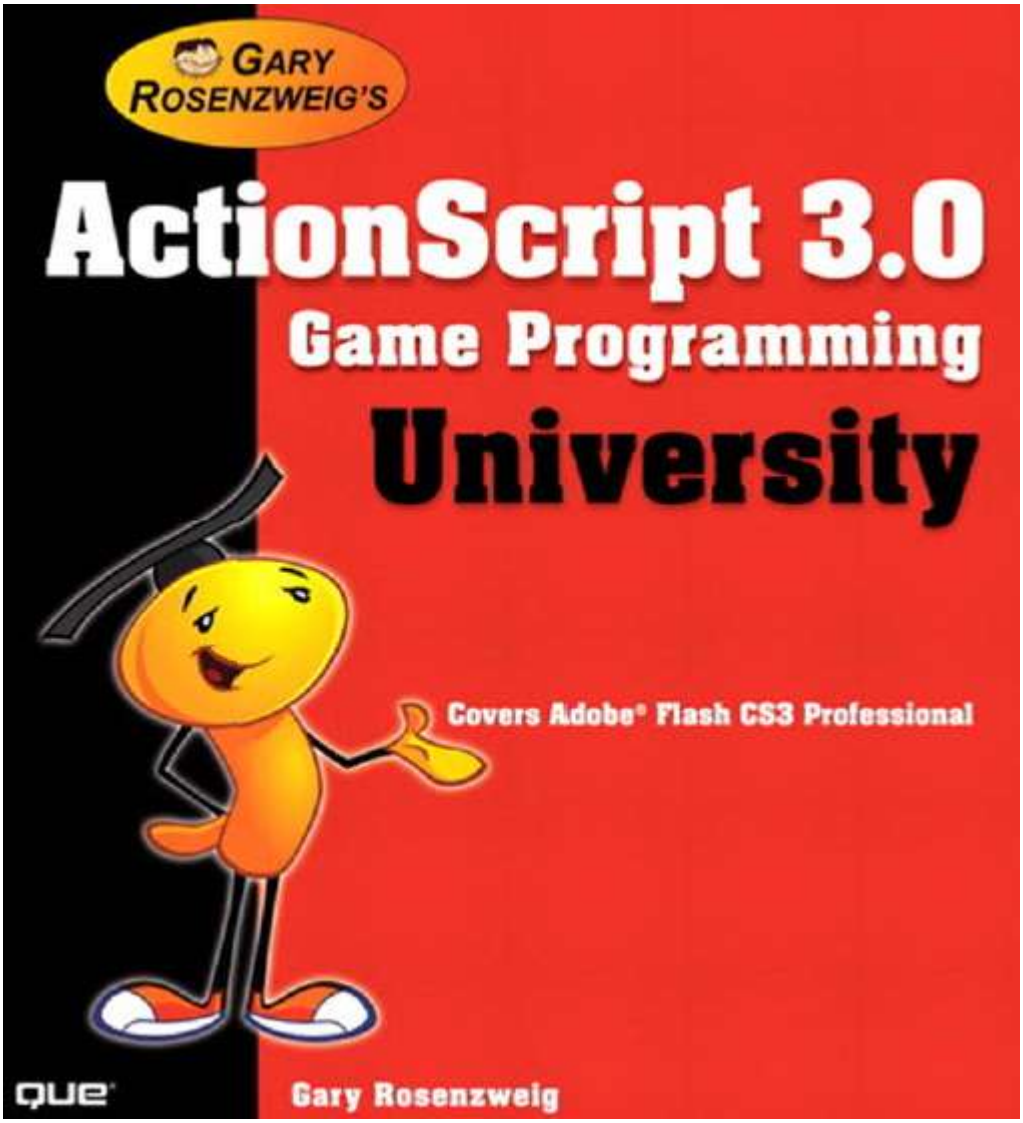
AS 3.0 游戏编程大学


GARY
ROSENZWEIG'S
ActionScript 3.0
Game Programming
University
Covers Adobe® Flash CS3 Professional
que
Gary Rosenzweig

本书目录

3. 一般性游戏框架: A Matching Game
4. 头脑游戏: Memory and Deduction
5. 动画游戏: Shooting and Bouncing Games
6. 拼图游戏: Sliding and Jigsaw
7. 方向和运动: Space Rocks
8. 不规则游戏: Match Three
9. 单词游戏: Hangman and Word Search
10. 问答游戏: Trivia and Quiz Games
11. 战斗游戏: Platform Games
12. 游戏世界: Driving and Exploration Game

绪论

对 Flash 游戏制作者来说，这是一个非常好的时期。到目前为止，还没有一款软件比这更好的来设计中小型游戏了。

Flash CS3 Professional 对我们来说，是快速的，强大的和容易掌握的。其中最关键的就是，Flash 最新发布的版本有着强大的 ActionScript 3.0 编程语言。

ActionScript1.0 和 2.0 让 Flash 制作者感到太伤心了，在程序的调试过程中，经常会出现一些 Bugs，导致不能快速的制作出自己的作品。

相对而言，ActionScript 3.0 编程语言会让你感到非同寻常。在使用中，你会发现它能够快速地，非常容易的制作出自己想象中的作品。

最后，就让这本书带你走进 Flash 游戏的世界中吧，希望通过本书的学习，它可以带给你更多的快乐。

0.1 Flash 游戏的发展

1995 年 10 月，对我这个游戏设计师来说，是非常激动的。因为 Macromedia 刚刚发布了 Shockwave 播放器，这样就可以把自己制作出来的作品发布到网络上了。

从最初的 Shockwave 播放器发布之后，还有两次比较激动人心的就是，Shockwave 3D 和 ActionScript3.0 的发布。

虽然 Flash 游戏环绕在我们周围，但如果和 Shockwave 游戏比起来，它就比较逊色了，因为 Shockwave 软件在 3D 方面是非常快速和强大的。

不管怎样，我们现在有了 ActionScript3.0，Flash 就会变得和 Shockwave 一样强大。例如，Flash 9 播放器已经占有了全球 80%的桌面电脑。截至到现在，几乎全部安装 Flash 8 播放器的用户已经将该软件更新到了 Flash 9 的版本，这对我们 Flash 游戏制作者来说，无非有莫大的帮助。

Flash 9 播放器还可以应用于 Linux 系统，网络电视窗口，游戏控制台等多方面软件，现在，我们拥有了 Flash 9player 和 ActionScript 3.0 以后，就可以轻便地来做这些事情了。

你可以开发双方独立的或者基于网络版本的 Flash 游戏，当然，还还可以利用第三方的软件来扩展你的 Flash 功能，以便编译出更强壮的代码。

总之，对制作中小型的游戏来说，ActionScript 3.0 的功能是非常强大和实用的。

0.2 本书为谁而写

本书适用于使用 Flash 制作游戏的任何人，读者可以根据自身情况选择不同的方式来用好此书：

对 Flash 绘图和编程都没有学过的新手来说，最好先学习一些基本的编程语法再看。当然，如果你对 Flash 有浓厚的兴趣，也可以直接用此书来学习 ActionScript 3.0；

如果你以前使用的是 ActionScript 1.0 或者 2.0，你也可以用此书快速的学会 ActionScript 3.0。不过，你应该尽可能的把以前 ActionScript 版本的编程方式忘掉。确切地来说，ActionScript 3.0 与前期的 1.0 和 2.0 的版本已经有很大不同。我自己认为它是一门全新的语言。

对于有些动画和编程基础的人来说，如果想进一步学习游戏编程，本书是再适合不过了；

如果你没有学过编程，但喜欢绘图，插图，或者动画。你可以用本书的例子作为素材。也就是说，你可以把本书中源文件拿出来用到你自己制作的游戏中去；

同样的，如果你是优秀的 ActionScript 3.0 程序师，本书也可以作为给你提供的一个代码库，你可以把它运用到自己设计的游戏中去。

0.3 使用本书需要具备的知识

除了正确地使用软件之外，对大多数读者来说，如果想从本书中获取更多的知识，还需要对 Flash 编程基本知识做些准备。

必备知识

对大多数读者来说，您应该对 Flash CS3 的运行环境很熟悉了。如果你是一位新的 Flash 用户，可以在 Flash CS3 的软件中打开用户指导说明，或者选择 Flash 帮助或按 F1 键查看帮助。另外，你也可以买本初级者的书翻阅或者在网上直接观看在线教程。

本书是分章节讲授的，各章节间没有太大的连贯性。如果你想应用本书范例中代码的话。你只要有些编程基础就可以了，比如 ActionScript1.0 或者 2.0，JavaScript，Java，Lingo，Perl，PHP，C＋＋等等任何结构化设计语言中的一样就可以。只要你稍微对循环，条件语句和函数等知识有些了解，也就很容易理解 ActionScript 3.0 语言了。在第一章中“Flash 和 ActionScript 3.0 的应用”就概括了这些基本的语法。

如果你是一位程序员，以前没有使用过 Flash 软件的话，首先应该阅读一下 Flash 用户指南中关于绘图和编程的基础部分。

软件应用

当然，学习本书需要用到 Flash CS3 Professional 或者更新的 Flash 8 Studio，本书不适用于之前 Flash 版本的用户。

Flash CS3 在 Windows 和 Mac 系统上使用效果是一样的，本书中的屏幕截图取自 Mac 系统中的 Flash 程序，与 Windows 系统中 Flash 的截图应该是一致的。

Flash 将来的版本很有可能是 ActionScript 3.0 编程语言的延续，或许一些菜单选项和快捷键要发生变化，但你仍然能够使用本书。这时，你要考虑在发布文档时设置成 Flash 9 播放器和 ActionScript3.0，以确保更大的兼容性。

源文件

如果你需要本书的源文件，请看下面的怎样获取源文件的说明。

0.4 在你的游戏中如何使用示例

本书包含 16 个完整的游戏，其中也包括一些经典的游戏，例如：match three，asilde－scrolling platform 游戏和 word search。经常有人这样问我：“能够把这些游戏运用到自己的作品中呢？”

答案就是，当然了，你可以把这些游戏都修改成自己的，例如，你可以改变里面的图形、游戏的开始或其他内容等部分都可以。但是本书不提倡把这些游戏或者源代码清单原封不动的发布或者复制。

当你在你的方案中用这些游戏作品时，不要直接复制过去，这样是不专业的，在下面的网站中可以找到答案 http://flashgameu.com

如果你仅使用其中的一部分代码，或者一个小游戏作为某个方案的框架，这无不可。

总之，要正确的了解这些常识和惯例，谢谢。

0.5 你在这本书中可以学到什么

第一章，“Flash 和 ActionScript 3.0 的应用”，主要介绍了 ActionScript 3.0 和一些基本的概念，如游戏编程策略，游戏清单等，这些可以帮助你在 Flash CS3 中更好地制作游戏。

第二章，“ActionScript 游戏元素”，呈献給大家一些函数的代码片段，例如创建文本域，绘制图形，播放声音等。这些在代码库中都是非常有用的，也始终贯穿于本书当中。

第三章至第十二章，每一章都包括一个或多个完整的游戏，书中正文的每一章都有游戏代码供你使用，或者你也可以用本书提供的源文件和代码。

第三章，“基本游戏结构：A Matching Game”，这个与本书其他的游戏有些不同。它包括完成游戏前怎样检查代码，设计一个完整的游戏需要哪些步骤等说明。

其余章节大部分在开始制作游戏之前都包含一个特定的主题，例如：在第四章中就以“数组和数据对象”开始讲解的。

但是，本书所介绍的内容远远不止书本上这些，在线还可以获取更多的知识。

0.6 关于 FlashGameU.com 网站

这个 FlashGameU.com 网站可以说是这本书的伙伴，在这个网站上可以找到这个源文件，补充资料以及新的内容，并设有 Flash 发展论坛和作者的 Blog 等。

本书的源文件是按章节划分的，在包内部还更进一步的对每个小游戏打包，在网站主页上有链接可以直接下载。

在网站上，你能够看到作者的 Blog，如果你对本书有什么建议或者意见，或者有关于 Flash 游戏发展存在的问题，可以直接在论坛上留言或者通过作者的 Blog 联系。

Flash 和 ActionScript 3.0 的应用

1.1 、ActionScript 3.0 是什么

ActionScript3.0 是在 2006 年发布 Flex 2 时提出来的，Flex 允许开发者创建的 Flash Player 程序和 Flash 一样。但是，Flash 却给我们提供了更加直观的界面，因此它就更适合用作游戏开发。

ActionScript 是在 Flash4 发布的时候提出来的，当时的名称就叫做 ActionScript，他的主要能是用来控制帧的跳转和鼠标，键盘的交互。直到发布 Flash5 之后，ActionScript 才正式地作为 ActionScript1.0 使用，但是它的发展速度相当快，到了 Flash MX2004（Flash 7），就引进了 ActionScript2.0，它遵循 ECMA 标准，在面向对象编程方面，它开始变得逐步强大。像应用于浏览器的 JavaScript，也是基于 ECMA 标准的。

注：Flash 9 播放器内部建有 2 套虚拟机，AVM1 用来执行 ActionScript1.0 和 2.0 代码，AVM2 用来执行 ActionScript3.0 代码，但是 AVM2 的速度要比 AVM1 快得多。

可以说 ActionScript3.0 最近几年已经发展到顶点了。Flash 每个版本的发布，大家都会把它推到极限。现在，我们已经有了一个优秀的开发环境来制作我们的 2D 游戏了，你将会体会到用一小段代码就可以让游戏运行起来的奥妙。

1.2 创建一个简单的脚本程序

首先，打开 Flash 应用程序，选择“文件”－>“新建”，会出现一个文档窗口，然后，选择第一项“Flash 文件（ActionScript3.0）”，点击“确定”，软件会自动生成一个“未命名－1”的空白文档。

可以看到，在该文档顶部有一个时间轴，还有帧数，你看到帧数量的多少取决于 Flash 软件显示窗口的大小。在 Flash 绘图时需要用到很多帧，但对于我们游戏制作者来说，用少数的几帧就能实现效果了。

时间轴可以有一层或者多层，在新建文档的窗口中默认自动打开一层，名称为“图层一”，在第一帧你可以看到，有个空心小圆圈的标志，它就表示该帧为空白关键帧。

注：关键帧是一个动画术语，在 Flash 动画制作中会经常用到，它的作用就是定义在某一个时间点上元素的状态和参数的变化。例如，选中时间轴的第 1 帧，在舞台的左端放置一个影片剪辑，然后在第 9 帧插入一个关键帧，并把该影片剪辑移动到舞台的右端，然后在第 1 帧至第 9 帧的任一帧点击鼠标右键创建补间动画，然后选择菜单“控制”－>“测试影片”，发现该影片剪辑已经能够在舞台上运动了。对我们游戏编程者来说，我们不会用关键帧来做动画，需要的只是在关键帧时间点上放置游戏中需要的“说明”，“播放”，“游戏结束”等物件而已。

你还可以在任何一层的任意关键帧输入自己的脚本代码，具体操作是选中这个关键帧，点击 Flash 菜单“窗口”－>“动作”或者按【F9】键即可打开脚本编辑区输入自己的代码。

在脚本编辑区的上部，有可以实现很多功能的控制选项，例如可以对代码进行格式套用等。但是，我们不准备详细介绍这些控制选项，因为后面写出的代码大部分都在外部类中。

下面，就让我们来创建一个简单的 Hello World 程序吧，在脚本编辑区输入以下代码：

```
trace(“Hello World”);
```

好了，就是这么简单，现在你已经创建了第一个 AS3.0 的程序。来测试一下吧，选择“控制”－>“测试影片”或者按【Ctrl】＋回车键。如果你没有创建自己的 Flash 影片，可以直接打

开 HelloWorld1.fla 文件进行测试。

现在，来看一下输出面板，在窗口的左上角你会看到一行输出了“Hello World”，但是它只用于在 Flash 9 中测试影片，它不会显示在舞台上，如果想显示在舞台上的话，我们还需要进一步做更多些的工作。

创建可视对象

如果想让“Hello World”在影片中能够看到的话，我们用 3 行代码就能够实现了。

第一行需要创建一个文本域显示在屏幕上，她的作用是用来存放文本的容器；

第二行把该段文本存放在文本域内就可以了；

第三行是把这个文本域添加到舞台上，舞台上影片的显示区域，你可以在舞台上布置自己的元件，在测试影片时就能够看到了。

显示对象，在 AS3.0 中实际上就是一个能够看得见的元素，他可以是文本区域，图形，按钮，或者像弹出菜单式的界面。显示对象里面还可以包含其他的显示对象，例如，在国际象棋游戏中，一个显示对象会包含所有的部分，棋盘是底部的一个显示对象，舞台也是一个显示对象。

下面给出这三行代码，可以把它粘帖到第 1 帧中代替以前的那一行代码：

```
var myText:TextField=new TextField();
myText.text="Hello World";
addChild(myText);
```

首先，先创建了一个 TextField 类型的变量，然后把“Hello World”赋值给这个变量，最后在把它添加到舞台上就完成任务了。

注：var 是一个关键字，用来声明变量，变量的数据类型写在冒号的后面，如果要赋值的话，值得数据类型必须要和数据类型一致。

然后，选择控制，测试影片，你就能够在影片的左上部发现 trace()出来的结果。

因为我们没有设置文本的任何格式，所以才会在影片的左上部出现软件默认的设置值。随着我们学习的深入，我们还可以设置文本的位置，大小，字体等格式。

创建一个 AS3.0 类文件

如果我们没有特别需求的话，不会在时间轴上写代码，而是在大多数时候，把代码都写到类文件中。

下面，就让我们用外部类建立一个 Hello World 程序吧^_^

首先，选择“文件”－>“新建”->“ActionScript 文件”，点击确定后创建一个脚本文件。我们可以看到，该文件与 fla 文件有很大不同，他没有时间轴和舞台工作区了，整个界面都是脚本编辑区域。

下面，我们尝试着用类文件写这个程序，你也可以先打开源文件看一下这个格式：

代码如下：

```
package
{
        import flash.display.MovieClip;
        import flash.text.TextField

        /**
        * ...
        * @author  ☆璀璨星晨☆
```

```
*/
public class DebugExample extends MovieClip
{

	public function DebugExample()
	{
		for (var i:int = 0; i < 10; i++ ) {
			shouNumber(i)
		}
		trace("循环体外")
	}

	private function shouNumber(whichNum:int):void
	{
		var myText:TextField = new TextField()
		myText.text = String(whichNum)
		myText.y = whichNum * 20
		addChild(myText)
	}

}


}
```

首先，类（Class）的声明要写在包（package）内才行，然后，才能根据程序的需要定义相关的内容。在这个类文件中，我们需要创建一个文本区域，并作为显示对象在舞台上显示出来。那么，根据这些要求，我们必须要导入 flash.display 类和 flash.text 类这两个类才行：

```
package {
import flash.display.*;
import flash.text.*;
```

注意：这只是根据需要导入的两个类，如果你想了解更多类方面知识的话，请参考 Flash 9 帮助文档。

继续往下看，下面的代码定义了一个 public 类，public 用来控制类成员的访问权限，表示该成员从外部可以调用或访问到。注意，这里类的名称必须和脚本文档名称一致才行，不妨都把它们命名为 HelloWorld3。另外，这个类继承于 MovieClip，这就意味着主要 MovieClip 所拥有的属性和方法，HelloWorld3 这个类都会继承过来，代码如下面所示：

```
public class HelloWorld3 extends MovieClip{
```

我们还会看到，在类里面还包含一个函数，函数名称和类名一样，也是 HelloWorld3。像这种函数名与类名相同的函数，我们称之为构造函数，在类初始化时该函数就会被立即执行。

在函数里面，是我们前面讲过的三行代码：

```
public function HelloWorld3() {
var myText:TextField = new TextField();
myText.text = “Hello World!”;
addChild(myText);
```

```
}
}
}
```

为了让这些代码能够在测试影片时被执行，你还需要新建一个 flash 文档，不妨也命名为 HelloWorld3（这里文件名可任意），注意要吧这个 fla 文件与刚才的类文件保存在一个文件夹下才行。注意，这里我们就不需要在时间轴上写任何东西啦，但必须要绑定一个文档类，才能够让刚才编写的代码执行。选择菜单“窗口”—>“属性”，在文档类后面的区域输入刚才的类名 HelloWorld3，

好了，现在可以打开控制—>测试影片了。在这里，系统就会自动编译这个类文件，对文档类进行初始化，执行构造函数里面的语句，并将“”Hello World 的文本显示到舞台上。

1.3Flash CS3 的使用

本节主要介绍了显示对象，显示列表，舞台，库和时间轴等。

显示对象和显示列表

显示对象就是可以在舞台上显示的对象，包含可以直接看到的文本，图形，视频和图片等内容。所有显示对象类都是 DisplayObject 类的子类。另外，显示对象容器也可以作为一种特殊类型的看不到的显示对象，在它的内部还可以包含子显示对象。

显示列表：由 Flash Player 呈现给屏幕内容的显示对象的层次结构，舞台是显示列表的根，附加到舞台或其子级上的所有显示对象构成了显示列表。当然，在列表中，也包括位于舞台边界以外的对象。

在这里，作者主要介绍了 MovieClip 和 Sprite。其中 MovieClip 通常是由 Flash 绘制工具创建出来的，因为绘制工具创建出来的是含有时间轴的影片，而只有 MovieClip 才拥有和时间轴相关的属性和操作。但是，在 AS3.0 中，MovieClip 的重要性已经不如以前了。

在以后编写程序的过程中，我们使用最多的就是 sprite 类，可以把它看成去掉时间轴的 MovieClip，它含有 graphic 方法，可以直接在自身绘图，还可以包含子显示对象。

舞台

舞台是 Flash 中的主绘图区，它就是在游戏播放时我们看到的部分。我们在制作游戏时，可以在舞台上绘制元件，还可以把库中的元件拖放到舞台上。

库

库是存储和组织在 Flash 中创建的各种元件的地方，它还用于存储和组织导入的文件，包括位图图形、声音文件和视频剪辑。 在库中可以组织文件夹中的库项目，查看项目在文档中使用的频率，并按类型对项目排序。

时间轴

时间轴用于组织和控制一定时间内的图层和帧中的文档内容。 与胶片一样，Flash 文档也将时长分为帧。 图层就像堆叠在一起的多张幻灯胶片一样，每个图层都包含一个显示在舞台中的不同图像。 时间轴的主要组件是图层、帧和播放头。

文档中的图层列在时间轴左侧的列中。每个图层中包含的帧显示在该图层名右侧的一行中。时间轴顶部的时间轴标题指示帧编号。 播放头指示当前在舞台中显示的帧。 播放文档时，

播放头从左向右通过时间轴。
时间轴状态显示在时间轴的底部，它指示所选的帧编号、当前帧频以及到当前帧为止的运行时间。

1.4 脚本编辑区

在创建游戏时，虽然 Flash 文档窗口对我们来说是很重要的，但是让我们花费大量时间的地方还是在动作面板中脚本编辑区。我们在前面章节中已经看到过这个编辑区，但是下面的这个却有些不同，因为在左边有一个 AS3.0 的分层菜单（也就是工具箱）。

我们可以通过键盘上的【F9】键或者选择窗口－>动作面板菜单命令，打开上面的脚本编辑区，在该区的上部，软件为我们提供了很多在代码编写过程中需要用到的辅助功能：
将新项目添加到脚本中：显示 AS 工具箱中具有的所有语言元素，从语言元素的分类列表中选择一项添加到脚本中。

查找：在 AS 编辑区中查找和替换文本；

语法检查：检查当前 AS 编辑器中的代码是否存在语法错误，如果有语法错误将在输出面板准中显示出来。

自动套用格式：设置 AS 的格式以实现正确的编码语法和更好的可读性，可以在首选参数对话框中设置自动套用格式的首选参数，从编辑菜单或通过动作面板弹出的菜单可以访问此对话框。

显示代码提示：显示正在编写的代码行的代码提示，这个功能在所有的按钮中是最有用的。例如当你在编写程序时，当你输入到 gotoAndStop（，你立即就会看到这个方法接受什么参数以及参数类型，这是不是很方便啊？^_^
调试选项：调试代码时用来添加或移除断点。
折叠成大括号：折叠括号中的代码。
折叠选项：当你选中多行代码时，点击该按钮，你就会发现这些代码自动折叠成一行了，如果想恢复的化，可以双击该行的。按住 Alt 键，可以折叠未选中的代码。
展开全部：展开以上方式折叠的全部代码
应用块注释：把所选的代码转化为块状注释
应用行注释：把所选的代码转化为行注释
取消注释：取消所选代码的注释标记
显示/隐藏工具箱：显示/隐藏动作面板左侧的工具箱
另外，在脚本编辑区的左边有一列数字，它是用来标识缩写代码的行数。当我们在测试影片出现错误时，将会在编译器错误报告中显示出错误代码的行数，描述等，这样可以保证我们能够很快的找到错误代码。

1.5 ActionScript 游戏编程策略

在游戏编程中要注意一下几点：
逐步逼近原则
无论是初学者还是有一定经验的编程人员，在写代码时都会碰到这样的问题，我不清楚这个工程的代码是怎样编写出来的。例如，一个程序员想制作一艘太空飞船，要求就是当按住方向键时，出现旋转效果。对于这个任务，其关键点就是：当按住左方向键时，在飞船的 rotation 属性减去某一角度；相反，当你按住了右方向键是，则要在飞船的 rotation 属性中加上某一角度即可。把它分开来思考就是这么简单～～

很多初学者在开始会觉得比较困难，他们总认为自己不能够制作一个完整的游戏，因为它看起来是非常的复杂。但是，如果你把它分成小块的话，一次就设计一小块，最后，你就可以完整地创建出任何游戏。

用好注释：

当你书写代码的时候，在适当地位置加上注释是非常有必要的，现在看起来好像是额外工作。但是，你过几个月后再返回来看这些代码时，就会发现加上的这些注释都非常有用。

一般来说，注释分为行注释和块注释，行注释的内容比较少，一般放在行代码的前面或后面；而块注释的内容就比较多了，一般用一个或者几个句子来说明，它放在一个函数或者一个代码块的前面。例如：

```
someActionScriptCode();//这是一个行注释

//这是一个行注释
someActionScriptCode();

/*这是一个块注释
块注释比较长
并且还包含下面代码的一段描述*/
```

另外，需要注意的就是，注释一定要简洁明了，也不要简单的重述代码中体现出来的内容。例如，下面这行注释就没有多大意义了：

```
//循环 10 次
for(var i:int=0;i<10;i++)
```

变量和函数名称的使用

不要担心函数名或者变量名过长不好理解，恰恰相反，使用描述性的变量名和函数名，还能够起到事半功倍的效果，因为它自身就可以作为描述性的说明，比如：

```
        public function putStuff(){
        for(var i:int=0;i<1;i++){
        var a:Ting=new Ting();
        a.x=i*10
        a.y=300
       addChild(a)
 }
}
```

这段代码是干什么的？它看起来好像是复制了 10 个影片剪辑并添加到舞台上了，但是这个影片剪辑具体是什么东西，它有什么用途？我们并不知道。下面，我们再看一下这段代码的另一种方式：

```
        public function placeEnemyCharacters(){
        for(var enemyNum:int=0;enemyNum<10;enemyNum++){
        var enemy:EnemyCharacter=new EnemyCharacter()
        enemy.x=enemyNum*10
        enemy.y=300
        addChild(enemy)
    }
```

}

如果你用上面这段代码书写的话，过几个月在反过来看的话，这一块代码是什么，有什么作用，就会一目了然。

注意：在循环语句中，有个不成文的惯例，就是循环变量一般用 i 标识，如果在内部还有循环嵌套的话，就用 j，k 表示，这样看起来比较标准美观。像上面用 enemyNum 声明变量的话，也是可以的。

把重复或类似的代码写成函数

如果你在一个程序中不止一次用到相同的代码，可以考虑一下，把该行代码写成函数的形式来调用。 例如，在你的游戏中有几处地方需要用到更新分数，假设这个分数是名称为 scoreDisplay 的文本，你可以这样写：

```
scoreDisplay.text="Score:"+playerScore;
```

试想，如果在这个程序中有 5 处地方需要添加这个分数的话，你是否要把这行代码复制到不同的 5 个地方去吗？其实你没有必要这样做，你完全可以把它定义成一个函数，在使用分数的地方直接调用该函数就可以了，例如：

```
showScore()
```

下面来定义这个函数

```
    public function showScore(){
    scoreDisplay.text="Score:"+playerScore;
}
```

另外，如果你要修改此行代码的话，也不用在整个程序中一一查找更改了，直接更改函数里面的这行就行。

即使代码不完全相同，你也可以用这个方法来处理，举个例子：假如你要复制 10 个影片剪辑 A 放置在舞台的左边，还要复制 10 个影片剪辑 B 放置在舞台的右边。那么，你可以先创建一个放置影片剪辑的函数，然后，对他调用两次就可以了，一次是剪辑 A 调用，一次是剪辑 B 调用。

分块测试你的代码

当你测试书写的代码时，分块测试比较容易调试出错误。例如，你想在舞台上放置 10 个圆，随机的位置，随机的颜色。你可以先写出创建出这 10 个随机位置的圆，然后测试代码，等得到你想要的效果之后。再添加这随机颜色的函数就可以了。

上面示例代码如下：

```
var num:int=0;
addEventListener(Event.ENTER_FRAME,creatCircle);
function creatCircle(e:Event):void
{
    var sp:Sprite=new Sprite();
    sp.graphics.beginFill(Math.random()*0xFFFFFF);
    sp.graphics.drawCircle(Math.random()*stage.stageWidth,Math.random()*stage.stageHeig
ht,30);
    sp.graphics.endFill();
    addChild(sp);
    num++;
    if (num>9)
```

```
        {
                removeEventListener(Event.ENTER_FRAME,creatCircle)
}
}
```

1.6 脚本基本概念

本节主要讲述了变量的声明和使用，条件语句，循环和函数等基本概念。

变量的声明和使用

在 ActonScript3.0 中，如果你初次使用变量，那么要用 var 关键字来声明一个变量，变量后面如果不加数据类型的话，就被归为未声明类型；一般建议变量后面跟上数据类型，以便在赋值的过程中保证值的数据类型和变量的数据类型一致。

```
var myValue:int=7
var myValue:Number=7.8
```

int 为有符号的整型数，有正负之分；uint 是一大于 0 的正整数；如果你想声明带小数点的值，可以选择 Number 类型。

另外，还有 String 和 Boolean 数据类型，String 类型来保存文本，Boolean 类型的值为 true 或 false。

上面这些是基元数据类型，可是，还包括数组等。

另操作运算符的使用在处理数据运算是也有至关重要的作用。下面是加减乘除的一些示例：

```
var myNumber:Number=7.8+2
var myOtherNumber:int=5-6
var myAnotherNumber:Number=myNumber*3
var myNextNumber:Number=myNumber/myOtherNumber
※        ※※※※※※※※※※※※※※※※※※※※
```

补充：可以在上面代码的下面加上 trace()函数,以便观察计算结果

```
trace(myNumber)
trace(myOtherNumber)
trace(myAnotherNumber)
trace(myNextNumber)
/*输出
9.8
-1
29.400000000000002
-9.8*/
※        ※※※※※※※※※※※※※※※※※※※※
```

另外，你还可以用＋＋，－－运算符来简化一些运算，＋＋为递增，－－为递减

```
myNumber++
```

你还可以用+=,-=,*=和/=算术赋值运算符来处理数据，例如，下面的运算会使变量加上 7

```
myNumber+=7
```

你还可以用()来设置这个运算符优先级顺序

```
var myNumber:Number=(3+7)*2
※          ※※※※※※※※※※※※※※※
trace(myNumber)
/*
输出：
20
*/
※          ※※※※※※※※※※※※※※※
```

另外，String 类型的值也可以应用＋，－运算符，例如

```
var myString:String="Hello"
var myOtherString:String=myString+"World"
myAnotherString+="World"
※          ※※※※※※※※※※※※※※※※※※※※※※※※
trace(myString);//该行放在第一行代码的下面
//此两行放在上面代码的最后即可
trace(myOtherString);
trace(myString);
/*
输出：
Hello
HelloWorld
HelloWorld
*/
※          ※※※※※※※※※※※※※※※※※※※※※※※※
```

当我们在类中声明变量后，这个变量就成为这个类的属性了，我们还可以在声明变量的关键字前面加上 public 或 private 等访问控制符，其区别就是 private 不能够被该类以外的代码访问。

条件语句

在 ActionScript 语言中，if 语句的功能和在其他编程语言中的作用是一样的

```
if(myValue==1){
doSomeThing()
}
```

()内的语句为条件表达式，如果它的值为真，执行{}内的流程语句；如果条件不成立，则为假，不执行流程语句。上面的条件是用==来检验左右两边的值或表达式是否相等的，另外还可以用>,<,>=,<=来判定。

在 if 结构中，你还可以添加 else 和 else if 来扩展功能

```
if(myValue==1){
doSomethig()
}else if(myValue==2){
doSomethingElse()
}else{
doNothing()
}
```

在上面的条件表达式中，还可以用逻辑运算符&&和||等来综合运算。

```
if((myValue==1)&&(myString=="This")){
doSomething()
}
```

循环

循环一般在 for 语句或者 while 语句中执行

for 语句的循环条件有三个要素，分别是循环变量设置初始值，循环条件表达式，改变循环变量值的语句。例如，下面的代码声明变量 i 并赋值为 0，条件是小于 10，并且每一次递增一来改变。

```
for(var i:int=0;i<10;i++){
doSomething()
}
```

你还可以在循环体内用 break 和 continue 来控制循环流程。break 用来直接跳出循环，不再执行循环体内后面的语句；continue 只是终止当前这一轮的循环，直接跳到下一轮循环，在这一轮循环中，循环体内 continue 后面的语句也不会被执行。

※　　※※※※※※※※※※※※※※※※※※※※※※※※※※

补充：可以用以下代码测试 break 和 continue 控制循环流程的作用

```
//使用 break 退出循环
for (var i:int=0; i<10; i++)
{
	if (i==3)
	{
		break;
	}
	trace("当前数字：\t"+i);
}
/*输出
当前数字：	0
当前数字：	1
当前数字：	2
//由以上输出可以看出，当 i＝3 时，break 跳出循环体，并且终止了循环。
*/
```

//continue 跳出当前的循环

把上面代码块中的 break 改为 continue 后，测试结果如下：

```
/*输出
当前数字：0
当前数字：1
当前数字：2
当前数字：4
当前数字：5
```

当前数字：6
当前数字：7
当前数字：8
当前数字：9*/
由上面的输出可以看出，当 i＝3 时，continue 后面的语句没有执行，而是直接跳转到下一轮循环，直到循环条件结束。
※ ※※※※※※※※※※※※※※※※※※※※※※※※

while 循环表示：当满足条件表达式的时候再执行循环体

```
var i:int=0
while(i<10){
i++
}
```

do...while 循环，先判断是否符合循环条件，然后在执行循环体，与 while 唯一不同的就是 do－while 循环体至少要执行一次

```
var i:int=0
do{
i++
}while(i<10)
```

函数
在 AS3.0 中，在创建函数之前，你必须要先声明这个函数，包括他里面包含的参数，返回值等，然后才能够调用。如果在类中的话，你也要用到 public 或者 private 等访问控制说明符。private 函数不能够被外部类访问，我们在本书讲解单类游戏的方法中，大部分用到的都是 private 类。
看下面这个示例，如果你把这个函数写在主时间轴时，我们就要去掉 private 关键字了。

```
private function myFunction(myNumber:Number,myString:String):Boolean{
if(myNumber==7)return true
if(myString.length<3) return true
return false
}
```

如果函数里面的参数是 7 或者字符串的位数小于三位，该函数值就返回 true。

1.7 测试和调试代码

本节主要讲述了在什么时候需要用到调试程序来调试代码，并对调试控制面板的功能进行了一一介绍。

在下面三种情况下需要调试代码：
第一，flash 在编译或运行代码时，出现了错误信息。例如变量拼写错误等，这时你就需要用到调试功能调试；
第二，程序没有按照预想的运行。例如，应该运动的对象没有运动，用户不能在文本中输入内容或者在游戏中，英雄的子弹命中敌人后不能被打死。像这类错误也需要用到调试功能。
第三，通过调试来改进你的代码。你可以通过调试来找到这个低效率的代码。

调试代码的几种方法

你可以通过几种不同的方法来调试你的代码，最简单的就是通过你的大脑计算出来，例如下面这几行代码：

```
var myNumber:int=7
myNumber+=3
myNumber*=2
myNumber++
```

上面，你不用运行代码就知道该变量结果是 21

如果代码比较长，计算比较复杂的话，用 trace()方法。可以在下面加入一行：

```
trace(myNumber)
```

用调试器来调试代码

例如，如果你编写了一组 if..else if 语句且无法确定在执行哪一条语句，那么，你就可以在语句前添加断点然后再在调试器中逐句检查这些语句（跟踪它们）。

设置断点

您可以在动作面板、“脚本”窗口或者“调试器”中设置断点。 在“动作”面板中设置的断点会保存在 FLA 文件中。 在调试器和“脚本”窗口中设置的断点不会保存在 FLA 文件中，并且只在当前的调试会话中有效。

重要说明： 如果在动作面板或“脚本”窗口中设置了断点并单击“自动套用格式”，请检查您的断点。如果“自动套用格式”命令删除了空行，则您的 ActionScript 可能已移到另一行。设置断点前最好对您的脚本执行自动套用格式操作。另要注意。不要在注释或空行上设置断点，在此处设置的断点将被忽略。

单步调试代码

在脚本中设置断点后单击调试器中的“继续”，便可以跟踪代码行，即控制调试器如何在语句和函数中移动。

例如，在下面的 ActionScript 3.0 代码中，如果将第一个断点设置在 13 行 showNumber(i) 代码上，调试影片时，flash player 会自动运行到 showNumber（i）时停止，如果单击“跳入”，调试器将进入第 17 行，再次单击“跳入”将进入第 18 行。

```
1.package
2.{
3.import flash.display.MovieClip;
4.import flash.text.TextField
5./**
6.@author ☆璀璨星晨☆
7.*/
8.public class DebugExample extends MovieClip
9.{
10.public function DebugExample()
11.{
12.for (var i:int = 0; i < 10; i++ ) {
13.showNumber(i)
14.}
15.trace(“循环体外”)
16.}
```

```
17.private function showNumber(whichNum:int):void
18.{
19.var myText:TextField = new TextField()
20.myText.text = String(whichNum)
21.myText.y = whichNum * 20
22.addChild(myText)
23.}
24.}
25.}
```

注： 此代码片断中的数字表示行号。 这些行号并非代码的一部分。

跳出 使调试器跳出函数。 只有当前在用户定义的函数中停止时，此按钮才能起作用；它将黄色箭头移到调用该函数的代码行后面一行。 在上面的示例中，如果在第 21 行中设置断点然后单击“跳出”，调试器将移动到第 15 行。 在用户定义的函数之外某一行处单击“跳出”的效果如同单击“继续”。

跳过 使调试器跳过一行代码。 此按钮将黄色箭头移动到脚本中的下一行。 在上面的例子中，如果在第 13 行停止，然后单击“跳过”，则会直接进入第 15 行，而不会跟踪 showNumber()，尽管 showNumber() 代码仍然会执行。

继续 离开播放器停止处的行并继续播放，直至到达一个断点。

结束调试会话 使调试器处于非活动状态，但是继续在 Flash Player 中播放 SWF 文件。

1.8 发布设置

路径：通过选择文件—>发布设置，在发布设置有格式，Flash 和 HTML 三个标签。

格式选项卡

发布设置决定发布 flash 动画时使用哪种导出格式。默认情况下 flash CS3 导出一个 swf 文件和一个包含 swf 的 HTML 文件；GIF，JPEG 和 PNG 为图片格式；windows 放映格式和 Macintosh 放映格式为嵌入 flash 播放器的 exe（Windows 系统）和 app（Macintosh 系统）。

Flash 选项卡

版本：用于指定发布影片的 Flash player 的版本

加载顺序：首帧所有层的下载方式，哪些内容将在影片下载时首先出现，该选项只影响首帧

ActionScript 版本：影片所用的 ActionScript 所用的版本，单击右侧的设置按钮，可以指定本影片所用的类文件包的路径，以及警告模式与严谨模式的设置。

生成大小报告：生成一个报告，按文件列出最终 Flash 内容中的数据量。

忽略 Trace 动作： 使 Flash 忽略当前 SWF 文件中的 Trace 动作 (trace)。 如果选择该选项，“跟踪动作”的信息不会显示在“输出”面板中。

防止导入： 防止其他人导入 SWF 文件并将其转换回 FLA 文档。 可使用密码来保护 Flash SWF 文件。

允许调试： 激活调试器并允许远程调试 Flash SWF 文件。 可让您使用密码来保护 SWF 文件。

压缩影片：

(默认)压缩 SWF 文件以减小文件大小和缩短下载时间。 当文件包含大量文本或 ActionScript 时，使用此选项十分有益。 经过压缩的文件只能在

Flash Player 6 或更高版本中播放。
针对 Flash Player 6 r65 优化： 如果在“版本”弹出菜单中选择了 Flash Player 6，则选择此选项可以将版本指定为 Flash Player 6。 更新的版本使用
ActionScript 注册分配来提高性能。 用户必须拥有 Flash Player 6 或更高版本。
导出隐藏的图层： （默认）导出 Flash 文档中所有隐藏的图层。 取消选择“导出隐藏的图层”将阻止把生成的 SWF 文件中标记为隐藏的所有图层（包括嵌套在影片
剪辑内的图层）导出。 这样，您就可以通过使图层不可见来轻松测试不同版本的 Flash 文档。
导出 SWC：
导出 .swc 文件，该文件用于分发组件。 .swc 文件包含一个编译剪辑、组件的 ActionScript 类文件，以及描述组件的其它文件。
JPEG 品质：flash 对文件中的位图图像使用 JPEG 格式进行压缩，压缩量越大，文件越小，图片质量越差，如果发布的影片中不包含位图图像，该设置无效。
音频流和音频事件：指定播放时的流式声音和事件声音的采样率和压缩方式。这些设置仅对影片中尚未指定事件属性的声音有效。如果勾选覆盖声音设置，则以上两项
设置对影片中的所有声音都有效。

HTML 选项卡
该选项卡用于生成包含 flash 影片的 HTML 文档，在所生成的文档中含有指定电影在浏览器窗口中的位置，背景色，影片尺寸等。

模板：指定使用的模板，可以开启版本检测全屏等功能
尺寸：可以选择匹配影片，像素和百分比等选项，并可以修改下面宽高的属性值
回放：
开始时暂停：开始时暂停影片播放，直到单击影片中的播放按钮，或者选择快捷菜单中的开始播放为止；
循环：设置影片是否循环播放
显示菜单：当用户用鼠标右击影片时快捷菜单是否可用，如果取消该选项，菜单中将只有“关于 Flash”选项；
设备字体：使用经过消除锯齿处理的系统字体替换那些用户系统中未安装的字体；
品质：通常质量越低播放速度越快
窗口模式：可以选择透明窗口和不透明窗口；
HTML 对齐：设定影片窗口在浏览器窗口中的位置；
缩放：定义影片在用户所指定的宽度和高度边界内如何放置
显示警告信息：设定 flash 是否显示错误信息警告有关标签设置冲突。

1.9 脚本游戏编程检验表

在创建 flash 游戏时，需要把好多的因素都要考虑进去。有时忘记一个关键的因素就会导致游戏不能够正确的运行。为了避免这些问题，下面有一个清单供大家参考：

1.发布文档设置
发布设置对话框和影片属性面板是容易忘记的关键因素；
1.1 文档类设置是否正确

前面我们讲到了怎样在舞台属性面板中设置文档类。如果有外部文档类没有在这里绑定的话，在测试影片时就不会访问到这个类文件；

1.2 发布设置是否正确

在本书中，要确保在发布设置中 Flash 的版本要选择 flash 9 和 ActionScript3.0，以便播放器能够正确的编译源文件。

1.3 检验本地回放安全性设置

在发布设置的 flash 标签中，有一个本地访问安全性设置，可以选择通过本地文件访问还是通过网络访问。为确保 flash 影片的安全可靠，你需要根据实际情况选择其中的一项。

2 类，函数和变量名的检验

在这里，即使你养成了良好的编程习惯，也经常会犯一些简单的错误。

2.1 注意区分大小写

当你在给变量，函数等命名时，一定要注意区分大小写。比如，myVariable 和 myvariable 就是两个完全不同的变量；同样，你如果命名了一个 myClass 类，那么，在调试影片时，软件将会自动调用名称为 myClass 的构造函数，此时，如果你不小心把该函数名写成了 myclass，那么在初始化时它就不会被调用。

2.2 影片剪辑类文件的绑定

如果一个影片剪辑需要在链接属性中绑定一个类时，他既可以选择默认的类名，也可以输入一个已创建的类文件名。例如：你有一个 EnemyCharacter 的影片剪辑，并且要绑定一个 EnemyCharacter.as 类文件。那么，你就可以在基类中输入该类的文件名（注意一定要包含路径哦～～）。一定要记住，这里经常容易出错哦。

2.3 类扩展类型是否正确

如下面代码所示：

public class myClass extends Sprite{

这里的扩展类型是选择 sprite 还是 movieClip，主要看这个影片主时间轴有几帧。如果是单帧的话，就使用 sprite，多帧就使用 movieclip。

2.4 构造函数明是否正确

如果你把类文件名命名为 myClass，那么相应的构造函数的名称也应该为 myClass，否则，在类初始化时构造函数不会被执行。

3 运行结果问题

在调试影片时，虽然有些问题不会在输出面板上显示出来，但是这个问题还是会在 flash 的制作过程中暴露出来。

3.1 访问空对象问题

这个问题比较棘手。主要来说，当影片跳转到下一帧后，就开始尝试着设置或者访问某个对象，如果该对象属性访问不到的话，就会生成一个错误提示。TooEarlyExample.fla 和 TooEarlyExample.as 举例来说（该文件下载地址见 AS3.0 游戏编程大学目录内链接）。测试影片后，fla 文件绑定的类文档让主时间轴跳转到第二帧，在第二帧中，有两个文本域，首先，软件会先尝试着设置或访问第一个文本域的属性，但是在运行时出现一个引用的错误(输出面板错误为：TypeError: Error #1009: 无法访问空对象引用的属性或方法。at TooEarlyExample$iinit())，因此，第一个文本不能够正确的被赋值。另当影片初始化完成后，在主时间轴内可以正常的调用第二个文本域的函数，并能够正常的显示出来。

3.2 正确的处理对象

完成你的 flash 制作后正确的处理对象，绝对是一个好习惯。例如，你在制作射击游戏时，

在一分钟内会有成百上千的子弹射出来，当他们离开屏幕后，还是保留在内存中的。为了移除这些对象，你需要在变量或数组中删除所有的对象，并用 removeChild 方法移出显示列表。

3.3 变量的类型要正确

当 int 或者 uint 变量类型可用时，就不要用 Number 类型，这样处理速度会更快一些。如果你在数组中定义成百上千个 Number 类型的数值，就会发现影片运行的相当慢。还有，比这更糟糕的就是使用未定义变量，他们在运行时占用很大的内存开销。另还需要注意，在创建影片剪辑时，能用 sprite 的就不要用 movieclip。

4.测试发布影片

下面的内容是关于你在测试发布影片时需要注意的事项

4.1 是否需要禁用快捷键

当你在测试影片时用到键盘输入的话，有时候你会发现部分按键不响应。这是因为当前的测试环境需要使用这些快捷键。换句话说，如果你想让发布到网页上的影片需要自定义快捷键的话，选择菜单“控制”—>“禁用快捷键”选项即可。

4.2 是否在其它帧频中测试过

如果你的动画是基于时间制作的话，帧频设置为 1—60 中的任何一个数值都没有问题，影片是以同一个速度运动。如果使用帧来生成动画（特别是当帧频比较低时 6 或 12），用户在不同运行速度的机器上看到的效果就不一样。

4.3 是否在服务器上测试过

当你在本地测试影片时，影片内连接的多媒体（比如声音，图片，视频等）立刻会加载完毕。但是，当你把影片上传到服务器上测试时，就需要等待几秒或几分钟的时间来加载这些内容。在这里，你在测试影片时，可以选择视图—>模拟下载设置选项，可以选择一个你想要的模拟下载速度。比如，你选择了 56K 后，这个影片在重新加载时就以你设置的 这个速度进行。新开始加载以你刚才设置的速率，当然，你也可以在服务器上加载运行。对这个问题，我们可以制作一个加载界面来等待加载完成，在第二章就会看到这方便的实例

以上代码检验清单主要目的就是，让你用最少的时间调试代码，把更过的时间用到游戏的设计中去。现在，我们已经介绍了 ActionScript3.0 的一些基础知识，下面的章节就开始来制作我们的游戏啦～～

ActionScript 游戏基础

2.1 创建可视对象

2.1.1 影片剪辑的应用

如果你在库中有一个影片剪辑，想把它添加到舞台上的话，有两种方法可供选择：

第一种方法

先把这个影片剪辑从库中拖放到舞台上，在属性面板上取一个实例名称，例如 myClipInstance，然后你就可以通过代码来控制舞台上这个影片剪辑的相关属性了。下面代码定义该影片剪辑的坐标为（300，200）

```
myClipInstance.x=300
myClipInstance.y=200
```

第二种方法

用纯代码来操作。首先，在库中选中影片剪辑，点击“右键”—>“链接”后出现链接属性对话框，选中“链接”—>“为 ActionScript 导出”，然后设置一个类名即可，例如 Mascot(注意，类名首字母一般为大写)。现在我们就用纯代码的形式把库中的这个影片剪辑添加到舞台上。对此，我们先声明一个变量作为该对象的实例，然后用 addChild 把它添加到显示列表。代码如下：

```
var myMovieClip:Mascot=new Mascot()
addChild(myMovieClip)
```

因为我们没有设置这个影片剪辑的任何属性，所以它默认在舞台上的(0,0)位置显示。我们可以用这个实例的 x，y 属性来设置它的坐标，用 rotation 属性来设置这个对象的旋转度数

```
var myMovieClip:Mascot=new Mascot()
myMovieClip.x=275
myMovieClip.y=150
myMovieClip.rotation=10
addChild(myMovieClip)
```

用上面的代码把影片剪辑添加到舞台上看起来比较繁琐，但是用代码可以很容易地生成多个此影片剪辑的副本，并把他们添加到舞台上。下面我们让 Mascot 影片剪辑生成 10 个副本，从左到右间隔 50 个像素水平排列，并把这个对象缩放为 50%。代码如下：

```
for(var i=0;i<10;i++){
var mascot:Mascot=new Mascot()
mascot.x=50*i+50
mascot.y=300
mascot.scaleX=.5
mascot.scaleY=.5
addChild(mascot)
}
```

测试上面两段代码，你可以看到，第一个对象 Mascot 在上面，坐标是(275,100),下面的 Mascots 生成了 10 个副本，横坐标从 50 到 500 依次展开，纵坐标是 300，缩放率为 50%.

2.1.2 按钮的制作

下面介绍制作按钮的三种方法

第一种方法：（把影片剪辑制作成按钮）

把影片剪辑制作成可点击的按钮，首先，你需要利用前面的知识把该影片剪辑添加到舞台上，然后在注册一个可接受鼠标事件的侦听器即可。下面的代码将库中类名为 Mascot 的影片剪辑放在舞台的(100,150)处

```
var myMovieClip:Mascot=new Mascot()
myMovieClip.x=100
myMovieClip.y=150
addChild(myMovieClip)
```

下面来注册这个侦听器，你需要用到 addEventListener()侦听函数，包括响应侦听事件的类型，该类型为常量，例如 MovieEvent.CLICK 将要响应一个鼠标点击事件，另外还需要包含一个处理响应事件的函数

```
myMovieClip.addEventListener(MouseEvent.CLICK,clickMascot)
function clickMascot(event:MouseEvent)
{
trace("你点击了 mascot")
}
```

测试代码时，这个 clickMascot 函数就会向输出窗口发送一个事件。但是，在大多数情况下，我们需要这个影片剪辑看起来更像按钮，这是，就用到了 buttonMode 属性。把它设置为 true 后，当你的鼠标划过这个影片剪辑的时，鼠标的箭头形状就会变成手型了。

```
myMovieClip.buttonMode=true
```

第二种方法：（将按钮添加到舞台）

当然，你还可以创建一个按钮实例，就像我们对影片剪辑的操作一样，在这里，把库中的按钮链接类名命名为 LibraryButton

```
var myButton.LibraryButton=new LibraryButton()
myButton.x=450
myButton.y=100
addChild(myButton)
```

这个按钮的形式与前面影片剪辑最大区别就是，双击按钮，可以看到，在时间轴上按钮有 4 个不同的帧，第一帧为鼠标划过之前显示的状态；第二帧为鼠标经过时显示的状态；第三帧为鼠标按下还未释放时显示的状态；最后一帧为按钮的点击区域，该区域在任何时候都是不可见的。下面，你还可以为这个按钮添加上侦听，代码如下：

```
myButton.addEventListener(MouseEvent.CLICK,clickLibraryButton)
function cliclLibraryButton(event:MouseEvent)
{
trace(你点击了 Library Button!)
}
```

第三种方法：

用 SimpleButton 类型创建，这时，你需要有 4 个大小一样的影片剪辑，链接类名为 ButtonUp，ButtonOver，ButtonDown，ButtonHit，然后用 SimpleButton 构造方法，把这四个

影片剪辑传递给 SimpleButton 的实例，代码如下：

```
var mySimpleButton:SimpleButton=new SimpleButton(new ButtonUp(),new ButtonOver(),new ButtonDown(),new ButtonHit())
mySimpleButton.x=450
mySimpleButton.y=250
addChild(mySimpleButton)
```

另外，还可以添加一个侦听函数

```
mySimpleButton.addEventListener(MouseEvent.CLICK,clickSimpleButton)
function clickSimpleButton(event:MouseEvent)
{
trace(你点击了 Simple Button!)
}
```

2.1.3 绘制图形

并不是所有舞台上的元素都是从库中拖放出来的，你可以用 AS3.0 代码直接画出线条或者一些基本的图形。在主时间轴上可以输入以下代码：

画直线，首要设置直线的粗细，颜色等样式，起始点和终止点。

```
this.graphics.lineStyle(2,0x000000)
this.graphics.moveTo(100,200)
this.graphics.lineTo(150,250)
```

上面代码设置了线的粗细为 2 个像素，颜色为黑色，线条从(100,200)到(150，250)。注意，this 关键字在这里不是必须的，当你想在一个确定的影片剪辑内画线时，也可以直接用这个对象的实例名称，例如

```
myMovieClipInstance.graphics.lineTo(150,250)
```

画曲线，首先需要确定起始点（如果没有起始点的话会从（0，0）位置开始绘制），终点和控制点。

```
this.graphics.curveTo(200,300,250,250)
```

然后，我们再用另一条直线来结束这个曲线

```
this.graphics.lineTo(300,200)
```

绘制矩形，需要确定矩形左上角的坐标，矩形的宽度和高度

```
this.graphics.drawRect(50,50,300,250)
```

绘制圆角矩形，在矩形的基础上再增加圆角的宽度和高度两个参数即可

```
this.graphics.drawRoundRect(40,40,320,270,25,25)
```

绘制圆形，需要中心点坐标和半径

```
this.drawCircle(150,100,20)
```

绘制椭圆 和绘制矩形需要的参数一样，需要确定椭圆注册点左上角的坐标，椭圆的宽度和高度

```
this.graphics.drawEllipse(180,150,40,70)
```

另外，还可以用 beginFill 填充颜色

```
this.graphics.beginFill(0x333333)
this.graphics.drawCircle(250,100,20)
```

并用 endFill 来结束填充
this.graphics.endFill()

.1.4 绘制文本和超链接文本

```
//在舞台上生成一个文本
var myText:TextField=new TextField()
myText.text="Click it Out"
addChild(myText)
//设置文本域坐标的位置
myText.x=50
myText.y=50
//设置文本域的宽高
myText.width=200
myText.height=30
//设置文本域的边框
myText.border=true
//设置文本域不可选状态
myText.selectable=false
//创建文本格式对象设置文本内的字体,大小以及样式等,你还可以用一行代码来创建文本格式 var myFormat:TextFormat=new TextFormat("Arial",24,0x000000,true)
var myFormat:TextFormat=new TextFormat()
myFormat.font="Arial"
myFormat.size=24
myFormat.bold=true
//可以用 setTextFormat 和 defaultTextStyle 两种方法来应用设置好的文本。
//myText.defaultTextFormat=myFormat
myText.setTextFormat(myFormat)
```

现在，你可以把上面的代码复制到你的 fla 文件的主时间轴上进行测试就可以了。

另外，我们在看一下创建文本格式的两种用法的区别，对于上面的代码，我们可以把它分为两部分，上面部分为文本赋值代码，下面部分为格式设置代码，对于此种情况，要让格式应用与文本的话，必须用 setTextFormat 方法才有效；如果如果格式设置代码在前，文本赋值代码在后，那么要用 defaultTextFormat 属性才行。另外还要注意 setTextFormat 为方法，defaultTextFormat 为属性。

创建超链接文本

其中最简单的方法就是用 htmlText 属性，在里面添加 HTML 代码，例如

```
var myWebLink:TextField=new TextField()
myWebLink.htmlText="点击后面链接访问我的博客<A HREF='http://xiaocui.blogbus.com'>Flash 脚本学习</A>"
addChild(myWebLink)
```

测试上面代码，我们发现链接能够正常使用了，我们想以下，如果让链接加上下划线，改变

字体的颜色应该怎么处理呢？我们来看下面的代码：

```
var myStyleSheet:StyleSheet=new StyleSheet()
//注意颜色值的格式为＃FFFFFF
myStyleSheet.setStyle("A",{textDecoration:"underline",color:"#0000ff"})
var myWebLink:TextField=new TextField()
myWebLink.width=300
myWebLink.styleSheet=myStyleSheet
myWebLink.htmlText="点击后面链接访问我的博客<A HREF='http://xiaocui.blogbus.com'>Flash脚本学习</A>"
addChild(myWebLink)
```

另外，我们还可以不用链接到网页窗口，还可以用侦听的方式，例如：

```
myLink.htmlText="Click<A HREF='event:testing'>here</A>"
addEventListener(TextEvent.LINK,textLinkClick)
function textLinkClick(event:TextEvent)
{
trace(event.text)
}
```

2.1.5 创建 sprite 组

这节，我们来看一下 sprite 作为容器的功能。

首先，先创建一个 sprite 容器，在容器内绘制一个 200×200 的矩形，这个矩形有 2 像素的边框，灰色填充。

```
var sprite1:Sprite=new Sprite()
sprite1.graphics.lineStyle(2,0x000000)
sprite1.graphics.beginFill(0xcccccc)
sprite1.graphics.drawRect(0,0,200,200)
addChild(sprite1)
```

现在，我们再把这个 sprite 容器移动到舞台坐标为（50，50）处

```
sprite1.x=50
sprite1.y=50
```

然后，我们再来创建第二个 sprite 容器，把它放置在舞台的（300，50）处，在该容器内也绘制一 200×200 的矩形，2 像素边框，灰色填充。代码如下：

```
var sprite2:Sprite=new Sprite()
sprite2.graphics.lineStyle(2,0x000000)
sprite2.graphics.beginFill(0xcccccc)
sprite2.graphics.drawRect(0,0,200,200)
sprite2.x=300
sprite2.y=50
addChild(sprite2)
```

最后，我们来创建第三个 sprite 容器，它包含一个圆形，我们把它添加到 sprite1 容器内，

并给它一个黑色填充，代码如下：

```
var sprite3:Sprite=new Sprite()
sprite3.graphics.lineStyle(2,0x000000)
sprite3.graphics.beginFill(0x333333)
sprite3.graphics.drawCircle(0,0,25)
sprite3.x=100
sprite3.y=100
sprite1.addChild(sprite3)
```

好了，把上面三段代码添加到 fla 文件主时间轴上测试一下。我们看到，在舞台上出现了刚才绘制的这三个图形。也发现，我们刚才设置了圆形的坐标为（100，100），但它并没有出现在舞台坐标的（100，100）处，这是什么原因呢？原来，我们把 sprite3 添加到 sprite1 后，sprite3 的坐标位置就是相对于 sprite1 的注册点的位置了。也就是说，子对象的坐标点是以其父容器为参照对象的。在上面的这个示例中，，sprite3 的坐标点（100，100）就是相对于它的这个父容器 sprite2 注册点的相对位置。

下面，我们在 sprite1 和 sprite2 中添加侦听函数，当你点击它们之中哪一个，sprite3 就设置为哪个容器的子对象。也就是说，你可以用鼠标点击来控制这个圆形在两个 sprite 容器间来回跳转。

```
sprite1.addEventListener(MouseEvent.CLICK,clickSprite)
sprite2.addEventListener(MouseEvent.CLICK,clickSprite)
function clickSprite(event:MouseEvent):void
{
   event.currentTarget.addChild(sprite3)
   //event.target.addChild(sprite3);
}
```

下面内容为个人理解所得：

****

如果想让鼠标移到 sprite1 和 sprite2 上时出现小手的形状，就用到前面学到的 buttonMode 的属性了，添加这句代码就可以实现：

```
sprite1.buttonMode=true
sprite2.buttonMode=true
```

2.1.6 设置层深

本节只要讲了用 setChildIndex（）方法来设置显示对象的层深。

setChildIndex 方法允许你向上或向下移动显示对象在显示列表内的位置。你可以把显示列表考虑成一个数组，它的索引位置是从第 0 层开始的。如果你创建了 3 个显示对象，那么他们的位置就是第 0，1，2 层。第二层的对象在外面，第 0 层的在最里面。

如果你想把某一个影片剪辑移动到所有显示对象的最里层，可以用

```
setChildIndex(myMovieClip,0)
```

执行这条语句之后，其余的对象将会自动往上提升一层，及他们的索引位置都加上了 1，原来第 0 层的元素移到了第 1 层，第 1 层的元素移到了第 2 层……

如果想把某一显示对象移到所有对象的上面，这时就要用到 numChildren 属性，它的含义就是该容器内显示对象的数目。比如现在某一容器内有三个显示对象，这个 numChildren 的值

就是 3，层次列表分别是第 0，1，2 层。那么，最外层的层深就是第 numChildren－1 层。
setChildrenIndex(myMovieClip,numChildren-1)
为了更好的理解设置层深的问题，书中作者给我们提供可一个实例 settingSpriteDepth.fla，用代码生成圆的三个副本，并把它交错叠放在舞台上，你可以点击其中的任何一个圆，让它显示在最外面（设置层深为最高）
下面为示例源代码：
****
简单介绍一下，首先在舞台上画一圆形，转化为影片剪辑，类名为 Circle，主时间轴内代码如下：

```
var circle1:Circle = new Circle();
var circle2:Circle = new Circle();
var circle3:Circle = new Circle();
circle1.x = 250;
circle1.y = 175;
circle2.x = 300;
circle2.y = 175;
circle3.x = 275;
circle3.y = 225;
addChild(circle1);
addChild(circle2);
addChild(circle3);
circle1.addEventListener(MouseEvent.CLICK, clickCircle);
circle2.addEventListener(MouseEvent.CLICK, clickCircle);
circle3.addEventListener(MouseEvent.CLICK, clickCircle);
function clickCircle(event:MouseEvent)
{
    setChildIndex(MovieClip(event.currentTarget),numChildren-1);
}
```

个人理解：看一下 clickCircle 函数里面的代码，在这里 event.currentTarget 指的是 MovieClip 类的显示对象，而不是显示对象容器。所以，这里的 currentTarget 也可以用 target 代替。
本

2.2.1 鼠标输入

前面已经了解到，我们可以把一个影片剪辑制作成按钮来响应鼠标事件，这就是鼠标输入的一种。除此之外，鼠标还可以做更多的事情，下面我们就用一个小实例来说明一下。

我们先用代码在舞台上绘制一个圆形和一个文本，当鼠标在舞台上划过时，在文本内可实时的显示出光标在舞台上的坐标，另外，当鼠标划过圆形时，该图形变形 100％的不透明，划出时变成 50％的透明度。

现在，我们先来分析一下上面的题目，在舞台上绘制圆形和文本，这时我们在前面学过的东西，在这里正好复习一下。另外，当鼠标滑过，划出时怎么控制？怎么得到实时看到光标在舞台上的坐标？这时，我们就用到了鼠标的另外两个事件划过为 MouseEvent.ROLL_OVER，划出为 MouseEvent.ROLL_OUT，取得鼠标位置的坐标就用到了 mouseX，mouseY 属性，有关透明度的就是 alpha 属性了。在这里，如果想要实时得到光标位置，还需要用到的就是帧频事件 Event.ENTER_RAME.

```
//先用前面学到的知识来创建一个文本，绘制一个黑色圆
var mouseLocText:TextField=new TextField()
addChild(mouseLocText)
var mySprite:Sprite=new Sprite()
mySprite.graphics.lineStyle(2,0x0000ff)
mySprite.graphics.beginFill(0x000000)
mySprite.graphics.endFill()
addChild(mySprite)
//添加实时显示鼠标位置的侦听函数
addEventListener(Event.ENTER_FRAME,showMouseLoc)
function showMouseLoc(event:Event)
{
mouseLocText.text="X="+mouseX+"Y="+mouseY
}
```

当鼠标划入时，该显示对象的实例变成 100％的不透明，即 alpha=1;当鼠标划出时，该对象变成 50％的透明度，即 alpha＝0.5.这时，我们需要用到鼠标的划过 MouseEvent.ROLL_OVER，划出 MouseEvent.ROLL_OUT 事件，代码如下：

```
mySprite.addEventListener(MouseEvent.ROLL_OVER,rolloverSprite)
function rolloverSprite(event:MouseEvent):void
{
mySprite.alpha=1
}
mySprite.addEventListener(MouseEvent.ROLL_OUT,rolloutSprite)
function rolloutSprite(event:MouseEvent):void
{
mySprite.alpha=.5
}
```

测试上面代码，我们可以看到， 这个圆形开始的时候是 50％的透明度，当鼠标划过的时候变成 100％的不透明。

其它相关的鼠标事件

单击 CLICK，双击 DOUBLE_CLICK,按下 MOUSE_DOWN,弹起 MOUSE_UP,划过 ROLL_OVER,划出 ROLL_OUT,移动 MOUSE_MOVE 等。

target 与 currentTarget 的区别

先测试一下原来的代码，无论你点击矩形还是圆形都没有错误提示。现在我们把 clickSprite 函数内的第一句注释掉，把第二句打开，然后再测试一下影片，鼠标点击矩形时不会出现问题，当鼠标点击到圆形上时就出现错误的提示，内容为：

```
ArgumentError: Error #2024: 不能将对象添加为其自身的子对象。
    at flash.display:isplayObjectContainer/addChild()
    at CreatingSpriteGroups_fla::MainTimeline/clickSprite()
```

这也就是说，在 CreatingSpriteGroups 文件主时间轴的 clickSprite 函数内出现了错误，内容是不能将对象添加为其自身的子对象。为什么会出现这种错误呢？翻开黑羽大哥的书找到了答案，^_^。target 表示发生事件的显示对象，而非容器；而 currentTarget 为当前侦听事件的节点，往往是容器。只有当添加事件侦听的显示对象和发生事件的显示对象为同一个时，currentTarget 才会于 target 相等。此时，currentTarget 才可能是非容器显示对象。那么，上

面的代码，我们就可以这样来理解，当鼠标点击到圆形上时，event.target 就指圆形这个显示对象，而 event.currentTarget 就指圆形这个显示对象的容器。显然，只有容器里面才能够存放显示对象，用 target 的话系统就会报错。

2.2.2 键盘输入

检测键盘输入是依靠 KEY_UP,KEY_DOWN 这两个键盘事件来完成的。当用户按下按键时，就会自动发送 KEY_DOWN 事件；这时，如果你为这个事件设定一个侦听函数的话，那么，就可以用这个函数来处理一些事情了。

在游戏的开发中，我们一般将事件的发送者设为舞台，即用显示列表中任何一个显示对象的 stage 属性来侦听键盘事件即可。这也是所谓的键盘全局监听。

另一定要注意，在使用键盘输入测试影片时，一定要在菜单控制中的禁用快捷键一项打上勾，否则 stage 监听不到快捷键的。代码格式如下：

```
stage.addEventListener(KeyboardEvent.KEY_DOWN, keyDownFunction);
```

当这个侦听函数 keyDownFunction 被调用后，你就可以来使用键盘事件的一些属性了，有表示按键信息的 keyCode，charCode，类型为 uint 型；表示辅助键是否按下的 altKey，ctrlKey，shiftKey，类型为 Boolean 型；还有表示按键区域的 keyLocation，该值为正整数值。

下面简单介绍一下比较常用的 keyCode 和 charCode 属性：

keyCode 属性：该属性记录的值是一个正整数，对应着标准键盘上的一个键，该正整数为键控代码。

CharCode 属性：该属性表示键盘输出字符的 ASCII 码，该数值也为正整数。

上面的定义也许比较抽象，现在，来举例说明一下，当我们在键盘上按下【A】时，屏幕上会出现 a，这时，keyCode 的值为 65，charCode 的值为 97；当我们按下 shift＋A 时，屏幕上会出现 A，keyCode 值为 65 保持不变，charCode 的值变为 65。为什么会是这样的数呢？我们先来查一下 ASCII 代码表，a 的 ASCII 为 97，A 的 ASCII 为 65。因为按下的都是【A】键，所以键控代码（keyCode）不变（shift 为辅助键）；屏幕输出为 a 时，ASCII 码就显示 97，屏幕显示为 A 时，ASCII 码就显示 65。

根据上面的规律，我总结如下：charCode 与屏幕输出的字符相关，区分大小写；keyCode 与被按下的按键相关，每一个按键都有一个不同的键控代码。

另外，在这里也说一下 String（字符串）的 fromCharCode 属性，它是把 ASCII 代码值转换为所表示的字符。

代码演示：

```
stage.addEventListener(KeyboardEvent.KEY_DOWN, keyDownFunction);
function keyDownFunction(event:KeyboardEvent)
{
    trace("Key Pressed:"+String.fromCharCode(event.charCode)+"\tcharCode:"+
String(event.charCode)+"\tkeyCode:"+ String(event.keyCode));
}
```

把上面代码输入主时间轴内，来按住【A】和【SHIFT】＋【A】来验证一下吧：

输出结果如下：

```
Key Pressed:a    charCode:97 keyCode:65
Key Pressed:A    charCode:65 keyCode:65
```

在游戏设计过程中，我们不用担心按键是否被按下，更关心的是这个键一直被按着。比如，我们在玩驾驶游戏过程中，怎样通过向上的方向键来控制汽车油门的操作。这时，就需要我们给按键设置一个布尔值，按下时为 true，弹起时为 false。不管在什么时候，我们只需要通过检测这个布尔值就知道键盘是否被按下了。

下面，我们就用这种方法来检测一下空格键是否被按下。第一个函数，把空格键按下的状态设为 true，第二个函数，把空格键弹起的状态设置为 false。

```
var spacebarPressed:Boolean;
stage.addEventListener(KeyboardEvent.KEY_DOWN, keyDownFunction);
function keyDownFunction(event:KeyboardEvent)
{
    trace("Key Pressed:"+String.fromCharCode(event.charCode));
    if (event.charCode == 32)
    {
      spacebarPressed = true;
      trace("空格键被按下");
    }
}

stage.addEventListener(KeyboardEvent.KEY_UP, keyUpFunction);
function keyUpFunction(event:KeyboardEvent)
{
    if (event.charCode == 32)
    {
      spacebarPressed = false;
      trace("空格键被弹起");
    }
}
```

GPU2.2.3 文本输入 - [FLASH 脚本]

TextField 对象有一种类型为输入文本，它与静态文本和动态文本的区别就是，输入文本是可选的，并且还可以往里面输入内容。下面，我们来创建一个输入文本：

```
var myInput:TextField = new TextField();
myInput.type = TextFieldType.INPUT;
addChild(myInput);
```

这样创建的文本在屏幕的左上角，很难找到，也没有边框，我们可以用 TextFormat 属性做一些改进。

下面的代码把字体设置为 12 号，位置（10，10），高度 18.宽度 200，有边框

```
var inputFormat:TextFormat = new TextFormat();
inputFormat.font = 'Arial';
inputFormat.size = 12;
var myInput:TextField = new TextField();
```

```
myInput.type = TextFieldType.INPUT;
myInput.defaultTextFormat = inputFormat;
myInput.x = 10;
myInput.y = 10;
myInput.height = 18;
myInput.width = 200;
myInput.border = true;
addChild(myInput);
stage.focus = myInput;
```

最后一行的代码设置文本输入光标在文本域内。

文本域默认为单行，你可以用 multiline 属性改变它的状态。但是，对于大多数输入文本来说，我们只用一行就可以了，注意，在这里输入文本后按回车键不会换行。但是，我们可以用这个键来捕获信号，即当该按键（字符码为 13）被按下或弹起时来触发侦听函数实现一定的功能。

```
myInput.addEventListener(KeyboardEvent.KEY_DOWN, checkForReturn);
function checkForReturn(event:KeyboardEvent) {
if (event.charCode == 13) {
acceptInput();
}
}
```

这个 acceptInput 函数把文本内的字符寸锉刀 theInputText 字符串变量中，然后在输出窗口中显示出来，谭后在一处这个文本，代码如下：

```
function acceptInput() {
var theInputText:String = myInput.text;
trace(theInputText);
removeChild(myInput);
}
```

根据本人测试结果，键盘事件为 KEY_DOWN 可以输出结果，但 KEY_UP 却不能显示结果。百思不得其解，请高手指点。（在这里感谢 back1994 的帮助，这个问题可能是我软件的问题。）

2.3 创建动画

接下来，我们就开始来创建动画了，并用他们来模拟真实世界的运动。

2.3.1 帧频事件运动

我们通过设置影片剪辑的 x，y 的坐标，来改变他们在屏幕的位置是非常容易的。如果我们想使这个影片剪辑运动起来，并给它设置一特定的速度运动，这时就需要用到

ENTER_FRAME 事件了。例如，你可以在舞台上绘制一个任意图形，并转换为影片剪辑，类名为 Hero，下面，我们就通过一小段程序复制库中 Hero 的一个副本，并且让他每帧移动一个像素，代码如下：

```
var hero:Hero = new Hero();
hero.x = 50;
hero.y = 100;
addChild(hero);
```

```
addEventListener(Event.ENTER_FRAME, animateHero);
function animateHero(event:Event) {
hero.x++;
}
```

现在，测试影片可以看到，这个 hero 角色开始从屏幕坐标（50，100）处每帧以一个像素的速度水平向右移动，如果想让该影片剪辑每次移动 10 个像素的话，代码改为

```
hero+=10
```

注意：你还可以通过改变舞台属性面板上的帧频来控制对象的运动速度，系统默认值为每秒 12 帧，你可以更改该值为[1,60]内的任意数。事实上，你如果选择了帧频为 60 的话，并不意味着影片每秒运行 60 帧，如果该影片剪辑比较大，并且你的机器比较慢，那么，他将达不到 60 帧。对此，我们有一个更好的办法来解决这个差距，就是用下面介绍的基于时间的动画来处理。

下面，来试作一个练习。在游戏的过程中，我们经常遇到任务行走，这个问题的 flash 中怎么解决的呢？不必担心，其实这个很简单，首先用动画制作的方法制作一个影片剪辑，该影片剪辑有 8 帧，从第二帧到第八帧为步行的一个循环，每帧代表不同的步进，第一帧作为站立的位置被保留。制作完成以后，在时间轴内输入下面的帧频事件和帧频函数，在函数内，我们将要创建该角色每帧在水平位置移动 7 像素，后面的条件代码是检测该影片剪辑的当前帧数，如果为 8 帧的话，它将要返回的第二帧继续运动，否则，它继续下面的一帧。源代码如下：

```
var hero:Hero = new Hero();
hero.x = 100;
hero.y = 200;
addChild(hero);
addEventListener(Event.ENTER_FRAME, animateHero);
function animateHero(event:Event)
{
    hero.x += 7;
    if (hero.currentFrame == 8)
    {
        hero.gotoAndStop(2);
    } else
    {
        hero.gotoAndStop(hero.currentFrame+1);
    }

}
```

2.3.2 用计时器控制影片的运动

计时器，就好比我们使用的钟表。你还可以创建一个计时器，每秒用它来调用一个函数。当你声明一个计时器后也就创建了一个 Timer 对象，你需要设置时间间隔的毫秒数，还有循环次数，为 0 的话就是循环无数次。

下面代码创建一个 Timer 对象，每隔 1S 就触发调用一次 timerFunction 函数，源代码如下：

```
// 创建一个 Timer 对象，每秒调用一次 timerFunction 函数
var myTimer:Timer = new Timer(1000);
myTimer.addEventListener(TimerEvent.TIMER, timerFunction);
// 函数内为画一黑色填充圆点
function timerFunction(event:TimerEvent)
{
    this.graphics.beginFill(0x000000);
    //currentCount 为计时器从 0 开始后触发的总次数。 如果已重置了计时器，则只会计入重置后的触发次数。
    this.graphics.drawCircle(event.target.currentCount*10,100,4);
}
// 计时开始
myTimer.start();
```

上面代码源文件见 UsingTimers.fla。

另外，你还可以用计时器来完成上节用帧频事件做的动画，全部代码如下：

```
var hero:Hero = new Hero();
hero.x = 100;
hero.y = 200;
addChild(hero);
var heroTimer:Timer = new Timer(80);
heroTimer.addEventListener(TimerEvent.TIMER, animateHero);
function animateHero(event:Event)
{
    hero.x += 7;
    if (hero.currentFrame == 8)
    {
        hero.gotoAndStop(2);
    } else
    {
        hero.gotoAndStop(hero.currentFrame+1);
    }
}
heroTimer.start();
```

我们发现，调用函数内部的代码没变，唯一不同的就是侦听器不一样。前面为帧频侦听，这里为计时器侦听。当你测试影片时，会发现这个人物运动的速度和上面的那么差不多。如果你再重新设置一下舞台的帧频，比如改为 6 或者 60，会发现该影片运动的速度没有明显的变化。

另：具体区别就是 ENTER_FRAME 是按帧发生的,Timer 是你可以自定义的时间和循环次数.

2.3.3 基于时间的帧频动画

该方法的原理就是，首先声明一个整型变量来存储初始化 flash player 后经过的时间，然后再用侦听事件来调用侦听函数，在侦听函数内计算出当前帧与上一帧的时间差，再乘以一个比例系数应用到影片剪辑的坐标，从而改变该对象的运动。

首先，现在舞台上绘制一个小球，转换为影片剪辑，并把它从库中拖放到舞台的左边，实例名称为 ball。

打开主时间轴－>动作，在脚本编辑区内，输入下面的代码。先声明一个变量来存储初始化 flash player 后经过的时间，该时间以毫秒为单位的，再通过 getTimer()函数，把返回的时间传递給刚才声明的变量 lastTime。

getTimer()函数，是用来返回自 SWF 文件开始播放时起已经过的毫秒数。

```
var lastTime:int=getTimer()

// trace("lastTime:"+lastTime);
```

然后，我们用 ENTER_FRAME 事件来侦听，并声明一个 animateBall 侦听函数来调用。

```
addEventListener(Event.ENTER_FRAME, animateBall);
```

在下面的这个函数内，第一行代码：先用 getTimer（）函数得到系统当前帧的时间，并把该时间与上一帧的时间差赋值給一个整型变量 timeDiff; 第二行代码：更新 lastTime 时间，供下一次侦听函数被调用时使用；第三行代码：把该时间差乘以一个 0.1 系数作为小球横坐标的变化位置。比如，先在舞台属性中设置帧频为 10，也就是说，影片在运行时，每帧所用的时间大概是 100ms，比如 flash player 初始化经过的时间为 5ms，则 lastTime 初始值就为 5；当运行到下一帧时将触发侦听器调用 animateBall 函数，也就是说，其中间隔了 100ms，即 timeDiff 为 100ms，更新后的 lastTime 为 105ms，下面小球水平位置增加了 10 像素。也就是说，每经过一秒钟小球的水平位置就向右移动了 100 像素。

```
function animateBall(event:Event) {

// trace("getTime:"+getTimer());

var timeDiff:int = getTimer()-lastTime;

lastTime += timeDiff;

ball.x += timeDiff*.1;

}
```

下面，我们把上面的两行注释打开，测试影片。输出结果如下：

lastTime:6

getTime:67

getTime:167

getTime:267

getTime:367

getTime:470

getTime:577

通过上面代码我们可以验证，影片每一帧所运行的时间大概是 100ms，但不是那么准确。也就是说，用帧频事件做时间计时器的话，只能粗略的计算。我自己认为帧频事件的快慢与电脑的性能等相关。如果精确计时的话，就用函数获取系统的时间来计时。

另外，还可以尝试改变一下影片的帧频，改为 12 或者 60，看一下小球的运行速度和效果有什么不同？

2.3.4 基于物理学的运动

用代码来制作动画，你不仅能够让物体沿着预设的轨迹运行，还能够赋予它一些物理属性，让他更像自然界的一个实物。关于物理学的运动，可以是基于帧频的还可以是基于时间的。

下面，我们来创建一个基于时间的动画，用速度和重力加速度来控制物体的运动。

注：重力加速度是一个相对于地面恒定的加速度（在这里，地面就是屏幕的底部）。在现实中，重力加速度是 9.8m/s,而在 flash 世界中，每个事物都是用每毫秒多少像素来计算的。例如，1 像素代表 1m 的话，那么 0.0098 是 0.0098 米/毫秒，也就是 9.8 米/秒。在这里， 我们仅仅是用来做游戏，而不是建造严谨的物理模拟运动，简单的用 0.001 或其他合适的数值表示就可以了，只要运动的状态看起来比较像就可以了。

下面，我们来做一个实例，和上节一样，首先在舞台上绘制一个小球，转换为影片剪辑并拖放到舞台的左下角，实例名称为 ball。

这里，我们设置这个重力加速度为.00098，并且用 dx，dy 分别表示小球水平和垂直方向的初始速度分量.

//设置重力加速度

```
var gravity:Number=.00098

//设置初始速度

var dx:Number=.2

var dy:Number=-.8
```

为了让这个小球运动起来，我们来创建一个 ENTER_FRAME 侦听，并把 flash player 初始化时间赋值给一个变量。

```
// 声明一个整型变量存储初始化时间，添加侦听

var lastTime:int = getTimer();

addEventListener(Event.ENTER_FRAME, animateBall);

// 创建动画

function animateBall(event:Event) {

// 获取时间间隔

var timeDiff:int = getTimer()-lastTime;

lastTime += timeDiff;
```

dy 变量是垂直方向上的，他与重力的方向相反。

```
dy+=gravity*timeDiff
```

小球的移动位置由 dx，dy 这两个速度决定。在下面两行代码中，时间差用来计算出需要移动几个像素的距离。

```
ball.x += timeDiff*dx;

ball.y += timeDiff*dy;

}
```

2.4.1 用方向键控制影片的移动

用鼠标或者键盘来控制影片的运动是非常常见的。在本章的前面部分，我们学习了怎样判定空格键是否被按下。同样，我们还可以用这种方法来判定方向键是否被按下。虽然方向键没有显著的特征，但我们可以用它们各自的键控代码来表示：37（左），38（上），39（右），

40（下）。

同样，现绘制一个影片剪辑拖放到舞台，实例名称为 mascot。

首先，我们先声明 4 个布尔型变量来存储 4 个方向键的初始状态

```
var leftArrow:Boolean = false;
var rightArrow:Boolean = false;
var upArrow:Boolean = false;
var downArrow:Boolean = false;
```

然后，用 KEY_DOWN，KEY_UP 和 ENTER_FRAME 侦听器来处理影片的运动。

```
// 设定事件侦听
stage.addEventListener(KeyboardEvent.KEY_DOWN, keyPressedDown);
stage.addEventListener(KeyboardEvent.KEY_UP, keyPressedUp);
stage.addEventListener(Event.ENTER_FRAME, moveMascot);
```

当用户按下任意一个方向键时，我们设定它的布尔值变量为真 true

```
// 按下时把方向变量 boolean 值设置为 true
function keyPressedDown(event:KeyboardEvent) {
if (event.keyCode == 37) {
leftArrow = true;
} else if (event.keyCode == 39) {
rightArrow = true;
} else if (event.keyCode == 38) {
upArrow = true;
} else if (event.keyCode == 40) {
downArrow = true;
}
}
```

同理，用户松开方向键后，我们把 boolean 变量设置为 false

```
function keyPressedUp(event:KeyboardEvent) {
if (event.keyCode == 37) {
leftArrow = false;
} else if (event.keyCode == 39) {
rightArrow = false;
} else if (event.keyCode == 38) {
upArrow = false;
} else if (event.keyCode == 40) {
downArrow = false;
}
}
```

现在，我们用这些布尔变量作为影片运动的条件，当某一个方向键被按下时，就给影片赋予相应方向的一个速度，代码如下：

```
function moveMascot(event:Event) {
var speed:Number = 5;
if (leftArrow) {
mascot.x -= speed;
}
```

```
if (rightArrow) {
mascot.x += speed;
}
if (upArrow) {
mascot.y -= speed;
}
if (downArrow) {
mascot.y += speed;
}
}
```

保存测试影片，现在你就可以通过键盘上的四个方向键来控制影片的左右上下运动了。在代码中，每个方向键的布尔值变量都是分开设置的，但你可以把方向键组合来用。例如，你可以同时按下向右键和向下键，这时，影片就向右下方运动了。如果，你同时按住左方向键和右方向键时，影片就会静止不动了，因为它们左右速度叠加后正好为 0。

2.4.2 拖拽影片

上一节讲了用键盘方向键控制影片的移动。现在，我们再来看一下移动影片的另一种方法，实现：用户鼠标点击影片时，开始拖拽；用户释放鼠标时，停止拖拽。当鼠标点击影片时，可以在该影片上侦听 MOUSE_DOWN 事件，但是释放鼠标时我们不能把该影片作为侦听对象，而是要用舞台代替才行，只有这样，当鼠标不论在不在影片上面，只要侦听到 MOUSE_UP 事件，就能够停止拖拽了。代码如下：

```
// 设置侦听
mascot.addEventListener(MouseEvent.MOUSE_DOWN, startMascotDrag);
stage.addEventListener(MouseEvent.MOUSE_UP, stopMascotDrag);
mascot.addEventListener(Event.ENTER_FRAME, dragMascot);
```

另一个因素就是光标的偏移量，我们允许鼠标从影片的任何一点拖拽。首先，我们需要先计算出鼠标点击影片上的点相对于影片中心点的局部坐标，并把该点存储到 clickOffset 变量中，我们还可以用这个变量计算此刻是否有拖拽产生，如果是，clickOffset 将要设置成一个点，否则，就把 null 赋值給它。

```
// 鼠标点击的位置相对于影片的偏移量
var clickOffset.Loint = null;
```

当用户点击影片时，这个偏移量由点击事件的 localx，localy 获取

```
// 鼠标点击时
function startMascotDrag(event:MouseEvent) {
clickOffset = new Point(event.localX, event.localY);
}
```

当鼠标释放时，该点被置为 null

```
// 鼠标释放时
function stopMascotDrag(event:MouseEvent) {
clickOffset = null;
}
```

对于帧频事件调用的 dragMascot 函数，如果这个 clickOffset 不是空值，我们将要设置这个影片的位置为当前光标的位置减去偏移量。

```
// 帧频动画
function dragMascot(event:Event) {
if (clickOffset != null) {
mascot.x = mouseX - clickOffset.x;
mascot.y = mouseY - clickOffset.y;
}
}
```

测试影片，用鼠标点击这个影片然后从不同点拖拽，看一下这个 clickOffset 是怎么样处理的。

另外，也可以用 startDrag（），stopDrag（）方法实现开始停止拖拽过程，代码如下：

```
mascot.addEventListener(MouseEvent.MOUSE_DOWN, startMascotDrag);
stage.addEventListener(MouseEvent.MOUSE_UP, stopMascotDrag);

function startMascotDrag(event:MouseEvent):void
{
    mascot.startDrag();
}
function stopMascotDrag(event:MouseEvent):void
{
    mascot.stopDrag();
}
```

上面代码可以实现前面相同的效果。

2.4.3 碰撞检测

在游戏中，当有很多对象在屏幕上移动时，检测他们之间的相互碰撞是非常有用的。AS3.0 包含两个碰撞检测函数，hitTestPoint 函数用于检测某个点与显示对象间是否发生了碰撞；hitTestObject 函数用于检测两个显示对象间是否发生了碰撞。

具体来说，用 hitTestObject 函数作为影片的方法时，要将另一个影片的引用作为参数传入，格式如下：

```
sprite1.hitTestObject(sprite2)
```

如果两个影片发生了碰撞则返回 true，否则，返回 false。但是这种检测方法的精度很低，主要用来检测矩形影片的碰撞；hitTestPoint 函数用来检测两个影片是否相碰时，需要先定义两个 Number 类型的参数，作为检测点。根据影片是否与该点相交，返回 true 或者 false。另外第三个参数为可选参数，其值为 boolean 类型，默认为 false，检查影片的边框是否与该点相碰；如果设置为 true 的话，则用来检查影片的实际形状与该点是否相碰。

下面，我们通过一个实例来验证这两个函数的使用方法，首先，在舞台上绘制两个动态文本，分别命名为 messageText1，messageText2；然后再绘制一个月牙形，一个星形影片剪辑，实例名称分别为 crescent，star。代码如下：

```
addEventListener(Event.ENTER_FRAME, checkCollision);
```

```
function checkCollision(event:Event)
{
    // 检测鼠标指针是否与月牙形影片相碰
    if (crescent.hitTestPoint(mouseX, mouseY, true))
    {
        messageText1.text = "hitTestPoint: YES";
    } else
    {
        messageText1.text = "hitTestPoint: NO";
    }
    // 把星形坐标作为鼠标跟随
    star.x = mouseX;
    star.y = mouseY;

    // 检测星形与月牙形是否相碰
    if (star.hitTestObject(crescent))
    {
        messageText2.text = "hitTestObject: YES";
    } else
    {
        messageText2.text = "hitTestObject: NO";
    }
}
```

2.5 访问外部数据

有时候访问游戏的外部信息是非常必要的，比如我们在游戏中需要访问或加载外部网页或文本域内的参数，或者把相关的信息保存为本地文件。

2.5.1 外部变量

假如你有一个游戏是根据一些选项来改变的。例如，在 jigsaw puzzle 游戏运行时需要加载不同的图片，或者一个 arcade 游戏以不同的速度运行。你可以从 flash 影片所在的 HTML 页面中获取一些参数。在这里有多种方法能够实现，如果你在发布设置中默认 HTML 为模板的话，就可以通过 AC_FL_RunContent 函数中的 flashvars 属性来传递参数。

下面是一个简短的 AC_FL_RunContent 函数，在 flash cs3 导出影片后，你可以在 html 文件夹中发现这些代码。在这里我增加了 flashvars 参数

```
<script language="javascript">
AC_FL_RunContent(
'codebase',
'http://download.macromedia.com/p ... flash.cab#version=9,0,0,0',
```

‘width’, ‘550’,
‘height’, ‘400’,
‘src’, ‘ExternalVariables’,
‘quality’, ‘high’,
‘flashvars’, ‘puzzleFile=myfilename.jpg&difficultyLevel=7’);
</script>
这个 flashvars 属性包括一对值，它们之间用&符号隔开。在这里，puzzleFile 设置为 myfilename.jpg, difficultyLevel 设置为 7.
当 flash 影片开始运行时，将通过 LoaderInfo 方法来调用这些信息。下面这行代码将检索所有的参数并把它们赋值給声明的 paramObj 变量
var paramObj:Object = LoaderInfo(this.root.loaderInfo).parameters;
若访问单个属性值，你需要用下面的代码：
var diffLevel:String＝paramObj["difficultyLevel"]
另外，你还可以传递游戏的其它常量，例如图片的名字，开始等级，速度，位置等等。

读读源文件：
//加载外部参数信息
var paramObj:Object = LoaderInfo(this.root.loaderInfo).parameters;
//读取外部 diffLevel 值，并赋值給 messageStrng 变量
var diffLevel:String = paramObj["difficultyLevel"];
var messageString = "difficultyLevel: "+paramObj["difficultyLevel"];
messageString += "\n";
//然后在读取 puzzleFile 的值，和 diffLevel 相加后赋值给文本
messageString += "puzzleFile: "+paramObj["puzzleFile"];
messageText.text = messageString;
当运行这个影片 ExternalVariables.fla，它就会加载 ExternalVariables.html 文件。

2.5.2 加载数据

在 AS3.0 中，加载外部文本文件是相当容易的，尤其是 XML 格式的文件。例如，你想加载一个琐碎的问题文件，那么，这个 XML 文件内容可以这样写：
<LoadingData>
<question>
<text>This is a test</text>
<answers>
<answer type=”correct”>Correct answer</answer>
<answer type=”wrong”>Incorrect answer</answer>
</answers>
</question>
</LoadingData>

为了加载这些数据，你需要用到 URLRequest 和 URLLoader 两个函数，另外，当信息加载完成后还需要添加一个侦听函数还调用相关的数据。代码如下：
var xmlURL:URLRequest = new URLRequest(“LoadingData.xml”);
var xmlLoader:URLLoader = new URLLoader(xmlURL);
xmlLoader.addEventListener(Event.COMPLETE, xmlLoaded);

xmlLoaded 函数里面是一些 trace 语句，在输出面板上显示传递进来的数据

```
function xmlLoaded(event:Event) {
var dataXML = XML(event.target.data);
trace(dataXML.question.text);
trace(dataXML.question.answers.answer[0]);
trace(dataXML.question.answers.answer[0].@type);
}
```

以上，你可以看到，从 xml 文件内获取数据是非常简单的，首先把加载完成的数据赋值給 xml 对象的实例 dataXML，然后用 dataXML.question.text 来检索这个 question 文本，用 dataXML.question.answers.answer[0]来检索第一个答案，还可以用@type 获取这个 answer 的类型。

2.5.3 保存本地数据

在游戏的制作过程中，经常需要考虑到在用户计算机上读取和存储相关的数据。例如，在玩游戏过程中，我想存储游戏的分数，或者存储一些游戏的选项。这时，为了实现上面的功能，我们将用到本地共享对象，它类似于浏览器的 Cookie。我们需要调用 SharedObject.getLocal（）以在应用程序中创建共享对象，如带记忆功能的计算器。用户关闭计算机时，flash player 将在用户计算机的共享对象中保存最后一个值，计算机下一次运行时，将包含先前所拥有的值。注意，该对象只能用于当前客户端，以下代码显示如何将返回的共享对象引用赋值给变量：

```
var myLocalData:SharedObject = SharedObject.getLocal(“mygamedata”);
```

如果你存储了一些数据，你可以通过下面的代码来访问：

```
trace(“Found Data: “+myLocalData.data.gameinfo);
```

下面，来设置一下这个 gameinfo 的属性值

```
myLocalData.data.gameinfo = “Store this.”;
```

测试上面代码，或者打开源文件 SavingLocalData.fla 测试影片，在第一次运行时你会发现在输出面板上显示 undefined，因为在 trace 之前还没有对 myLocalData.data.gameinfo 进行赋值，不过在第二次测试时，就会在输出面板上显示 Store this.作为返回的结果了。

2.6 其它游戏元素

这里有一些简单的代码块来完成一系列的任务。你可以根据需要把它们添加到你设计的游戏中去。

2.6.1 自定义光标样式

为了满足游戏的不同风格，你可以用自定义影片来替换你的光标。例如，你想在游戏中运用大光标，或者在射击游戏中需要用十字准线光标。虽然，你不能够改变计算机的光标，但是在视觉上，你可以把它设为不可见。然后，再用一个自定义影片代替，并让它浮在所有显示对象的上面。

首先，现在舞台上拖放一个按钮，放置在第一层，另外再绘制一个方向键图形，转换为影片剪辑，放置在第二层，实例名称为 arrow。

把鼠标光标设置为不可见，用 Mouse.hide()命令

```
Mouse.hide()
```

然后，先确保把自定义影片放置在所有可显示对象的上面，如果你用脚本创建对象并添

加到屏幕上的话，必须通过 setChildIndex 命令把自定义影片放置在所有显示对象的顶部。为了让这个自定义影片跟随鼠标，还需要用到 ENTER_FRAME 侦听器

```
addEventListener(Event.ENTER_FRAME, moveCursor);
function moveCursor(event:Event) {
arrow.x = mouseX;
arrow.y = mouseY;
}
```

另外，还需要把这个自定义光标的 mouseEnabled 属性设置为 false，表示该对象不接收鼠标事件。默认值为 true，表示该对象接受鼠标事件。

```
arrow.mouseEnabled = false;
```

测试上面代码，可以看到一个真正的自定义鼠标就做成了。

2.6.2 加载影片

flash 是一种流媒体，也就是说，只要缓冲完少许的动画就能够直接播放了。这个功能非常好，比如你手工制作了 1000 帧的动画，当影片下载完前面的几十帧就能够正常播放了，在播放过程中继续下载后面的部分。但是，对 flash 制作的游戏来说，这种方法却行不通，因为游戏的每个元素都是用代码直接控制的，有一个元素未加载完成的话，这个游戏就不够正常的运行。

因此，大多数游戏都使用加载影片的功能强制等到所有元素都加载完成后才能够开始，等待过程用加载进度条等显示加载信息。

一个简单地方法就是，在第一帧把影片设置为 stop，

```
stop()
```

然后，再添加一个 ENTER_FRAME 侦听，在每一帧调用 loadProgress 函数

```
addEventListener(Event.ENTER_FRAME, loadProgress);
```

这个函数用 this.root.loaderInfo 获取影片加载的状态，他有 bytesLoaded 和 bytesTotal 两个属性，然后，我们再把这两个属性值除以 1024 转换成千字节。代码如下：

```
function loadProgress(event:Event) {
// 获取下载总字节和已下载字节
var movieBytesLoaded:int = this.root.loaderInfo.bytesLoaded;
var movieBytesTotal:int = this.root.loaderInfo.bytesTotal;
// 转换成千字节
var movieKLoaded:int = movieBytesLoaded/1024;
var movieKTotal:int = movieBytesTotal/1024;
```

为了让玩家能够看到下载进度，我们还需要在舞台上绘制一个文本域，显示格式为：Loading：5K/32K

```
//显示下载进度
progressText.text = “Loading: “+movieKLoaded+”K/”+movieKTotal+”K”;
```

当已下载字节与影片总字节相等时，我们就移除这个帧频侦听，并把影片跳转到第二帧

```
// 加载完成移除侦听
if (movieBytesLoaded >= movieBytesTotal) {
removeEventListener(Event.ENTER_FRAME, loadProgress);
gotoAndStop(2);
}
```

}

源文件 LoadingScreen.fla 中，在第一帧输入上面的代码，在第二帧添加一个 33K 的图片。然后选择控制－－测试影片，这时会立即看到第二帧的内容。在测试环境中，选择视图－－下载设置为 56K，然后再选择视图－－模拟下载，你就会看到这个下载进度了。

2.6.4 随机数字

几乎在任何游戏中都会用到随机数字。

在 AS3.0 中，用 Math.random()函数创建随机数，返回值是一个大于等于 0 小于 1 的数。例如：

```
var random1:Number=Math.random()
```

通常，我们用到明确的随机数范围，下面代码随机生成介于 0 和 10 之间的数

```
var random2:Number=Math.random()*10
```

如果想得到一个随机整数，你可以用 Math.floor 方法来实现，floor 返回小于或等于指定数字或表达式最接近的整数。下面代码生成 0～9 的随机整数

```
var random3:Number=Math.floor(Math.random()*10)
```

如果定义的范围不包含 0，可以在返回的结果上加上指定数即可。下面代码生成 1～10 的随机整数

```
var random4:Number=Math.floor(Math.random()*10)+1
```

2.6.5 洗牌阵列

在游戏中，随机数最常用的地方就是在纸牌游戏的洗牌中了。例如，你手中有 52 张纸牌，现在想随机地把它打乱，这需要怎样处理呢？

首先，你要创建一个新数组，简单地按顺序进行排列。下面的代码显示从 0 到 51 有序数字

```
// 创建有序数组
var startDeck:Array = new Array();
for(var cardNum:int=0;cardNum<52;cardNum++) {
startDeck.push(cardNum);
}
trace(“Unshuffled:”,startDeck);
```

测试影片，输出结果如下：

```
Unshuffled:
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
```

为了打乱这个数组元素的顺序，我们先选择这个数组随机位置上的一个元素，并把它存放在一个新的数组里面，然后，在原数组中把该元素删除，直至把原数组元素清空为止。

```
// shuffle into new array
var shuffledDeck:Array = new Array();
while (startDeck.length > 0) {
var r:int = Math.floor(Math.random()*startDeck.length);
shuffledDeck.push(startDeck[r]);
startDeck.splice(r,1);
}
trace(“Shuffled:”, shuffledDeck);
```

测试影片，因每次运行生成的随机数顺序不一样，只要打乱顺序即可：
Shuffled:
3,42,40,16,41,44,30,27,33,11,50,0,21,23,49,29,20,28,22,32,39,25,17,19,8,7,10,37,2,12,31,5,46,
26,48,45,43,9,4,38,15,36,51,24,14,18,35,1,6,34,13,47

2.6.7 系统数据

在游戏的设计过程中，获取玩家电脑的版本信息也很重要，从某方面来说，客户端机器的性能也会影响到游戏的一些细节。例如，你可以用 stage.stageWidth 和 stage.stageHeight 属性获取舞台的宽度和高度，如果 flash 影片设置为屏幕尺寸与浏览器窗口匹配的话，那么，屏幕的尺寸将随浏览窗口的大小而变化。
你可以用 Capabilities 对象获取 flash 影片所在电脑的多种信息。在这里列举出几种常用的属性：

Capabilities.playerType:指定播放器的类型。"StandAlone"，用于独立的 Flash Player；"External"，用于外部的 Flash Player 或处于测试模式下；"PlugIn"，用于 Flash Player 浏览器插件；"ActiveX"，用于 Microsoft Internet Explorer 使用的 Flash Player ActiveX 控件。
Capabilities.language:指定运行播放器的系统的语言代码。在英文系统上，该值返回双字母代码 (en)。

Capabilities.os：返回操作系统的类型和版本。例如:Windows XP
Capabilities.screenResolutionX, Capabilities. screenResolutionY：返回屏幕的分辨率 1280*800
Capabilities.version：flash 播放器的版本，例如 WIN 9,0,45,0。从这里你可以获取操作系统的类型和 flash 播放器的版本。

另外， 你还可以获取更多的 Capabilities 属性。在源文件 SystemData.fla 中，列出了以上功能的代码：
附：读读源文件：

```
function showInfo(event:Event)
{
    var output:String = "";
    output += "stageWidth: "+stage.stageWidth+"\n";
    output += "stageHeight: "+stage.stageHeight+"\n";
    output += "playerType: "+Capabilities.playerType+"\n";
    output += "language: "+Capabilities.language+"\n";
    output += "os: "+Capabilities.os+"\n";
    output += "screenResolutionX: "+Capabilities.screenResolutionX+"\n";
    output += "screenResolutionY: "+Capabilities.screenResolutionY+"\n";
    output += "version: "+Capabilities.version+"\n";
    output += "serverString: "+Capabilities.serverString+"\n";
    messageText.text = output;
}
stage.scaleMode = StageScaleMode.EXACT_FIT;
addEventListener(Event.ENTER_FRAME,showInfo);
```

匹配游戏制作方法

游戏玩法：
开始游戏后，有 30 张背景卡片，其中有两张是相同的。当鼠标点击任意一张卡片后，该卡片自动翻转，然后再点击下一张卡片，翻转后若两张卡片相同，分数加 100 分，并在屏幕上移除这两张卡片；若不相同，分数会减 5 分；待所有的卡片都匹配完成后，游戏结束，最后显示出所得分数和所用时间。

制作步骤（主要分成游戏元素的组合，游戏代码的实现两部分）：
第一步:
1.本书大部分游戏都是用三帧方法来实现的。第一帧为游戏规则介绍，开始游戏功能实现；第二帧游戏体实现；第三帧为游戏结束后数据显示和重新开始游戏功能的实现。
2.前期准备工作。根据本游戏规则，需要事先准备 15 张尺寸相同的卡片，3 种音效文件；在元件库中拖入两个按钮。
3.第一帧元素布置：用文本框输入游戏名称和游戏规则，并添加一按钮作为开始游戏控制，实例名称为：playButton。并在主时间轴内输入以下代码：

```
//让影片停止在第一帧
stop();
//对按钮进行侦听，点击后跳转到第二帧 playgame。
playButton.addEventListener(MouseEvent.CLICK,startGame);

function startGame(event:MouseEvent)
{
    gotoAndStop("playgame");
}
```

4.第二帧元素布置：新建 1 个影片剪辑，名称以及连接类名为 MatchingGameObject；并把 MatchingGameObject 拖放到舞台的左上角；另再建立一个影片剪辑，名称和类名为 Card，在 Card 的第一帧绘制一卡片大小区域，填充为黑色。在这里大小为 50×50；另再导入准备好的这 15 张卡片，分别存放在该影片剪辑的 2 至 16 帧；最后在属性内设置帧名为 playgame。
5.第三帧元素布置：绘制一个静态文本，添加内容为 GAME OVER，另绘制两个动态文本，实例名称分别为 showScore，showTime。再添加一实例名称为 playAgainButton 的按钮，以便点击后返回重新开始游戏。主时间轴代码为：

```
//声明游戏分数和时间的变量，并把值传递给绘制的文本
var gameScore:int;
var gameTime:String;
showScore.text = "分数: "+String(gameScore);
showTime.text = "时间: "+gameTime;
//给按钮添加侦听，点击后返回到第二帧重新开始游戏
playAgainButton.addEventListener(MouseEvent.CLICK,playAgain);
function playAgain(event:MouseEvent)
{
    gotoAndStop("playgame");
```

}

第二步：

1. MatchingGameObject 类

```
package
{
    import flash.display.MovieClip;
    import flash.events.MouseEvent;
    import flash.events.TimerEvent;
    import flash.events.Event;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.utils.getTimer;
    import flash.utils.Timer;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    /**
    * ...
    * @author ☆璀璨星晨☆
    */
    public class MatchingGameObject extends MovieClip
    {
        // 声明游戏所用的静态常量
        private static    const boardWidth:uint = 6;//卡片的列数
```

```
        private static    const boardHeight:uint =5;//卡片的行数

        private static    const cardHorizontalSpacing:Number = 52;//卡片水平间隔

        private static    const cardVerticalSpacing:Number = 52;//卡片垂直间隔

        private static    const boardOffsetX:Number = 145;//卡片在屏幕上的左上角横坐标

        private static    const boardOffsetY:Number = 90;//卡片在屏幕上左上角纵坐标

        private static    const pointsForMatch:int = 100;//应用在游戏的分数中，如果匹配分数加 100 分

        private static    const pointsForMiss:int = -5;//应用在游戏的分数中，如果不匹配减 5 分


        // 声明所用的变量

        private var firstCard:Card;//设定两个临时变量

        private var secondCard:Card;

        private var cardsLeft:uint;//用于卡片计数，为 0 时游戏结束

        private var gameScore:int;//分数

        private var gameStartTime:uint;//初始化时间

        private var gameTime:uint;//获取游戏时间


        // 声明两个文本域

        private var gameScoreField:TextField;

        private var gameTimeField:TextField;


        // 卡片自动翻转时间变量
```

```
private var flipBackTimer:Timer;


// 声明声音变量
var theFirstCardSound:FirstCardSound = new FirstCardSound();
var theMissSound:MissSound = new MissSound();
var theMatchSound:MatchSound = new MatchSound();


// 初始化函数
public function MatchingGameObject():void
{
    // 生成舞台上 15 对卡片列表的索引值，并存入到数组 cardlist 中
    var cardlist:Array = new Array();
    for (var i:uint=0; i<(boardWidth*boardHeight)/2; i++)
    {
        cardlist.push(i);
        cardlist.push(i);
    }
    // 把 30 张卡片添加到舞台上
    cardsLeft = 0;
    for (var x:uint=0; x<boardWidth; x++)
    {// 横向方块数
        for (var y:uint=0; y<boardHeight; y++)
```

```
{// 纵向方块数
var c:Card = new Card();// 生成卡片的副本
c.stop();// 让卡片影片剪辑停止在第一帧
c.x = x*cardHorizontalSpacing+boardOffsetX;// 设置卡片在屏幕上的左上角点坐标
c.y = y*cardVerticalSpacing+boardOffsetY;
   var r:uint = Math.floor(Math.random()*cardlist.length);// 得到一个随机卡片的索引值
c.cardface = cardlist[r];//把卡片索引值赋值给 c.cardface
cardlist.splice(r,1);//从数组列表中移除卡片
 c.addEventListener(MouseEvent.CLICK,clickCard);//设置一点击卡片的侦听函数
c.buttonMode = true;
addChild(c);//把卡片添加到屏幕
cardsLeft++;//生成卡片数自动加 1
      }
}
//设置文本格式
var format:TextFormat=new TextFormat();
format.size=25;
format.bold=true;
format.color=0xffff00;
// 设置游戏分数文本域，并赋值
gameScoreField = new TextField();
addChild(gameScoreField);
```

```
gameScoreField.defaultTextFormat=format;

gameScoreField.selectable=false;

gameScoreField.width=150;

gameScore = 0;

//调用计分函数

showGameScore();



//设置时钟文本域

gameTimeField = new TextField();

gameTimeField.x = 400;

addChild(gameTimeField);

gameTimeField.defaultTextFormat=format;

gameTimeField.selectable=false;

gameTimeField.width=150;

gameStartTime = getTimer();

gameTime = 0;
//调用计时函数
addEventListener(Event.ENTER_FRAME,showTime);
}
//  点击卡片函数响应
public function clickCard(event:MouseEvent)
{
var thisCard:Card = (event.target as Card);//  把点击的卡片对象赋值给他的实例

if (firstCard == null)
{
firstCard = thisCard;//第一次点击后翻转
thisCard.startFlip(thisCard.cardface+2);
```

```
playSound(theFirstCardSound);

} else if (firstCard == thisCard)
{//第二次点击后再翻转回去
firstCard.startFlip(1);
firstCard = null;
playSound(theMissSound);

} else if (secondCard == null)
{//如果再点击第二张图片，显示正面
secondCard = thisCard;//
thisCard.startFlip(thisCard.cardface+2);

//比较两个卡片
if (firstCard.cardface == secondCard.cardface)
{
// 如果相匹配，都移除屏幕
removeChild(firstCard);
removeChild(secondCard);
// 两个变量值复位
firstCard = null;
secondCard = null;
// 计算分数，匹配的话加 100 分
gameScore += pointsForMatch;
//调用显示分数文本函数
showGameScore();
playSound(theMatchSound);

// 通过 cardsLeft 检测游戏是否结束
cardsLeft -= 2;// 若匹配一次，总数量 2
if (cardsLeft == 0)
{
//卡片全部匹配完成后，跳转到第三针，显示最终分数和时间
MovieClip(root).gameScore = gameScore;
MovieClip(root).gameTime = clockTime(gameTime);
MovieClip(root).gotoAndStop("gameover");
}
} else
{
gameScore += pointsForMiss;
showGameScore();
playSound(theMissSound);
//如果两张图片不匹配，则 2 秒后自动翻转回去
flipBackTimer = new Timer(2000,1);
```

```
flipBackTimer.addEventListener(TimerEvent.TIMER_COMPLETE,returnCards);
flipBackTimer.start();
}
} else
{
returnCards(null);
playSound(theFirstCardSound);
firstCard = thisCard;
firstCard.startFlip(thisCard.cardface+2);
}
}
// 卡片自动翻转回去
public function returnCards(event:TimerEvent)
{
firstCard.startFlip(1);
secondCard.startFlip(1);
firstCard = null;
secondCard = null;
flipBackTimer.removeEventListener(TimerEvent.TIMER_COMPLETE,returnCards);
}
public function showGameScore()
{
//显示游戏分数
gameScoreField.text ="分数:"+String(gameScore);
}
public function showTime(event:Event)
{
//显示游戏时间毫秒数
gameTime = getTimer()-gameStartTime;
gameTimeField.text = "时间:"+clockTime(gameTime);
}
public function clockTime(ms:int)
{
var seconds:int = Math.floor(ms/1000);
var minutes:int = Math.floor(seconds/60);
seconds -= minutes*60;
var timeString:String = minutes+":"+String(seconds+100).substr(1,2);
return timeString;
}
public function playSound(soundObject:Object)
{
var channel:SoundChannel = soundObject.play();
}
}
```

```
}
```

1. Card

```
package
{
import flash.display.MovieClip;
import flash.events.Event;

public dynamic class Card extends MovieClip {;
private var flipStep:uint;
private var isFlipping:Boolean = false;
private var flipToFrame:uint;

// 这个函数将代替主函数中所有的 gotoAndStop 函数
public function startFlip(flipToWhichFrame:uint)
{
isFlipping = true;
flipStep =10;
flipToFrame = flipToWhichFrame;
this.addEventListener(Event.ENTER_FRAME, flip);
}
// 需要 10 侦完成卡片翻转
public function flip(event:Event)
{
flipStep--;//侦数递减

if (flipStep > 5)
{// 前 5 侦翻转时放大状态
this.scaleX = .20*(flipStep-6);
} else
{// 后 5 侦翻转时放大状态
this.scaleX = .20*(5-flipStep);
}
// 翻转中间时刻跳转到所需侦的卡片
if (flipStep == 5)
{
gotoAndStop(flipToFrame);
}
// 翻转完成后移出侦听
if (flipStep == 0)
{
this.removeEventListener(Event.ENTER_FRAME, flip);
}
}
}
```

}

第四章 头脑游戏 -记忆力和推理小游戏

GPU4.1 数组和数据对象

章，我们来制作下面两个小游戏，记忆力和推理小游戏。记忆力游戏要求玩家观察和重复一个顺序，每次，这个顺序都会延长，直到玩家不能够记住位置，第二个游戏，要求玩家猜出一个顺序，根据后面的反馈，设法猜出正确的顺序。为了完成上面的游戏，我们需要用到数组和数据对象的知识，用开存储游戏信息和决定玩家每次猜完后状态的输出结果。

4.1 数组和数据对象

4.1.1 数组

一个数组就是一个值列表。例如，在玩家开始游戏后，如果要选择游戏人物角色的列表，那么，我们就可以把这个列表存储为数组。

```
var characterTypes:Array = new Array();
characterTypes = [“Warrior”, “Rogue”, “Wizard”, “Cleric”];
```

我们还可以用 push 方法增加数组的内容，与前面代码生成相同的结果：

```
var characterTypes:Array = new Array();
characterTypes.push(“Warrior”);
characterTypes.push(“Rogue”);
characterTypes.push(“Wizard”);
characterTypes.push(“Cleric”);
```

上面这个例子，我们存储的是字符串。其实，在数组中，我们可以存储任何类型的值，比如 number 类型或 sprite 和 movie clip 显示对象等。制作游戏时，经常用数组来存储我们创建的影片剪辑。例如，在第三章中，为了访问方便，我们把每张卡片都存储到数组中就用到了该方法。如果我们想创建 10 张卡片，代码如下：

```
var cards:Array = new Array();
for(var i:uint=0;i<10;i++) {
var thisCard:Card = new Card();
cards.push(thisCard);
}
```

在数组中，常用的增加和删除元素的方法如下：

方法	举例	说明
push	myArray.push("Wizard")	在数组尾部添加一个元素
pop	myArray.pop()	将数组最后一个元素删除，并返回这个元素
unshift	myArray.unshift("Wizard")	在数组的头部添加一个元素
shift	myArray.shift("Wizard")	将数组第一个元素删除，并返回这个元素
splice	myArray.splice(7,2,"Wizard","Bard")	从数组的某位置开始，删除指定数目的元素，并插入一些新元素
indexOf	myArray.indexOf("Rogue")	返回元素的索引值，如果没有发现的话返回－1
sort	myArray.sort()	对数组进行排序，返回值为排序后的数组

4.1.2 Object 类型

数组在存储单一数值的功能是非常强大的。如果要把某些混合的值存储到一起怎么办呢？设想，在一个冒险游戏中，在某一级别中，需要有一个角色类型和生命值在一组中。在这里，

用 Object 类型就比较方便了。
首先，定义一个 Object 类型，并新建一个空的 Object 对象，然后，用点运算符来添加它的属性：

```
var theCharacter:Object = new Object();
theCharacter.charType = “Warrior”;
theCharacter.charLevel = 15;
theCharacter.charHealth = 0.8;
```

另外，你还可以用下面的方法来创建这个变量

```
var theCharacter:Object = {charType: “Warrior”, charLevel: 15, charHealth: 0.8};
```

Object 可以动态添加属性，你可以在创建 Object 类型后再添加它的属性。并且，这些属性可以是任意类型的变量，在这个 Object 中，你不需要声明这个变量，只要对它进行赋值就行了。

4.1.3 数据对象数组
从现在开始，我们将要用数据对象数组来存储数据元素。例如，一个数据对象可以这样写：

```
var thisCard:Object = new Object();
thisCard.cardobject = new Card();
thisCard.cardface = 7;
thisCard.cardrow = 4;
thisCard.cardcolumn = 2;
```

上面就是一个数据对象数组。在第三章中，我们可以把所有的卡片放在 object 对象中。
4.2 记忆游戏
游戏玩法：
开始游戏后，在屏幕上会出现 5 盏不同颜色的灯，初始时为关闭状态。此时，屏幕上有一盏灯点亮，玩家点击该灯后完成任务；之后，会依次点亮 2 盏灯，玩家按照点亮的顺序重复完成，若重复的顺序正确，会接着点亮三盏灯……直到你不能正确的找出所有顺序为止，游戏结束。最终显示出分数和时间。

制作步骤：
第一步：前期准备工作
新建一个影片剪辑，命名为 LightColors，根据灯的形状大小在影片内的帧上分别绘制 5 种不同的颜色。在这里颜色分别为 ff0000,cc6600,ff00ff,00ff00,0000ff。另再新建一个影片剪辑，名称和链接名都为 Light，然后在影片内建立两帧的动画，第一帧名称为 off，第二帧名称为 on，图层改名为 Lables；新建 Script 图层，在时间轴内输入代码 stop(); 新建 Border 层，绘制一灯罩图形；新建 Shade 图层，在灯罩内绘制一灰色的图形，透明度为 75%，再新建 Light 图层，把刚才绘制的 LightColors 影片剪辑拖入到灯罩内。在这里，要注意存放图形的三个图层的位置关系，由上到下依次是 Border，Shade 和 Light。

第二步：编程设置策略
我们通过代码来创建所有的可视元素。首先，先创建 5 盏灯并为他们各设置一种不同的颜色。然后，再创建 2 个文本区域，一个用来提示玩家注意观察灯光显示的顺序，另一个用来提示在当前情况下有多少盏灯点亮。Lights 影片剪辑内的 5 种灯光的颜色，将被存储在一个数组内，以便我们比较容易地控制这些灯光的亮和灭。另外，还需要把灯光显示的顺序存储在一个空数组 A 内，每次添加一个元素。游戏运行时，我们声明一个数组 B，并把数

组 A 内的元素赋值给数组 B。然后，由玩家按照记忆中的顺序每点击一盏灯后，我们就把数组 B 的第一个元素移除，如果这个点击的顺序与原来产生的顺序相同，玩家就可以进入下一次循环；否则，游戏结束。

第三步：文档类 MemoryGame 定义

```
package
{
  import flash.display.*;//显示影片
  import flash.events.*;//鼠标事件调用
  import flash.text.*;//显示文本
  import flash.utils.Timer;//计时器调用
  import flash.media.Sound;//
  import flash.media.SoundChannel;//声音播放调用
  import flash.net.URLRequest;//加载外部声音文件调用

  public class MemoryGame extends MovieClip
  {
    static const numLights:uint = 5;
    private var lights:Array;// 用来存储影片剪辑的 5 种不同颜色的灯光
    private var playOrder:Array;//把每次递增后的元素顺序存储起来
    private var repeatOrder:Array;//repeatOrder 作为 playOrder 数组的备份，当玩家每点击一次，
repeatOrder 数组就删除一个元素，直到清空为止
    // 声明两个文本，用来存储提示玩家的数据
    private var textMessage:TextField;
    private var textScore:TextField;
    // 声明两个计时器，用来存储灯光开和关的间隔时间
    private var lightTimer:Timer;
    private var offTimer:Timer;
    var gameMode:String;// 游戏模式：play or replay
    var currentSelection:MovieClip = null;//保存当前选择的影片
    var soundList:Array = new Array();//存储声音
    public function MemoryGame()
    {
      // 设置文本格式
      var textFormat = new TextFormat();
      textFormat.font = "Arial";
      textFormat.size = 24;
      textFormat.align = "center";
      textFormat.bold=true;
      // 创建上面的文本
      textMessage = new TextField();
      textMessage.width = 550;
      textMessage.y = 110;
```

```
      textMessage.selectable = false;
      textMessage.defaultTextFormat = textFormat;
      addChild(textMessage);
      // 创建下面的文本
      textScore = new TextField();
      textScore.width = 550;
      textScore.y = 250;
      textMessage.selectable = false;
      textScore.defaultTextFormat = textFormat;
      addChild(textScore);
      // 加载声音，注意外部文件名称
      soundList = new Array();
      for (var i:int=0; i<5; i++)
      {
        var thisSound:Sound = new Sound();
        var req:URLRequest = new
URLRequest("http://xiaocui.blogbus.com/files/1233588133"+i+".mp3");
        thisSound.load(req);
        soundList.push(thisSound);
      }
      // 添加灯泡影片剪辑
      lights = new Array();
      for (i=0; i<numLights; i++)
      {
        var thisLight:Light = new Light();
        thisLight.lightColors.gotoAndStop(i+1);// 顺序选择适当的灯光颜色帧
        thisLight.x = i*75+100;// 分别放置在舞台上的对应位置
        thisLight.y = 175;
        thisLight.lightNum = i;// 记录灯光的编号
        lights.push(thisLight);//将灯泡添加到数组中
        addChild(thisLight);// 添加到舞台
        thisLight.addEventListener(MouseEvent.CLICK,clickLight);// listen for clicks
        thisLight.buttonMode = true;
      }
      // 重置声音序列，开始游戏
      playOrder = new Array();
      gameMode = "play";
      nextTurn();
    }
    public function nextTurn()
    {
      // //在序列中随机添加一个新的灯泡，并存储到数组内
      var r:uint = Math.floor(Math.random()*numLights);
      playOrder.push(r);
```

```
    // 显示文本赋值
    textMessage.text = "仔细观察和聆听.";
    textScore.text = "序列长度: "+playOrder.length;
    // 设置显示序列计时器
    lightTimer = new Timer(1000,playOrder.length+1);
    lightTimer.addEventListener(TimerEvent.TIMER,lightSequence);
    //开始计时
    lightTimer.start();
  }
  //播放下一个声音
  public function lightSequence(event:TimerEvent)
  {
    // 获取播放队列的位置
    var playStep:uint = event.currentTarget.currentCount-1;
    if (playStep < playOrder.length)
    {// 不是最后一次，点亮灯泡
      lightOn(playOrder[playStep]);
    } else
    {// 序列结束
      startPlayerRepeat();
    }
  }
  // 开始重复顺序
  public function startPlayerRepeat()
  {
    currentSelection = null;
    textMessage.text = "重复该顺序.";
    gameMode = "replay";
    repeatOrder = playOrder.concat();
    /*当开始重复这个序列时，我们需要把文本设置为 Repeat，gameMode 设置为 replay。
然后再把 playOrder 数组复制给 repeatOrder。注意，在这里我们用 concat 而不用 repeatOrder
= playOrder，因为用 concat 当改变第二个数组时不会改变 playOrder 数组的内容。*/
  }
  // 点灯，并设置定时熄灭
  public function lightOn(newLight)
  {
    soundList[newLight].play();// 播放声音
    currentSelection = lights[newLight];
    currentSelection.gotoAndStop(2);//开灯
    offTimer = new Timer(500,1);// 定时关掉
    offTimer.addEventListener(TimerEvent.TIMER_COMPLETE,lightOff);
    offTimer.start();
  }
  // 定时关灯
```

```
    public function lightOff(event:TimerEvent)
    {
      if (currentSelection != null)
      {
        currentSelection.gotoAndStop(1);
        currentSelection = null;
        offTimer.stop();
      }
    }
    // 接收鼠标点击事件
    public function clickLight(event:MouseEvent)
    {
      // 当影片在播放状态下，防止鼠标点击
      if (gameMode != "replay")
      {
        return;
      }
      //如果灯还没有关闭，先关闭它
      lightOff(null);
      // 比较
      if (event.currentTarget.lightNum == repeatOrder.shift())
      {
        lightOn(event.currentTarget.lightNum);
        // 检测序列是否结束
        if (repeatOrder.length == 0)
        {
          nextTurn();
        }
        // 出错后，结束游戏
      } else
      {
        textMessage.text="Game Over!";
        gameMode = "gameover";
      }
    }
  }
}
```