]; // 10 条文本，每条文本最大 255 个字符
```

```
int nText = 0; //文本读取序号
```

```
void readText(char * fileName)
```

```
{  
    char line[256];  
    FILE *arq = fopen(fileName, "r");  
    if(arq)  
    {  
        while(fgets(line, 255, arq) && nText < 10)  
        {  
            if(strlen(line) > 0) //不能输入空行  
            {  
                strcpy(textArray[nText], line);  
                nText++;  
            }  
        }  
    }  
}
```

**fprintf:**

```
int fprintf(FILE * fp, const char *fmt, ...);
```

功能：在选择的文件中写入格式化后的字符串。

#### **fputc:**

```
fputc( int ch, FILE *stream);
```

功能：在指定 stream 的当前文件位置中写入 ch 字符然后往前移动指示符。

即使 ch 因为历史原因被声明为 int 型，它仍然能被 fputc() 转换成无符号型 char，

fputc() 返回的是写入的字符。如果发生错误，则返回 EOF。使用二进制打开文件时，EOF 可能是一个有效的字符，且必须使用 ferror() 函数来决定是否发出错误。

#### **fputs:**

```
fputs(const char * str, FILE *stream);
```

功能：在指定的 stream 中写入指向 str 的 string 内容. 结束符不写入。

成功时返回非负数，失败产生 EOF。

#### **fread:**

```
size_t fread(void *buf, size_t size, size_t count, FILE * stream);
```

功能：从 stream 指向的流开始，读取流中数据，每个元素的长度为 size 字节，储存在 bu 指向 f 的数组中。文件指示符按照已读字符的数量推进。fread() 返回实际读取的元素的数量。如果读取的元素数量少于请求的数量，会产生错误或到达文件末端，你必须使用 feof() 或者 ferror() 来检测已到达的地方。

#### **fscanf:**

```
int fscanf(FILE *stream, const char *format, ...);
```

功能：从指定的 stream 中读取信息。fscanf() 已被分配值的参数的数量。该数量不包括跳过的文件。EOF 的返回值意思是在分配第一个参数之前错误就发生了。

#### **fseek:**

```
int fseek(FILE *stream, long int offset, int origin);
```

功能：设置 stream 指向以 origin 为基准，偏移 offset 个字节的位置。目的是支持随机存取

I/O 操作。成功 返回 0，否则返回非 0。通常，`fseek()` 只能被二进制文件使用。

#### **ftell:**

```
long int ftell (FILE *stream);
```

功能：返回文件当前位置相对于文件首的偏移字节数。对于二进制文件，返回值就是指针从文件开始处的字节数。对于文本 *stream*，返回值可能是无意义的(除了 `fseek()` 的参数)。因为字符可能已经转换，就像用回车符/换行符代替换行，这会影响文件的大小。

产生错误时返回-1

#### **fwrite:**

```
size_t fwrite(const void *buf, size_t size, size_t count, FILE *stream);
```

功能：从指向 *stream* 的流开始读取，读取长度为 *size* 位，，并储存在 *buf* 指向的数组中。文件指示符按照已读字符的数量推进。

返回实际写入的元素的数量。如果函数运行成功，则返回的值等于请求的数量，如果写入的元素数量少于请求的数量，会产生错误。

#### **log:**

```
double log(double num);
```

功能：返回 *num* 的自然对数。如果 *num* 是负数会产生 domain 错误，如果 *num* 是 0 可能会产生 range 错误

#### **log10:**

```
double log10(double num);
```

功能：返回基数为 10 的 *num* 的对数。如果 *num* 是负数会产生 domain 错误，如果 *num* 是 0 可能会产生 range 错误。

#### **memcmp:**

```
int memcmp(const void *buf1, const void *buf2, size_t count);
```

功能：比较 *buf1* 和 *buf2* 指向的数组的第一个字符。返回一个整数。



**memcpy:**

```
void *memcpy(void *to, const void *from, size_t count);
```

功能：从 from 指向的数组复制 count 个字符到 to 的数组里。如果数组重叠则 memcpy() 未定义。返回一个指针给 to。

**memmove:**

```
void *memmove(void *to, const void * from, size_t count);
```

功能：从 from 指向的数组复制 count 个字符到 to 的数组里。如果数组重叠，复制的内容会选择恰当的地方，把内容放进 to 里并从 from 里删除。返回一个指针给 to。

**memset:**

```
void *memset( void *dest, int c, size_t count );
```

功能：将 dest 所指向的某一块内存中的每个字节的内容全部设置为 c 指定的 ASCII 值，块的大小由第三个 count 指定

**pow:**

```
double pow(double base, double exp);
```

功能：返回 base 的 exp 次方。当 base 等于 0 和 exp 小于或等于 0 时会发生 domain 错误。base 是负数和 exp 不是整数时也有可能发生 range 错误。

**sign:**

```
double sign(double a);
```

功能：返回一个整数以表明数字的符号(+或-)。

**sin:**

```
double sin(double arg);
```

功能：返回 arg 的正弦值。arg 的值必须是弧度。

如：sin(degtorad(30)) 正弦值为 0.5

**sinh:**

```
double sinh (double arg);
```

功能：返回 arg 的双曲线正弦值。

#### **sprintf:**

```
sprintf(youractor.text, "Hello %s", stringvar);
```

功能：使用 sprintf 让固定文本(像" Hello")和变量文本(像角色名，数字...).

脚本语法：

```
char stringvar[256] = "";  
sprintf(youractor.text, "Hello %s", stringvar);
```

#### **sqrt:**

```
double sqrt (double num);
```

功能：返回 num 的平方根，如果调用的是负数，会产生一个 domain 错误。

#### **sscanf:**

```
int sscanf(const char *buf, const char *format,...);
```

功能：从一个字符串中读进与指定格式相符的数据

#### **strcat:**

```
char *strcat(char *str1, const char *str2);
```

功能：复制 str2 与 str1 连接并添加一个终止符给 str1. str1 后面原来的终止符被 str2 的第一个字符覆盖。如果数组重复则 strcat() 未定义。

#### **strchr:**

```
char *strchr(const char *str, int ch);
```

功能：查找字符串 s 中首次出现字符 ch 的位置。返回首次出现 ch 的位置的指针，如果 s 中不存在 ch 则返回 NULL。

#### **strcmp:**

```
int strcmp(const char *str1, const char *str2);
```

功能：比较两个字符串并基于结果返回一个整数。

说明：

当  $s1 < s2$  时，返回值  $< 0$

当  $s1 == s2$  时，返回值  $= 0$

当  $s1 > s2$  时，返回值  $> 0$

#### **strcpy:**

```
char *strcpy(char * dst, const char * src);
```

功能：复制 str2 的内容给 str1。Str2 必须是指向以 NULL 结束的字符串的指针。strcpy() 返回一个指针给 str1。

#### **strlen:**

```
size_t strlen(const char *str);
```

功能：返回 str 的长度，不包括结束符 NULL。

#### **strncat:**

```
char *strncat(char * s1, const char *s2, size_t n);
```

功能：把 s2 所指字符串的前 n 个字符添加到 s1 结尾处(覆盖 s1 结尾处的 '\0') 并添加 '\0'。

#### **strncmp:**

```
int strncmp(const char * s1, const char *s2, size_t n);
```

功能：比较两个字符串前 n 个字符并基于结果返回一个整数。

说明：

当  $s1 < s2$  时，返回值  $< 0$

当  $s1 == s2$  时，返回值  $= 0$

当  $s1 > s2$  时，返回值  $> 0$

#### **strncpy:**

```
char * strncpy(char *str1, const char *str2, size_t count);
```

功能：将字符串 str2 中最多 count 个字符复制到字符数组 str1 中。

**tan:**

```
double tan (double arg);
```

功能：返回 arg 的正切值，arg 的值必须是弧度。

如：tan(degtorad(45)) 正切值为 1

**tanh:**

```
double tanh(double arg);
```

功能：返回 arg 的双曲线正切值。