

FORMAÇÃO CONTÍNUA FBAUP

INTRODUÇÃO AO ADOBE FLASH 8 - TÉCNICAS DE PLANIFICAÇÃO, CONCRETIZAÇÃO E PUBLICAÇÃO DE PROJECTOS MULTIMÉDIA.



FBAUP, Pedro Amado, 2007-02-12
Versão 0.2 (alfa). Actualizada em 2007-02-26.

Este trabalho está licenciado sob uma Licença Creative Commons Atribuição-Usos Não-Comerciais-Partilha nos termos da mesma Licença 2.5 Portugal. Para ver uma cópia desta licença, visite <http://creativecommons.org/licenses/by-nc-sa/2.5/pt/> ou envie uma carta para Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Qualquer correcção ou sugestão:
[pamado\(at\)fba.up.pt](mailto:pamado(at)fba.up.pt)

Para mais informações:
<http://users.fba.up.pt/~pamado/designlab>

RESUMO E OBJECTIVOS DO CURSO:

No âmbito de uma formação Multimédia completa, esta formação tem por objectivo preparar os formandos na utilização do Adobe Flash 8 para a **planificação, criação e finalização de projectos** de estáticos ou dinâmicos para animação, criação de *websites*, CD-ROMs ou ainda para veicular e estruturar conteúdos de forma eficaz.

SUMÁRIO

Resumo e Objectivos do curso:.....	1
Sumário	2
Índice de Quadros e Figuras	6
Apresentação	8
Introdução	9
What is Flash?.....	9
Programa.....	10
Introdução ao Adobe Flash 8 (~6hrs);	10
ActionScript 2.0 fundamental (~6hrs)	10
1 Flash Basic Training (~6hrs)	13
1.1 Panorama geral.....	13
Tipos de Ficheiros	13
Bitmaps vs. vectors	14
Integração do Illustrator com o Flash (A desenvolver)	16
Flash 8 vs. Flash Professional 8	16
Como Aprender?.....	17
1.2 Interface (IDE).....	17
Página de abertura (Start Page)	17
Workspaces	18
Preferências.....	19
Cena ou Stage.....	20
Timeline	21
Frames vs. Keyframes.....	22
Layers.....	22
Exercício:	23
Guide Layer	23
1.3 Desenho e Cor (Ferramentas Básicas)	24
Seleção	24
Edição	24
Transformação.....	24

1.4	Paineis	25
	Property Inspector	25
	Library panel	25
	Actions panel	26
	Cenas	27
1.5	Animação Básica.....	27
	Property Inspector	27
	Timeline, Layers, frames, and keyframes	27
	Exercício:	28
1.6	Shape Tweens	28
	Shape tweening.....	29
	Shape hinting.....	29
1.7	Libraries, Symbols, and Instances	30
	Exercício:	31
1.8	Motion Tweens.....	32
	Exercício:	32
1.9	Timeline Effects	32
	Exercício:	32
1.10	Masking.....	32
	Exercício:	33
1.11	Tipografia.....	33
	Exercício:	36
1.12	Bitmaps	36
	Exercício:	37
1.13	Filters and Blend Modes (Professional Only)	37
1.14	Buttons.....	38
	Exercício:	39
1.15	Movie Clips.....	39
	Animated rollover buttons	40
1.16	ActionScript Essentials	40
	Script Assist	40
	Stop on frame	41
	GetURL	42
	Slide show.....	42
	Exercício:	42
	Creating a pop-up Menu (Buttons + Frame Labels)	43
	Exercício:	43
	Scrolling text	43
	Exercício:	44

Controlling Movie Clips (Targets)	44
Exercício:	44
Loading SWFs and images	45
Building a simple preloader 1	45
Exercício:	45
Building a simple preloader 2	47
Exercício:	48
Behaviors	49
Exercício:	49
1.17 Components	49
What are Components?	49
Exercício:	50
1.18 Som	50
Importar sons	50
Formatos suportados	51
Som na TimeLine e em Botões (Behaviour, Goto e <i>ActionScript</i>)	51
Iniciar e parar sons	52
Exercício:	52
1.19 Video	52
Importar e Comprimir video	54
Embedding video in a SWF file	55
1.20 Publicar Conteúdos	57
Publicar – <i>Preferences</i> e formatos	58
Criar um projector	58
FS command	58
Testing Movies	60
The Bandwidth Profiler and simulating a download	60
Generating a size report	63
The Flash Player Detection Kit	63
2 ActionScript 2.0 Essencial (~6hrs)	65
2.1 Introdução ao ActionScript 2.0	65
O que é o ActionScript?	65
ActionScript e eventos	65
Painel de Acções (Actions panel)	68
Syntaxe e Linguagem Fundamental	69
dot syntax e target paths	70
Visibilidade das Instâncias (scope)	70
2.2 Adding ActionScript	72

Tipos de Dados e Variáveis.....	72
Tipos de Dados Simples (Primitivos) e Complexos	72
Variáveis	74
Visibilidade das Variáveis (Scope)	75
Expressões, Declarações e operadores (operacionais, estruturais e terminais)	76
Operadores.....	78
Condições.....	80
Estrutura If.....	80
Exercício If.....	81
Estrutura Switch.....	82
Exercício Switch.....	83
Iterações (Estrutura For).....	83
Funções e Métodos	85
About built-in and top-level functions.....	87
Writing named functions	88
Using variables in functions.....	88
Passing parameters to a function.....	89
Returning values from functions	90
2.3 Introdução aos Objectos e Vectores (Arrays)	91
Vectores (arrays).....	91
Exercício Array	92
2.4 Movie Clips e Botões.....	93
Movie Clips	93
Controlar movie clips com ActionScript	93
Exercício	95
Desafio.....	95
Loading and unloading SWF files	95
Exercício	96
Desafio.....	96
2.5 Working with Text	96
2.6 Creating Basic User Input Forms.....	96
2.7 Controlling Sound.....	97
Creating and attaching sound	97
Loading sound.....	97
Start, stop and pause	97
Bibliografia e links úteis	101

ÍNDICE DE QUADROS E FIGURAS

Figure 1 - Pedro Amado (fotografia)	8
Figure 2 - Ficheiros Nativos do Flash	13
Figure 3 - Tipos de ficheiros mais comuns gerados pelo Flash	13
Figure 4 - Imagem Bitmap	16
Figure 5 – Imagem Vectorial	16
Figure 6 - Flash Help (F1)	17
Figure 7 - Start Page	18
Figure 8 - Workspaces	19
Figure 9 - Preferências (MacOS)	19
Figure 10 - Preferências Comuns	19
Figure 11 – Cena (Stage)	21
Figure 12 – Timeline	22
Figure 13 - ferramentas de Selecção	24
Figure 14 - Ferramentas de Edição (Tooblox)	24
Figure 15 – Ferramentas de transformação (Toolbox)	24
Figure 16 - Properties Inspector	25
Figure 17 - Library panel	25
Figure 18 - Actions panel	26
Figure 19 - Scene Panel	27
Figure 20 - Frames e KeyFrames	27
Figure 21 - Keyframes em Animação e Onion Skin ning	28
Figure 22 - Tween Básico (Shape)	29
Figure 23 - Shape Hints	29
Figure 24 - Shape Hints	30
Figure 25 - Motion Tweens	32
Figure 26 - Adicionar uma Máscara	33
Figure 27 - Properties Inspector (Tipografia)	36
Figure 28 - Import Options	37
Figure 29 - Inserting a Button	38
Figure 30 - Button Timeline	39
Figure 31 - Symbols Hierarchy	39
Figure 32 - Botão Animado	40
Figure 33 - Time based ActionScript	40
Figure 34 - User based ActionScript	40
Figure 35 - Script Assist (Actions panel)	41
Figure 36 - Stop on Frame (Script Assist)	41

Figure 37 - Get URL (Script Assist)	42
Figure 38 - Creating a Slideshow.....	42
Figure 39 - Popup Menu (Flash).....	43
Figure 40 - Scrolling text	44
Figure 41 - Preloader de 2 Frames	45
Figure 42 - behaviours Panel	49
Figure 43 - UI ScrollBar Component.....	50
Figure 44 - Sons em Layers.....	51
Figure 45 - StopAllSounds();.....	52
Figure 46 - Publish Preferences Window.....	58
Figure 47- Bandwidth Profiler.....	62
Figure 48 - Size Report	63
Figure 49 - Mouse Events.....	66
Figure 50 – ClipEvents	67
Figure 51 – Eventos Automáticos	67
Figure 52 - Paine de Acções	68
Figure 53 - Scope das Instancias.....	71
Figure 54 - Variaveis e trace()	74
Figure 55 - Declarar, Definir, Nomear e Inicializar variáveis	75
Figure 56 - Expressão Simples	77
Figure 57 - Exercício IF	81
Figure 58 - Loop For Falso (Timeline)	85
Figure 59 - Exercício Array.....	92
Figure 60 - Targeting MovieClips	95

APRESENTAÇÃO



FIGURE 1 - PEDRO AMADO (FOTOGRAFIA)

Pedro Amado

Técnico Superior de Design, FBAUP

Licenciado em Design de Comunicação, FBAUP

Aluno de Mestrado Multimédia, FBAUP (Finalizar a dissertação)

Estatuto de Director e Orientador no Movimento de Tempos Livres MOCAMFE

Realiza pequenas experiências em desenho digital, programação e construção *websites* desde 2001. Desenvolve Design Tipográfico nos tempos livres – www.typeforge.net

Actualmente a implementar o DesignLab, página de apoio ao Gabinete do Técnico de Design, FBAUP: <http://users.fba.up.pt/~pamado/designlab>

Quaisquer dúvidas que surjam ou eventuais materiais de apoio cuja necessidade se verifique dirijam-se a:

Pedro Amado

pamado@fba.up.pt

<http://users.fba.up.pt/~pamado/designlab>

INTRODUÇÃO

WHAT IS FLASH?

De notar que na definição da Adobe, o Flash não é indicado para a construção de websites *per se*, mas sim para a construção de aplicação de *rich media* que estas sim devem ser incluídas nos websites.

Flash is an authoring tool that designers and developers use to create presentations, applications, and other content that enables user interaction. Flash projects can include simple animations, video content, complex presentations, applications, and everything in between. In general, individual pieces of content made with Flash are called applications, even though they might only be a basic animation. You can make media-rich Flash applications by including pictures, sound, video, and special effects.

Flash is extremely well suited to creating content for delivery over the Internet because its files are very small. Flash achieves this through its extensive use of vector graphics. Vector graphics require significantly less memory and storage space than bitmap graphics because they are represented by mathematical formulas instead of large data sets. Bitmap graphics are larger because each individual pixel in the image requires a separate piece of data to represent it.

To build an application in Flash, you create graphics with the Flash drawing tools and import additional media elements into your Flash document. Next, you define how and when you want to use each of those elements to create the application you have in mind.

When you author content in Flash, you work in a Flash document file. Flash documents have the file extension .fla (FLA). A Flash document has four main parts:

The Stage is where your graphics, videos, buttons, and so on appear during playback. The Stage is described further in Flash Basics.

The Timeline is where you tell Flash when you want the graphics and other elements of your project to appear. You also use the Timeline to specify the layering order of graphics on the Stage. Graphics in higher layers appear on top of graphics in lower layers.

The Library panel is where Flash displays a list of the media elements in your Flash document.

ActionScript code allows you to add interactivity to the media elements in your document. For example, you can add code that causes a button to display a new image when the user clicks it.

You can also use ActionScript to add logic to your applications. Logic enables your application to behave in different ways depending on the user's actions or other conditions. Flash includes two versions of ActionScript, each suited to an author's specific needs. For more information about writing ActionScript, see Learning ActionScript 2.0 in Flash in the Help panel.

Flash includes many features that make it powerful but easy to use, such as prebuilt drag-and-drop user interface components, built-in behaviors that let you easily add ActionScript to your document, and special effects that you can add to media objects.

PROGRAMA

INTRODUÇÃO AO ADOBE FLASH 8 (~6HRS);

1. Edição e animação básica;
 - a) Interface (IDE), Filmes e Cenas;
 - b) Editar e desenhar;
 - c) Animação simples (*Shape e Motion Tween*);
 - d) Biblioteca, símbolos e instancias;
 - e) *Layers* e Máscaras;
2. Criação de interacção simples com o utilizador;
 - a) Botões;
 - b) *Movie Clips*;
3. Introdução ao *ActionScript*; (*Script Assist*);
4. Som e Vídeo;
5. Publicação de projectos;

ACTIONSCRIPT 2.0 FUNDAMENTAL (~6HRS)

6. Adicionar *Scripts* a objectos;
7. Introdução aos objectos e vectores;
8. *Movie Clips* Dinâmicos;
9. Manipulação de Texto;
10. *User Input*;
11. Controlo de Som;
12. Interacção entre filmes;

1 FLASH BASIC TRAINING (~6HRS)

1.1 PANORAMA GERAL

TIPOS DE FICHEIROS



FIGURE 2 - FICHEIROS NATIVOS DO FLASH



FIGURE 3 - TIPOS DE FICHEIROS MAIS COMUNS GERADOS PELO FLASH

Flash can be used to work with a variety of file types. Each type has a separate purpose. The following list describes each file type and its uses:

***FLA files** are the primary files you work with in Flash. These are the files that contain the basic media, timeline, and script information for a Flash document.*

***SWF files** are the compressed versions of FLA files. These files are the ones you display in a web page;*

***AS files** are ActionScript files. You can use these files if you prefer to keep some or all of your ActionScript code outside of your FLA files. These can be helpful for code organization and for projects that have multiple people working on different parts of the Flash content;*

***SWC files** contain the reusable Flash components. Each SWC file contains a compiled movie clip, ActionScript code, and any other assets that the component requires.*

***ASC files** are files used to store ActionScript that will be executed on a computer running Flash Communication Server. These files provide the ability to implement server-side logic that works in conjunction with ActionScript in a SWF file.*

JSFL files are JavaScript files that you can use to add new functionality to the Flash authoring tool. See Extending Flash for more information;

FLP files are Flash Project files (Flash Professional only). You can use Flash Projects to manage multiple document files in a single project. Flash Projects allow you to group multiple, related files together to create complex applications.

BITMAPS VS. VECTORS

*There are two main categories of graphic images: **bitmap and vector**. (ADOBE, b)*

Vector graphics vs. Raster graphics: pythagoras vs. Seurat. Vector graphics - remember algebra? Remember the day you started talking about graphs in Algebra? All those x and y coordinates making lines and shapes? Remember thinking that you will never use this stuff in the real world? Well you were right. Thanks to the good people who have written vector based drawing programs like Illustrator, Freehand, and CorelDraw. They have done all the dirty work for us. Now we just open up their programs, draw some circles and squares and drag some handles around until we get what we want.

It's not really that easy, but the point is that the shapes and lines are created using Algebraic equations. The images created in these programs are called vector-based graphics or just vector graphics. Vector graphics have many strengths and weaknesses. Their strengths include their ability to render type and large areas of color with relatively small file sizes. They can also be reduced or enlarged to any size without losing any image quality. One major weakness of vector graphics is their inability to show continuous tone images like photographs or complex blends.

Raster graphics - pixels, pixels, pixels

Enter raster graphics, which are also known as pixel-based graphics or bit-mapped graphics. Raster graphics are built on a grid of pixels. The number of pixels in one inch determines an image's resolution. For example: Image A is 3 inches tall by 5 inches wide. Image B is also 3 inches tall by 5 inches wide. Image A is saved at 72 dpi (dots per inch). Image B is saved at 300 dpi. As you move in closer to view the detail in Images A & B, You will notice Image A will start to look jagged and clunky. While at the same viewing distance, Image B will still appear smooth, with clean, fine lines. Image B has a higher dpi and therefore better resolution. For a

visual demonstration of this principle go to our section on Web Graphics vs. Print Graphics - Image Resolution.

Raster Graphics are best used for large photographic images. We use programs like PhotoShop to create and manipulate raster graphics. Their strengths include their ability to show continuous tones and shading in photographic images. Their weaknesses are in their poor scalability and relatively large file sizes.

Bringing it all together

As designers, we use vector and raster graphics every day. Programs like Quark Express and PageMaker let us combine the two types of graphics to make datasheets, letterheads, tradeshow graphics, and other printed pieces. In the web world, the only two widely supported graphic formats are GIF and JPEG, which are both raster graphic formats. New technologies like the Flash Plug-in are beginning to utilize the smaller file sizes of vector graphics, but the change is coming slowly.

***Bitmap images**, also referred to as raster images, are pixel-based. This means that location and color information about the image is stored in individual pixels within a grid. Each pixel has an assigned color; some pixels are white, while other pixels are blue. The information stored in a bitmap image regarding pixel location and color is what forms the image. Bitmap images are edited at the pixel level; in other words, the color of any one pixel can be changed. Additional attributes of bitmap images include:*

- 1. Bitmap images are usually created and edited in "photo" or "paint" programs such as Adobe Photoshop.*
- 2. Bitmap images are mapped to a grid.*
- 3. The size of the image is based on the image's resolution.*
- 4. Bitmap images are not easily scalable.*
- 5. Bitmap images are used for photorealistic images and, therefore, may involve complex color variations.*

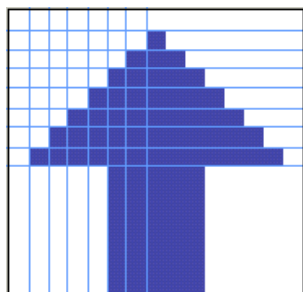


FIGURE 4 - IMAGEM BITMAP

Vector images are mathematically-based. All lines, shapes, etc. (also called objects) of a vector-based image are independent of one another. Figure 5.1-3 shows an image of a rose (actual size). Figure 5.1-4 shows one leaf of the same rose magnified, along with the paths and handles used within vector images. Additional attributes of vector-based images include:

1. *Vector-based images are usually created and edited in "draw" or "illustrate" programs such as Adobe Illustrator.*
2. *Vector-based images have smooth edges.*
3. *Vector-based images create curves or shapes.*
4. *Vector-based images are good for precise illustrations, but are not as good for photorealistic images.*
5. *Vector-based images are easily scalable, due to their use of mathematic formulas.*



FIGURE 5 – IMAGEM VECTORIAL

INTEGRAÇÃO DO ILLUSTRATOR COM O FLASH (A DESENVOLVER)

FLASH 8 VS. FLASH PROFESSIONAL 8

À primeira vista os dois programas apresentam-se muito semelhantes. No entanto as diferenças entre os dois são bastante dramáticas. A maior parte dos *upgrades* do programa residem na versão Professional: filtros, modos de dissolvência (*blend modes*), modos de

autoria *mobile* e algumas características avançadas. A versão básica está preparada essencialmente para a animação.

COMO APRENDER?

Recursos On-line:

<http://www.adobe.com/support/documentation/en/flash/>;

http://download.macromedia.com/pub/documentation/en/flash/fl8/flash_pdfs.zip;

Help do Flash (Tutoriais):

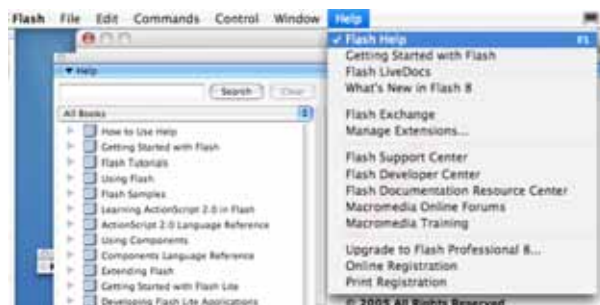


FIGURE 6 - FLASH HELP (F1)

1.2 INTERFACE (IDE)

PÁGINA DE ABERTURA (START PAGE)

“*Quickstart*” para os projectos mais comuns.



FIGURE 7 - START PAGE

Whenever Flash is running with no documents open, the Start page appears. The Start page provides easy access to frequently used actions.

The Start page contains the following four areas:

***Open a Recent Item** lets you open your most recent documents. You can also display the Open File dialog box by clicking the Open icon.*

***Create New** lists Flash file types, such as Flash documents and ActionScript files. You can quickly create a new file by clicking the desired file type in the list.*

***Create from Template** lists the templates most commonly used to create new Flash documents. You can create a new file by clicking the desired template in the list.*

***Extend** links to the Macromedia Flash Exchange website, where you can download helper applications for Flash, Flash extensions, and related information.*

The Start page also offers quick access to Help resources. You can take a tour of Flash, learn about Flash documentation resources, and find Macromedia Authorized Training facilities.

WORKSPACES

As paletas e janelas são reorganizáveis e a sua disposição pode ser guardada:

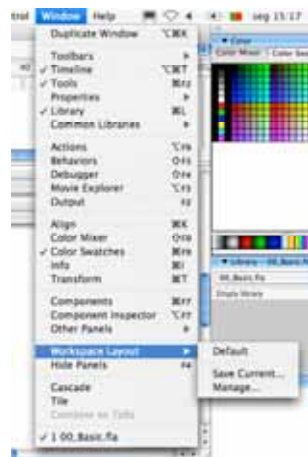


FIGURE 8 - WORKSPACES

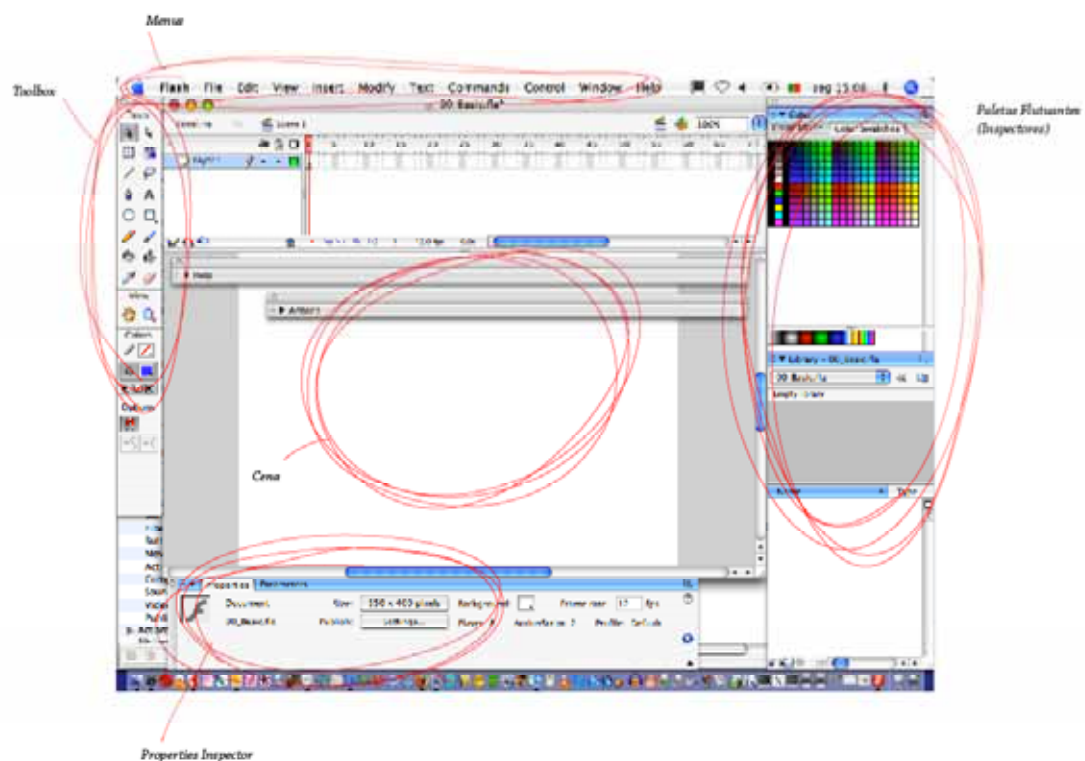
PREFERÊNCIAS



FIGURE 9 - PREFERÊNCIAS (MACOS)



FIGURE 10 - PREFERÊNCIAS COMUNS



CENA OU STAGE

The Stage is the rectangular area where you place graphic content, including vector art, text boxes, buttons, imported bitmap graphics or video clips, and so on when creating Flash documents. The Stage in the Flash authoring environment represents the rectangular space in Macromedia Flash Player or in a web browser window where your Flash document appears during playback. You can zoom in and out to change the view of the Stage as you work. The grid, guides, and rulers help you position content precisely on the Stage.

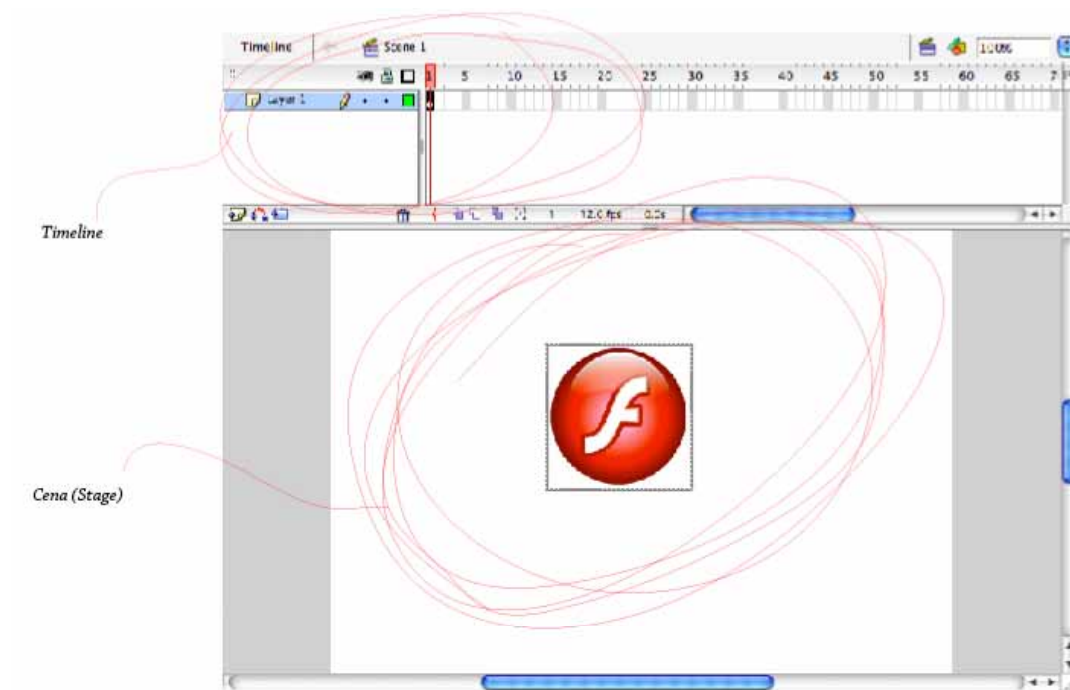


FIGURE 11 – CENA (STAGE)

TIMELINE

The Timeline organizes and controls a document's content over time in layers and frames. Like films, Flash documents divide lengths of time into frames. Layers are like multiple film strips stacked on top of one another, each containing a different image that appears on the Stage. The major components of the Timeline are layers, frames, and the playhead.

Layers in a document are listed in a column on the left side of the Timeline. Frames contained in each layer appear in a row to the right of the layer name. The Timeline header at the top of the Timeline indicates frame numbers. The playhead indicates the current frame displayed on the Stage. As a Flash document plays, the playhead moves from left to right through the Timeline.

The Timeline status display at the bottom of the Timeline indicates the selected frame number, the current frame rate, and the elapsed time to the current frame.

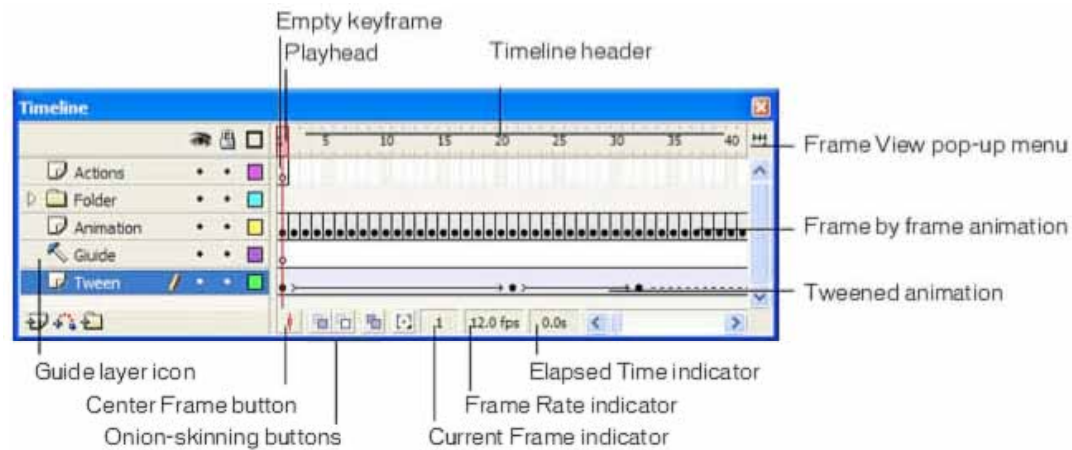


FIGURE 12 – TIMELINE

FRAMES VS. KEYFRAMES

Nomenclatura da animação tradicional. Frame, Keyframe, Tween.

A keyframe is a frame in which you define a change to an object's properties for an animation or include ActionScript code to control some aspect of your document. Flash can tween, or automatically fill in, the frames between keyframes you define in order to produce fluid animations. Because keyframes let you produce animation without drawing each individual frame, they make creating animation easier. You can easily change the length of a tweened animation by dragging a keyframe in the Timeline.

The order in which frames and keyframes appear in the Timeline determines the order in which they are displayed in a Flash application. You can arrange keyframes in the Timeline to edit the sequence of events in an animation.

LAYERS

Layers are like transparent sheets of acetate stacked on top of each other on the Stage. Layers help you organize the artwork in your document. You can draw and edit objects on one layer without affecting objects on another layer. Where there is nothing on a layer, you can see through it to the layers below.

To draw, paint, or otherwise modify a layer or folder, you select the layer in the Timeline to make it active. A pencil icon next to a layer or folder name in the Timeline indicates that the layer or folder is active. Only one layer can be active at a time (although more than one layer can be selected at a time).

When you create a new Flash document, it contains only one layer. You can add more layers to organize the artwork, animation, and other elements in your document. The number of layers you can create is limited only by your computer's memory, and layers do not increase the file size of your published SWF file. Only the objects you place into layers add to the file size. You can also hide, lock, or rearrange layers.

You can also organize and manage layers by creating layer folders and placing layers in them. You can expand or collapse layer folders in the Timeline without affecting what you see on the Stage. It's a good idea to use separate layers or folders for sound files, ActionScript, frame labels, and frame comments. This helps you find these items quickly when you need to edit them.

In addition, you can use special guide layers to make drawing and editing easier, and mask layers to help you create sophisticated effects.

EXERCÍCIO:

Criar, Ver, Editar e Organizar Layers.

GUIDE LAYER

Não é a mesma coisa que régua ou guias que se podem usar com a Cena.

For help in aligning objects when drawing, you can create guide layers. You can then align objects on other layers to the objects you create on the guide layers. Guide layers are not exported and do not appear in a published SWF file. You can make any layer a guide layer. Guide layers are indicated by a guide icon to the left of the layer name.

You can also create a motion guide layer to control the movement of objects in a motion tweened animation. For more information, see Tweening motion along a path in Using Flash.

1.3 DESENHO E COR (FERRAMENTAS BÁSICAS)

SELEÇÃO

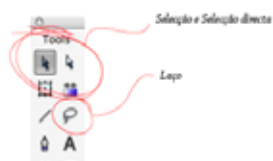


FIGURE 13 - FERRAMENTAS DE SELECÇÃO

EDIÇÃO



FIGURE 14 - FERRAMENTAS DE EDIÇÃO (TOOBLOX)

TRANSFORMAÇÃO

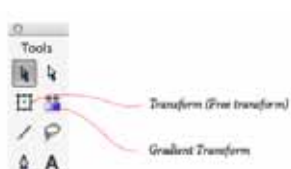


FIGURE 15 – FERRAMENTAS DE TRANSFORMAÇÃO (TOOLBOX)

1.4 PAINEIS

PROPERTY INSPECTOR

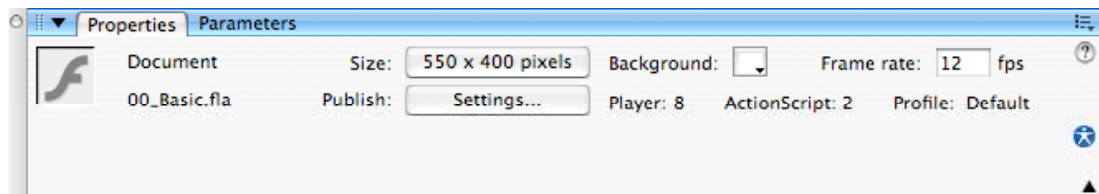


FIGURE 16 - PROPERTIES INSPECTOR

The Property inspector simplifies document creation by making it easy to access the most commonly used attributes of the current selection, either on the Stage or in the Timeline. You can make changes to the object or document attributes in the Property inspector without accessing the menus or panels that also control these attributes.

Depending on what is currently selected, the Property inspector displays information and settings for the current document, text, symbol, shape, bitmap, video, group, frame, or tool. When two or more different types of objects are selected, the Property inspector displays the total number of objects selected.

LIBRARY PANEL

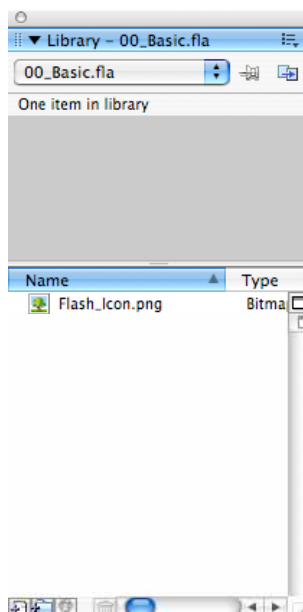


FIGURE 17 - LIBRARY PANEL

The Library panel is where you store and organize symbols created in Flash, as well as imported files, including bitmap graphics, sound files, and video clips. The Library panel lets you organize library items in folders, see how often an item is used in a document, and sort items by type. For more information, see Managing media assets with the library in Using Flash.

ACTIONS PANEL

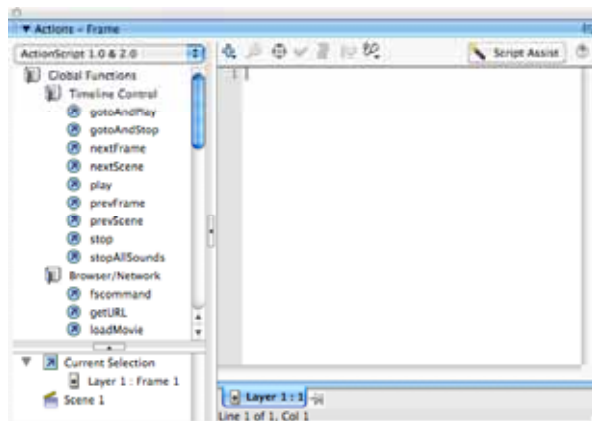


FIGURE 18 - ACTIONS PANEL

The Actions panel lets you create and edit ActionScript code for an object or frame. Selecting a frame, button, or movie clip instance makes the Actions panel active. The Actions panel title changes to Button Actions, Movie Clip Actions, or Frame Actions, depending on what is selected.

For information on using the Actions panel and writing ActionScript code, including switching between editing modes, see Using the Actions panel and Script window in Learning ActionScript 2.0 in Flash.

CENAS

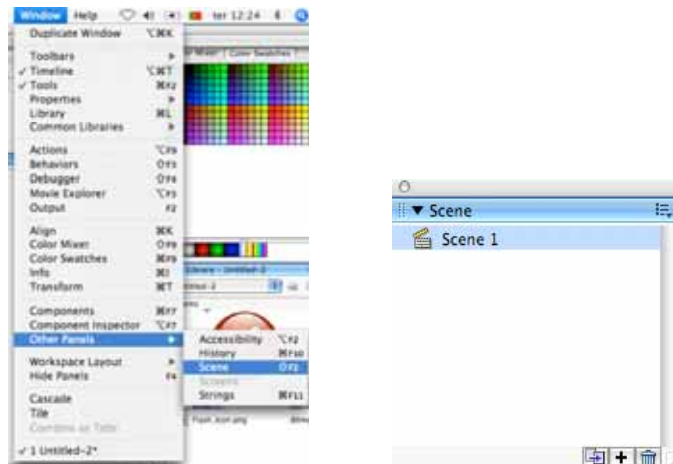
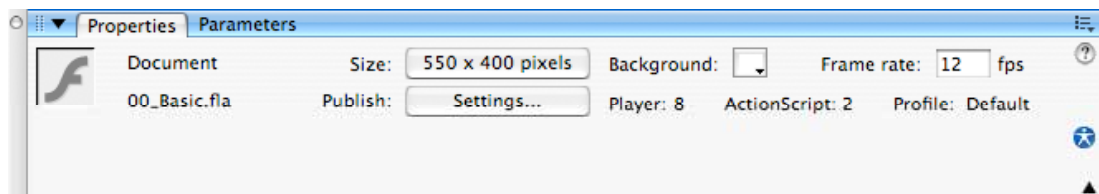


FIGURE 19 - SCENE PANEL

1.5 ANIMAÇÃO BÁSICA

PROPERTY INSPECTOR



TIMELINE, LAYERS, FRAMES, AND KEYFRAMES

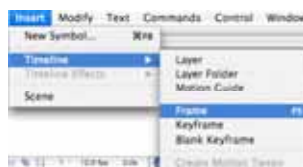


FIGURE 20 - FRAMES E KEYFRAMES



FIGURE 21 - KEYFRAMES EM ANIMAÇÃO E ONION SKIN NING

EXERCÍCIO:

Apagar, copiar e reverter *frames*

Testar filmes

Animação *Frame a Frame* vs. *Tweens* (ver capítulos que se seguem)

1.6 SHAPE TWEENS

Flash can create two types of tweened animation: motion tweening and shape tweening.

In motion tweening, you define properties such as position, size, and rotation for an instance, group, or text block at one point in time, and then you change those properties at another point in time. You can also apply a motion tween along a path. See Tweening instances, groups, and type and Tweening motion along a path.

In shape tweening, you draw a shape at one point in time, and then you change that shape or draw another shape at another point in time. Flash interpolates the values or shapes for the frames in between, creating the animation.

To apply shape tweening to groups, instances, or bitmap images, you must first break these elements apart. See Breaking apart groups and objects. To apply shape tweening to text, you must break the text apart twice to convert the text to objects.

Tweened animation is an effective way to create movement and changes over time while minimizing file size. In tweened animation, Flash stores only the values for the changes between frames.

To quickly prepare elements in a document for tweened animation, distribute objects to layers..

SHAPE TWEENING

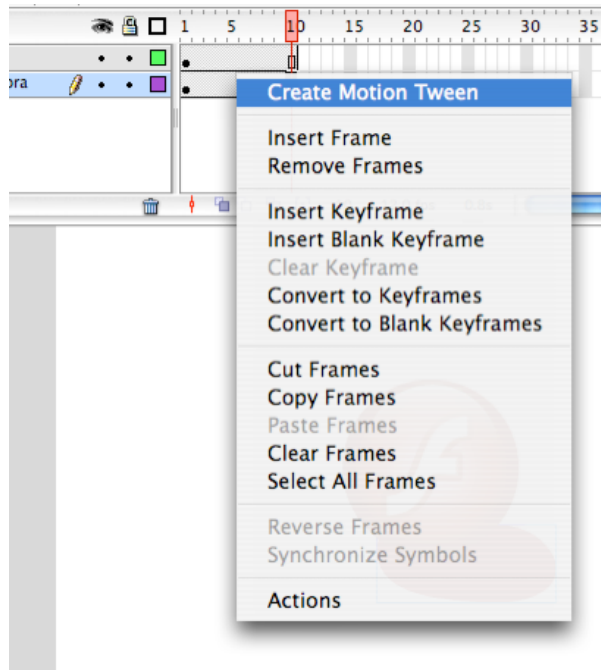


FIGURE 22 - TWEEN BÁSICO (SHAPE)

SHAPE HINTING

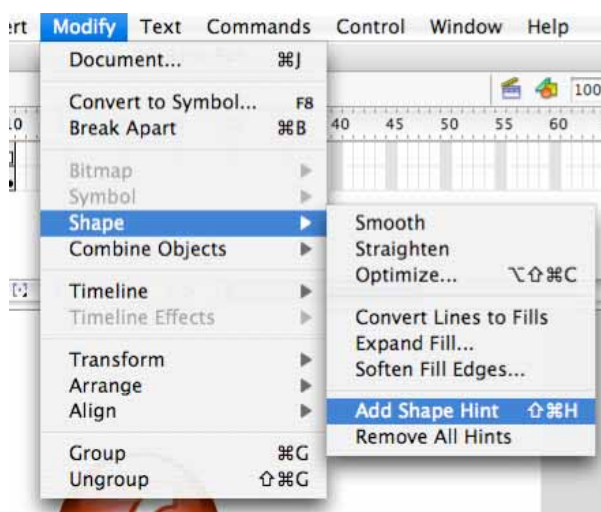


FIGURE 23 - SHAPE HINTS



FIGURE 24 - SHAPE HINTS

1.7 LIBRARIES, SYMBOLS, AND INSTANCES

A symbol is a graphic, button, or movie clip that you create in Macromedia Flash Basic 8 or Macromedia Flash Professional 8.

Caixas dentro de caixas – Ilustrar este conceito.

*You create the symbol only once; you can then reuse it throughout your document or in other documents. A symbol can include artwork that you import from another application. Any symbol that you create automatically becomes part of the library for the current document. For more information on the library, see *Managing media assets with the library*.*

*When you create a symbol in the authoring environment, each symbol has its own Timeline. You can add frames, keyframes, and layers to a symbol Timeline, just as you can to the main Timeline. For more information, see *Using the Timeline in Getting Started with Flash*. If the symbol is a movie clip or a button, you can control the symbol with ActionScript. For more information, see *Handling Events in Learning ActionScript 2.0 in Flash*.*

An instance is a copy of a symbol located on the Stage or nested inside another symbol. An instance can be very different from its symbol in color, size, and function. Editing the symbol updates all of its instances, but applying effects to an instance of a symbol updates only that instance.

Using symbols in your documents dramatically reduces file size; saving several instances of a symbol requires less storage space than saving multiple copies of the contents of the symbol. For example, you can reduce the file size of your documents by converting static graphics, such as background images, into symbols and then reusing them. Using symbols can also speed SWF file playback, because a symbol needs to be downloaded to Flash Player only once.

You can share symbols among documents as shared library assets during authoring or at runtime. For runtime shared assets, you can link assets in a source document to any number of destination document, without importing the assets into the destination document. For assets shared during authoring, you can update or replace a symbol with any other symbol available on your local network. See Using shared library assets.

Each symbol has a unique Timeline and Stage, complete with layers. When you create a symbol you choose the symbol type, depending on how you want to use the symbol in your document.

Use graphic symbols for static images and to create reusable pieces of animation that are tied to the main Timeline. Graphic symbols operate in sync with the main Timeline. Interactive controls and sounds won't work in a graphic symbol's animation sequence.

Use button symbols to create interactive buttons that respond to mouse clicks, rollovers, or other actions. You define the graphics associated with various button states, and then assign actions to a button instance. For more information, see *Handling Events in Learning ActionScript 2.0 in Flash*.

Use movie clip symbols to create reusable pieces of animation. Movie clips have their own multiframe Timeline that is independent from the main Timeline--think of them as nested inside a main Timeline that can contain interactive controls, sounds, and even other movie clip instances. You can also place movie clip instances inside the Timeline of a button symbol to create animated buttons.

EXERCÍCIO:

Criar Símbolos;
Alternar o método entre criar e editar símbolos;
A biblioteca de Ficheiros e Símbolos;

1.8 MOTION TWEENS

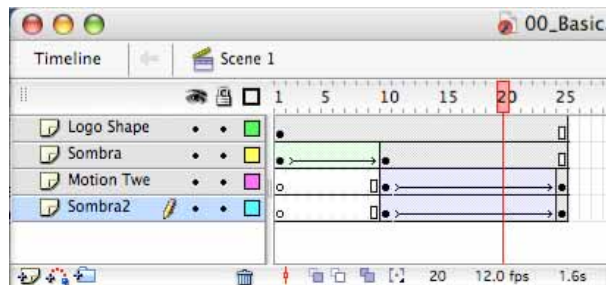


FIGURE 25 - MOTION TWEENS

EXERCÍCIO:

Motion Tweens básicos e com *Layer/Motion Guides*;
 Editar múltiplas *frames*;
Easing;

1.9 TIMELINE EFFECTS

Flash includes prebuilt Timeline effects that let you create complex animations with a minimal number of steps. You can apply Timeline effects to the following objects:

- *Text*
- *Graphics, including shapes, groups, and graphic symbols*
- *Bitmap images*
- *Button symbols*

EXERCÍCIO:

Adicionar, editar e remover *Timeline Effects*.

1.10 MASKING

Using a mask layer provides a simple way to selectively reveal portions of the layer or layers below it. Masking requires making one layer a mask layer and the layers below it masked layers.

You'll use the rectangular shape on the Stage to mask part of the road graphic and animation so that the animation fits better on the Stage.

On the Stage, with the Selection tool selected, click the rectangular shape below the road.

Drag the shape straight up and align the left edge of the shape with the left edge of the road. Right-click (Windows) or Control-click (Macintosh) the Mask layer in the Timeline and select Mask from the context menu.

The layer converts to a mask layer, indicated by a blue diamond-shaped icon. The layer immediately below the layer is linked to the mask layer. The masked layer's name is indented, and its icon changes to a blue layer icon.

In the Timeline, drag the Road layer to the Mask layer, placing it below the Car layer. The mask layer and the layers it masks are automatically locked.

To view the mask effect, select Control > Test Movie.

When you finish viewing the mask effect, close the SWF file window to return to your document.

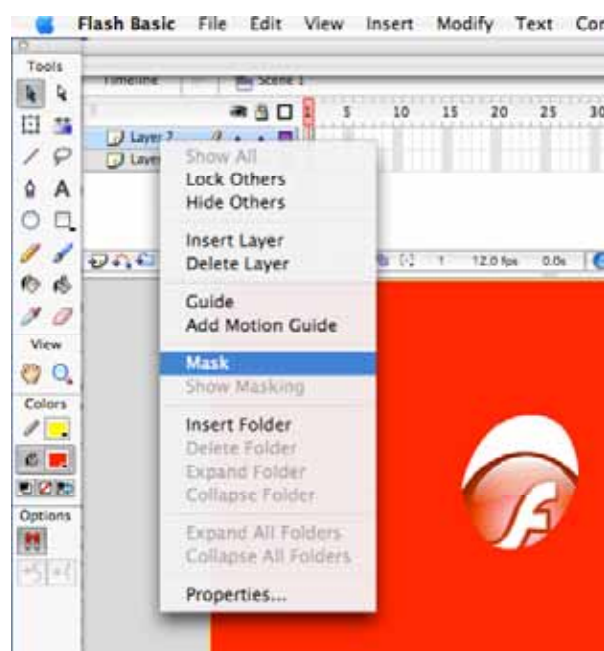


FIGURE 26 - ADICIONAR UMA MÁSCARA

EXERCÍCIO:

Máscaras e Animated masks

1.11 TIPOGRAFIA

You can include text in your Macromedia Flash Basic 8 and Flash Professional 8 applications in a variety of ways. You can create text blocks containing static text, text whose contents and

appearance you determine when you author the document. You can also create dynamic or input text fields. Dynamic text fields display dynamically updating text, such as sports scores, stock quotes, or news headlines. Input text fields allow users to enter text for forms, surveys, or other purposes.

Just like movie clip instances, text field instances are *ActionScript* objects that have properties and methods. By giving a text field an instance name, you can manipulate it with *ActionScript*. However, unlike with movie clips, you cannot write *ActionScript* code inside a text instance, because text instances don't have Timelines.

You can orient text horizontally, with left-to-right flow, or vertically (static text only), with left-to-right or right-to-left flow. You can select the following attributes for text: font, point size, style, color, tracking, kerning, baseline shift, alignment, margins, indents, and line spacing.

You can transform text as you would an object--rotating, scaling, skewing, and flipping it--and still edit its characters. See *About transforming text*. When you're working with horizontal text, you can link text blocks to URLs and make it selectable. See *Linking text to a URL (horizontal text only)*.

When you work with Flash FLA files, Flash substitutes fonts in the FLA file with other fonts installed on your system if the specified fonts are not on your system. You can select options to control which fonts are used in substitution. Substitute fonts are used for display on your system only. The font selection in the FLA file remains unchanged.

Flash also lets you create a symbol from a font so that you can export the font as part of a shared library and use it in other Flash documents.

Creating font symbols

To use a font as a shared library item, you can create a font symbol in the Library panel. You then assign the symbol an identifier string and a URL where the document containing the font symbol will be posted. In this way, you can link to the font and use it in a Flash application.

To create a font symbol:

1. Open the library to which you want to add a font symbol.

2. Select New Font from the options menu in the upper right corner of the Library panel.
3. In the Font Symbol Properties dialog box, enter a name for the font symbol in the Name text box.
4. Select a font from the Font menu or enter the name of a font in the Font text box.
5. If you want to apply a style to the font, select Bold or Italic.
6. (Optional) If you want to use bitmap fonts for your font symbol, select the Bitmap Text check box, and enter a font size in the Size text box. For information on bitmap text, see Setting anti-aliasing options for text.
7. Click OK.

To assign an identifier string to a font symbol:

1. Select the font symbol in the Library panel.
2. Do one of the following:
3. Select Linkage from the options menu in the upper right corner of the Library panel.
4. Right-click (Windows) or Control-click (Macintosh) the font symbol name in the Library panel, and select Linkage from the context menu.
5. Under Linkage in the Linkage Properties dialog box, select Export for Runtime Sharing.
6. In the Identifier text box, enter a string to identify the font symbol.
7. In the URL text box, enter the URL where the SWF file that contains the font symbol will be posted.
8. Click OK.

You can break text apart and reshape its characters. For additional text-handling capabilities, you can manipulate text in FreeHand and import the FreeHand file into Flash, or export the file from FreeHand as a SWF file.

You can preserve rich text formatting in text fields, using HTML tags and attributes.

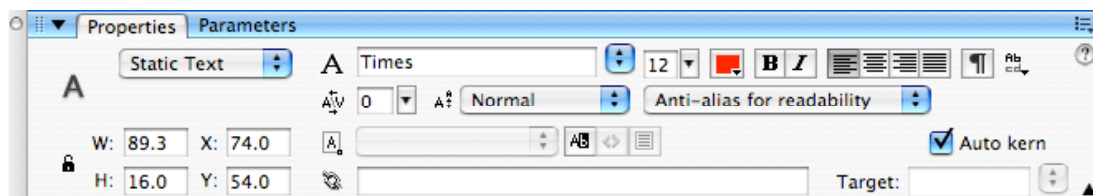


FIGURE 27 - PROPERTIES INSPECTOR (TIPOGRAFIA)

EXERCÍCIO:

Editar opções do Texto (*Static*, *Dinamic Single*, *Multi-line e Scrollable*)

Opções de *Aliasing*;

1.12 BITMAPS

When you import a bitmap into Flash, you can modify that bitmap and use it in your Flash document in a variety of ways. You can apply compression and anti-aliasing to imported bitmaps to control the size and appearance of bitmaps in your Flash applications.

Flash lets you break apart a bitmap into editable pixels. The bitmap retains its original detail but is broken into discrete areas of color. When you break a bitmap apart, you can select and modify areas of the bitmap with the Flash drawing and painting tools. Breaking apart a bitmap also lets you sample the bitmap with the Eyedropper tool to use it as a fill.

You can edit an imported bitmap in Fireworks or another external image editor by starting the editing application from within Flash.

To convert a bitmap's image to a vector graphic, you can trace the bitmap. Performing this conversion enables you to modify the graphic as you do other vector artwork in Flash.

The Trace Bitmap command converts a bitmap into a vector graphic with editable, discrete areas of color. This command lets you manipulate the image as a vector graphic; it is also useful if you want to reduce file size.

When you convert a bitmap to a vector graphic, the vector graphic is no longer linked to the bitmap symbol in the Library panel.

To convert a bitmap to a vector graphic:

1. Select a bitmap in the current scene.
2. Select Modify > Bitmap > Trace Bitmap.

3. Enter a Color Threshold value between 1 and 500.
4. When two pixels are compared, if the difference in the RGB color values is less than the color threshold, the two pixels are considered the same color. As you increase the threshold value, you decrease the number of colors.

If a Flash document displays an imported bitmap at a larger size than the original, the image may be distorted. Preview imported bitmaps to be sure that images are displayed properly.

A melhor opção é saber as definições da aplicação final e trabalhar com os bitmaps em função disso. Outras opções podem ser a optimização dos ficheiros finais com diferentes resoluções ou ainda aumentar a resolução disponível das imagens através das opções de compressão e Output para o Flash Player gerir. Este ultimo é o que produz piores resultados

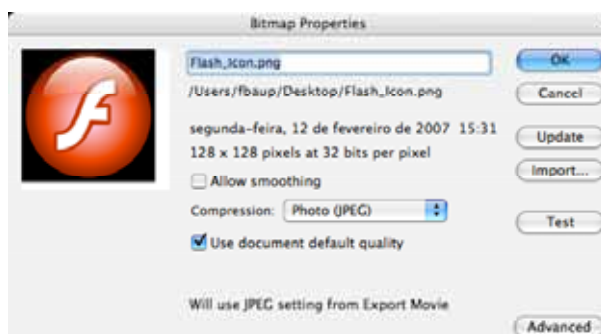


FIGURE 28 - IMPORT OPTIONS

EXERCÍCIO:

Importar e modificar as opções de compressão;
Sequências de Bitmaps;
Tracing;
Modificar a Opacidade.

1.13 FILTERS AND BLEND MODES (PROFESSIONAL ONLY)

Working with Filters

Working with Blend Modes

1.14 BUTTONS

Buttons are actually four-frame interactive movie clips. When you select the button behavior for a symbol, Flash creates a Timeline with four frames. The first three frames display the button's three possible states; the fourth frame defines the active area of the button. The Timeline doesn't actually play; it simply reacts to pointer movement and actions by jumping to the appropriate frame.

To make a button interactive, you place an instance of the button symbol on the Stage and assign actions to the instance. You must assign the actions to the instance of the button in the document, not to frames in the button's Timeline.

Each frame in the Timeline of a button symbol has a specific function:

The first frame is the Up state, representing the button whenever the pointer is not over the button.

The second frame is the Over state, representing the button's appearance when the pointer is over the button.

The third frame is the Down state, representing the button's appearance as it is clicked.

The fourth frame is the Hit state, defining the area that responds to the mouse click. This area is invisible in the SWF file.

You can also create a button using a movie clip symbol or a button component. There are advantages to using each type of button, depending on your needs. Creating a button using a movie clip enables you to add more frames to the button or add more complex animation. However, movie clip buttons have a larger file size than button symbols. Using a button component allows you to bind the button to other components, to share and display data in an application. Button components also include prebuilt features, such as accessibility support, and can be customized. Button components include the PushButton and RadioButton.



FIGURE 29 - INSERTING A BUTTON

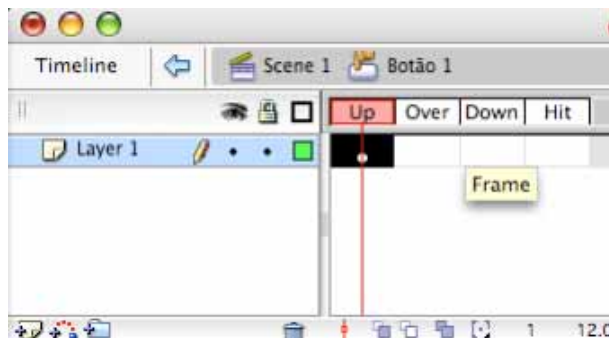


FIGURE 30 - BUTTON TIMELINE

EXERCÍCIO:

Tipos de Botões (Inserir, Editar e Modificar tipos na Timeline e na Library);
Basic, Rollover, Rollover button with down state, Hit State e Invisible button

1.15 MOVIE CLIPS

A movie clip symbol is analogous in many ways to a document within a document. **This symbol type has its own Timeline independent of the main Timeline.** You can add movie clips within other movie clips and buttons to create nested movie clips. You can also use the Property inspector to assign an instance name to an instance of a movie clip, and then reference the instance name in ActionScript.



FIGURE 31 - SYMBOLS HIERARCHY

ANIMATED ROLLOVER BUTTONS

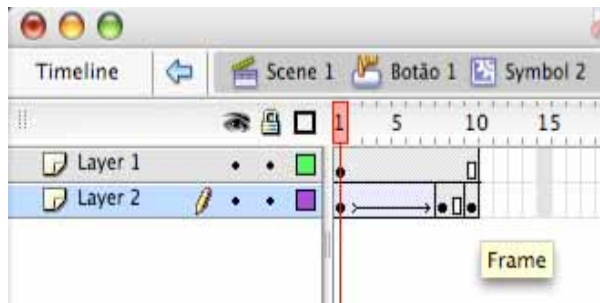


FIGURE 32 - BOTÃO ANIMADO

1.16 ACTIONSCRIPT ESSENTIALS

Time-based vs. user-based actions

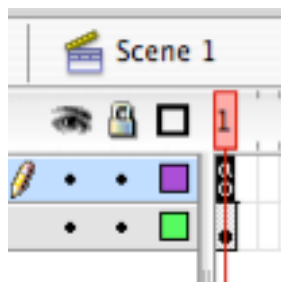


FIGURE 33 - TIME BASED ACTIONSCRIPT

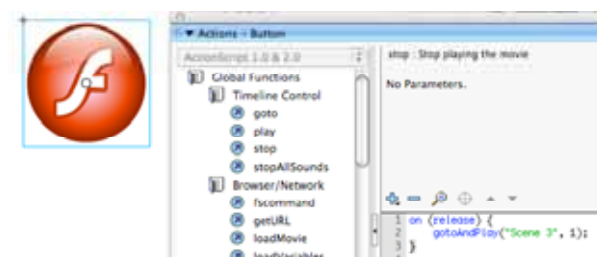


FIGURE 34 - USER BASED ACTIONSCRIPT

SCRIPT ASSIST

Script Assist lets you build scripts by selecting items from the Actions toolbox, the list on the left side of the Actions panel. (You can also select actions from the Add (+) pop-up menu.) The

Actions toolbox separates items into categories such as actions, properties, and objects, and also provides an index category that lists all items alphabetically. When you click an item once, its description appears at the upper right of the panel. When you double-click an item, it adds the item to the scrolling list on the right side of the panel in the Script pane.

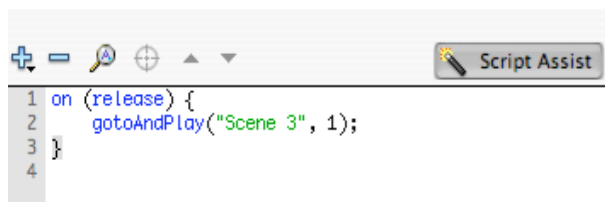


FIGURE 35 - SCRIPT ASSIST (ACTIONS PANEL)

In Script Assist mode, you can add, delete, or change the order of statements in the Script pane; you can also enter parameters for actions in text boxes above the Script pane. Script Assist also lets you find and replace text, view script line numbers, and pin a script--that is, keep a script in the Script pane when you click away from the object or frame.

STOP ON FRAME



FIGURE 36 - STOP ON FRAME (SCRIPT ASSIST)

GETURL

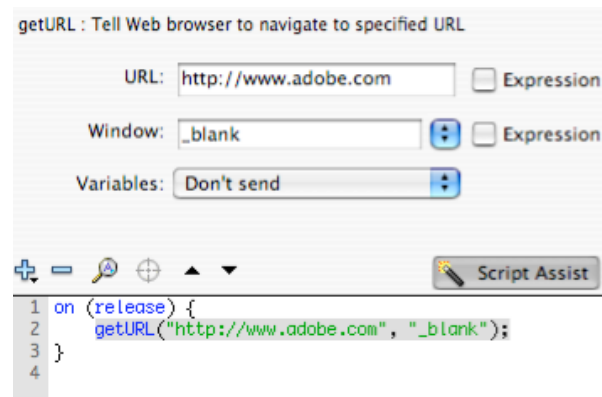


FIGURE 37 - GET URL (SCRIPT ASSIST)

SLIDE SHOW

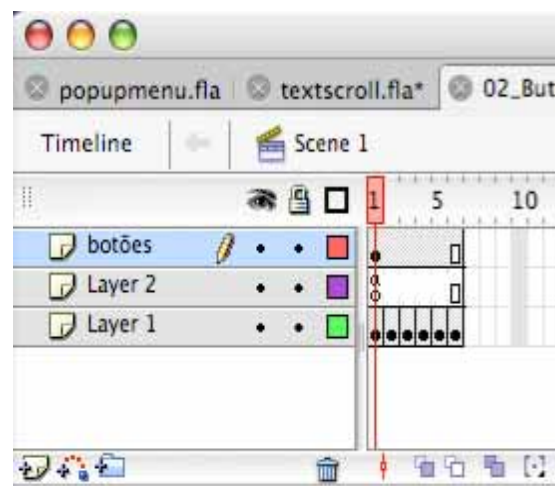


FIGURE 38 - CREATING A SLIDESHOW

EXERCÍCIO:

Import Options (Images)
GoTo Frame/Button Actions

Criar um ficheiro de Flash 800 x 450 px;
Importar uma sequencia de imagens e colocar em slideshow automático;
Gravar como [nome]slideshow.fl a

Criar uma nova cena no mesmo filme e repetir o exercício, mas com a opção de navegação manual;

CREATING A POP-UP MENU (BUTTONS + FRAME LABELS)

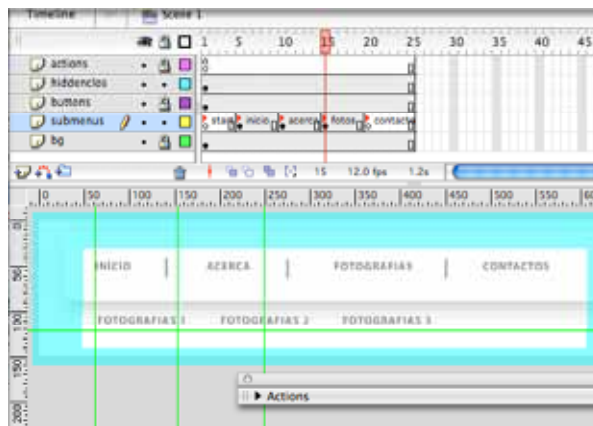


FIGURE 39 - POPUP MENU (FLASH)

EXERCÍCIO:

Criar um ficheiro Flash 800 x 150 px;

Menu *popup* com Início, Textos, Imagens, Animação, Filmes, Sons, ActionScript;

1 subnível para cada opção com 3 opções;

SCROLLING TEXT

There are several ways to create scrolling text in Flash. You can easily make dynamic text fields scrollable using menu commands or the text block handle.

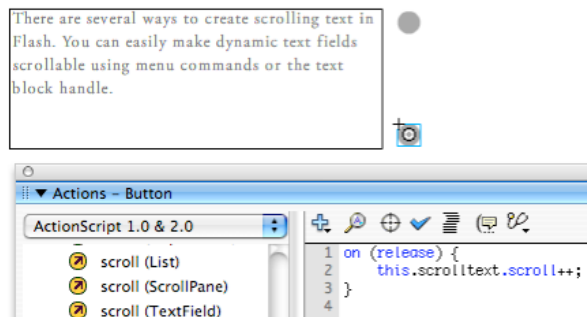


FIGURE 40 - SCROLLING TEXT

You can also add a ScrollBar component to a text field to make it scroll.

If you want to use ActionScript, you can use the scroll and maxscroll properties of the TextField object to control vertical scrolling and the hscroll and maxhscroll properties to control horizontal scrolling in a text block.

EXERCÍCIO:

Criar um Ficheiro de Flash com 800 x 450 px;

2 Cenas – primeira com uma caixa de texto scrollable com components e a segunda com botões manuais;

Gravar como [nome]texto fla

CONTROLLING MOVIE CLIPS (TARGETS)

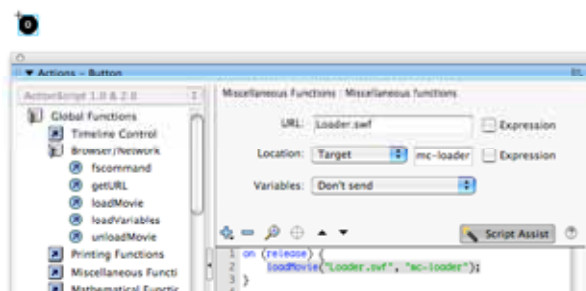


EXERCÍCIO:

Criar um ficheiro Flash com 800 x 450 px;

Criar uma animação do Logótipo do Flash a Saltar em *loop*;
 Criar um botão para parar a animação e reiniciar a mesma;
 Gravar o ficheiro com [nome].ani macao. fl a

LOADING SWFS AND IMAGES



BUILDING A SIMPLE PRELOADER 1



FIGURE 41 - PRELOADER DE 2 FRAMES



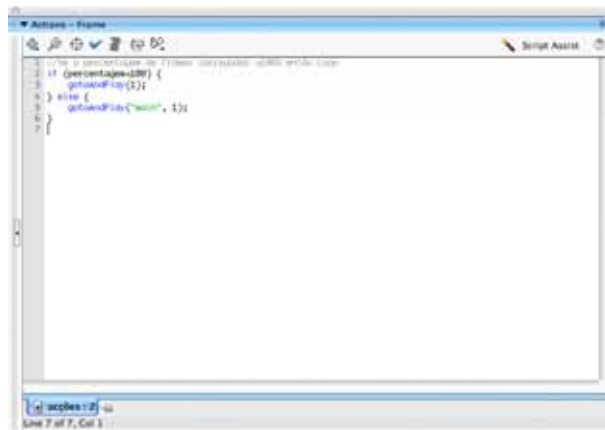
EXERCÍCIO:

Criar um preloader (desafio avançado – demonstração com 2 frames – Time based script);

```
/*
Preloader para filmes Flash
Imagens e ActionScript © Pedro Amado 2005.04
*/

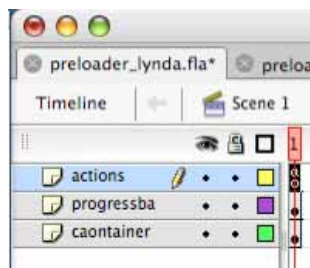
// definir variavel com o tamanho total em frames de todo o filme;
total_frames = this._totalframes;
//confirmar o tamanho;
//trace("Total de Frames do Filme: "+total_frames);
// Definir variavel que actualiza as frames carregadas;
current_frames = this._framesloaded;
//confirmar frames carregadas;
//trace("Frames Carregadas: "+current_frames);
//Definir a percentagem carregada;
//Para o caso do preload em frames percentagem obtida em valor de frames;
//percentagem = int((100*current_frames)/total_frames);

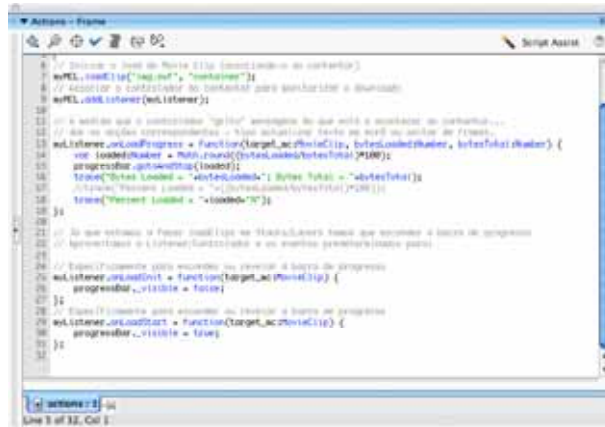
//Confirmar percentagem;
//trace("Percentagem carregada: "+percentagem+"%");
//mostrar total de frames;
var_frames = total_frames;
//mostrar a frame actual a a carregar;
var_frame = current_frames;
//bytes loaded text;
var_bytes = this.getBytesLoaded();
//Total bytes;
var_bytetestotal = this.getBytesTotal();
//mostrar percentagem;
percent_loaded = String(percentagem+"% do filme");
//Para obter a percentagem em Bytes;
percentagem = int((100*this.getBytesLoaded())/this.getBytesTotal());
//Forçar o update do filme;
//updateAfterEvent();
```



```
//Se a percentagem de frames carregadas <100% então Loop  
if (percentagem<100) {  
    gotoAndPlay(1);  
} else {  
    gotoAndPlay("main", 1);  
}
```

BUILDING A SIMPLE PRELOADER 2





EXERCÍCIO:

Criar um preloader (desafio avançado – demonstração com 1 frame – Object based script);

```

//Iniciar um objecto específico (contentor) para carregar MovieClips
var myMCL:MovieClipLoader = new MovieClipLoader();
// Inicializar um "controlador" de loading do contentor...
var myListener:Object = new Object();

// Iniciar o load do Movie Clip (associando-o ao contentor)
myMCL.loadClip("img.swf", "container");
// Associar o controlador ao contentor para monitorizar o download;
myMCL.addListener(myListener);

// À medida que o controlador "grita" mensagens do que está a acontecer ao
contentor...
// dar as acções correspondentes - tipo actualizar texto em ecrã ou saltar de
frames.
myListener.onLoadProgress = function(target_mc:MovieClip, bytesLoaded:Number,
bytesTotal:Number) {
    var loaded:Number = Math.round((bytesLoaded/bytesTotal)*100);
    progressBar.gotoAndStop(loaded);
    trace("Bytes Loaded = "+bytesLoaded+"; Bytes Total = "+bytesTotal);
    //trace("Percent Loaded = "+((bytesLoaded/bytesTotal)*100));
    trace("Percent Loaded = "+loaded+"%");
};

```

```
// Já que estamos a fazer loadClips em Stacks/Layers temos que esconder a barra de
progresso
// Aproveitamos o Listener/Controlador e os eventos predeterminados para:

// Especificamente para esconder ou revelar a barra de progresso
myListener.onLoadInit = function(target_mc:MovieClip) {
    progressBar._visible = false;
};
// Especificamente para esconder ou revelar a barra de progresso
myListener.onLoadStart = function(target_mc:MovieClip) {
    progressBar._visible = true;
};
```

BEHAVIORS

EXERCÍCIO:

Repetir o comportamento Goto com o Script Assist e user/Time based Actions
 Voltar a falar nisto no Som e vídeo...

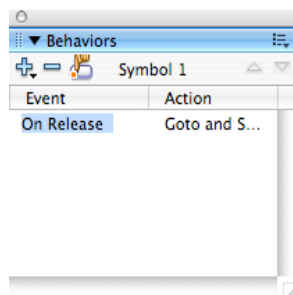


FIGURE 42 - BEHAVIOURS PANEL

1.17 COMPONENTS

WHAT ARE COMPONENTS?

Components are movie clips with parameters that let you modify their appearance and behavior.

A component can provide a wide range of functionality. A component can be a simple user interface control, such as a radio button or a check box, or it can be a complicated control element, such as a media controller or a scroll pane. A component can even be nonvisual, such as the focus manager that lets you control which object receives focus in an application.

Components let you separate coding and design. They also let you reuse code, and download components created by other developers.

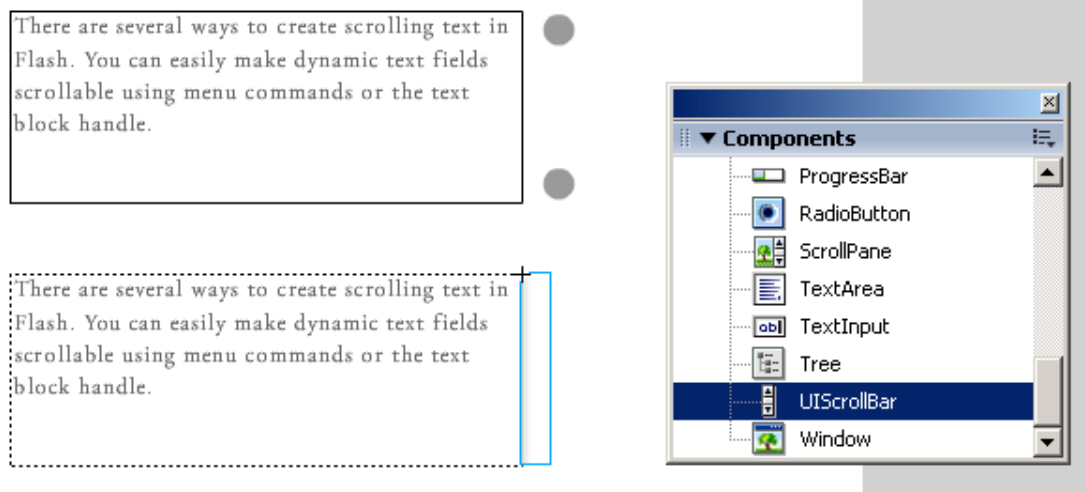


FIGURE 43 - UI SCROLLBAR COMPONENT

EXERCÍCIO:

UI Scrollbar Component vs. *Text scroll* manual.

1.18 SOM

IMPORTAR SONS

To add a sound to a document from the library, you assign the sound to a layer and set options in the Sound controls in the Property inspector. It is recommended that you place each sound on a separate layer.

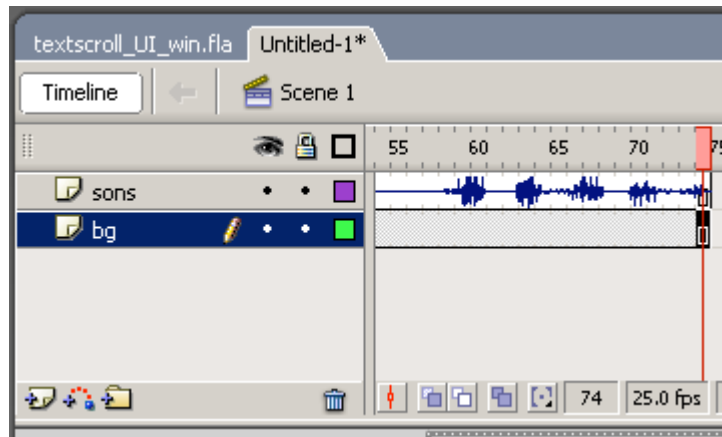


FIGURE 44 - SONS EM LAYERS

FORMATOS SUPORTADOS

You can import the following sound file formats into Flash:

WAV (Windows only)

AIFF (Macintosh only)

MP3 (Windows or Macintosh)

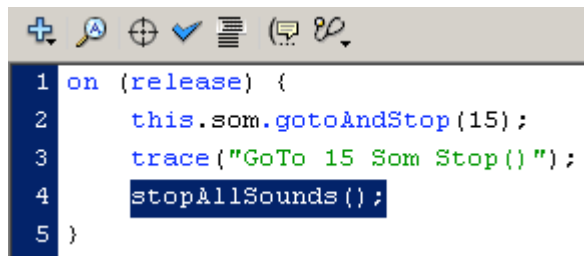
SOM NA TIMELINE E EM BOTÕES (BEHAVIOUR, GOTO E ACTIONSCRIPT)

Adding sounds to buttons

You can associate sounds with the different states of a button symbol. Because the sounds are stored with the symbol, they work for all instances of the symbol.

1. Select the button in the Library panel.
2. Select Edit from the options menu in the upper right corner of the panel.
3. In the button's Timeline, add a layer for sound.
4. In the sound layer, create a regular or blank keyframe to correspond with the button state to which you want to add a sound.
5. For example, to add a sound that plays when you click the button, create a keyframe in the frame labeled Down.
6. Click the keyframe you created.
7. Select Window > Properties.
8. In the Property inspector, select a sound file from the Sound pop-up menu.
9. Select Event from the Synchronization pop-up menu.

INICIAR E PARAR SOM



```
1 on (release) {  
2     this.sound gotoAndStop(15);  
3     trace("GoTo 15 Som Stop()");  
4     stopAllSounds();  
5 }
```

FIGURE 45 - STOPALLSOUNDS();

EXERCÍCIO:

Music on/off

1.19 VIDEO

Macromedia Flash Basic 8 and Macromedia Flash Professional 8 provide several ways for you to include video in your Flash documents. How you choose to deploy your video will determine how you create your video content, and how you integrate it for use with Flash. This section describes different video features in Flash, and how you can use them to incorporate video content.

Flash provides several methods for integrating and delivering video content. The ways in which you can incorporate video into Flash are:

***Streaming video content** Flash lets you host video files using Flash Communication Server, a server solution optimized for the delivery of streaming, real-time media. You can import video clips stored locally into your Flash documents, and later upload them to the server. This allows you to more easily assemble and develop Flash content. You can also use the new FLVPlayback component or ActionScript to control video playback and provide intuitive controls for users to interact with the video.*

You can host your own Flash Communication Server, or, you can use a hosted Flash Video Streaming Service (FVSS). Macromedia has partnered with several content delivery network (CDN) providers to offer hosted services for delivering on-demand Flash Video across high-performance, reliable networks. Built with Flash Communication Server and integrated directly into the delivery, tracking, and reporting infrastructure of the CDN network, FVSS provides the

most effective way to deliver Flash Video to the largest possible audience without the hassle of setting up and maintaining your own streaming server hardware and network.

Progressively downloading video from a web server *If you don't have access to Flash Communication Server or FVSS, you can still enjoy the benefits of downloading video from an external source when you use progressive downloading. Progressively downloading a video clip from a web server doesn't provide the same real-time performance that Flash Communication Server does; however, you can use relatively large video clips, and keep the size of your published SWF files to a minimum. You can also use the new FLVPlayback component or ActionScript to control video playback and provide intuitive controls for users to interact with the video.*

Importing embedded video *You can import video clips into Flash as embedded files. As with an imported bitmap or vector artwork file, an embedded video file becomes part of the Flash document. For this reason, you can only import very short duration video clips. For information on file formats supported for importing embedded video, see About embedding video in a SWF file.*

Importing video in QuickTime format *You can import video clips in QuickTime format as linked files. Flash documents that contain linked QuickTime video must be published in QuickTime format. A linked video file does not become part of the Flash document. Instead, the Flash document maintains a pointer to the linked file.*

Importing FLV files in the Library *You can import video clips in Macromedia Flash Video (FLV) format directly into Flash. When you import FLV files, you use the encoding options already applied to the files. You do not need to select encoding options during import. There are several options for controlling the playback of video files:*

Using the FLVPlayback component *New to Flash Professional 8, the FLVPlayback component lets you quickly add a full-featured FLV or MP3 playback control to your Flash movie. FLVPlayback provides support for both progressive downloading and streaming FLV files. FLVPlayback lets you easily create intuitive video controls for users to control video playback, as well as the ability to apply premade skins, or to apply your own custom skins to the video interface.*

You can use video behaviors (prewritten ActionScript scripts) to control video playback. *For more information, see Controlling video playback using behaviors.*

*Controlling video playback in the Timeline If you are comfortable with **ActionScript**, you can write custom **ActionScript** to control video playback. You can play or stop a video, jump to a frame, and control video in other ways. You can also display a live video stream from a camera.*

IMPORTAR E COMPRIMIR VIDEO

The Video Import wizard provides a streamlined interface for importing video into a Flash document. The wizard lets you select whether to import a video clip as a streamed, progressively downloaded, embedded, or linked file. Also, depending on the location of your file, the Video Import wizard provides a series of options for different deployments.

If the video clip you wish to import is located on your local computer, you can browse to it and import the video. You can also import a video stored on a remote web server or Flash Communication Server by providing the URL of the file.

Flash 8 Video Encoder to compress video (Pro only)



Flash documents with embedded video can be published as SWF files.

Flash documents with linked video must be published in QuickTime format.

The following video file formats are supported for importing embedded video if QuickTime 7 is installed:

Audio Video Interleaved .avi

Digital video .dv

Motion Picture Experts Group .mpg, .mpeg

QuickTime video .mov

EMBEDDING VIDEO IN A SWF FILE

When you import a video clip as an embedded file, you select options in the Video Import wizard for embedding, encoding, and editing the video. Click the Next button to advance through panes in the wizard, and click the Back button to return to previous panes.

You can import video clips as embedded files in several file formats, depending on your system. For information on supported video file formats, see [Supported file formats for video](#). You can preview frames of an imported video by dragging the playhead along the Timeline. However, the sound does not play back. To preview the video with sound, use the Test Movie command. For more information, see [Testing document download performance](#)

When you import a video as an embedded file, you have the option to edit the video prior to importing it. You can also apply customized compression settings, including bandwidth or video quality settings. You select editing and encoding options in the Video Import wizard.

NOTE

Once a video clip is imported, it cannot be edited.

To embed video within the SWF file:

To import the video clip into the current Flash document, select

1. File > Import > Import Video.
2. The Import Video wizard is displayed.
3. Select the video clip on your local computer that you want to import.
4. Select the Embed Video in SWF and Play in Timeline check box.
5. Choose the symbol type with which to embed the video within the SWF.
6. You can choose to embed the video as embedded video, a movie clip, or graphic symbol. The method you choose depends on how you intend to integrate the video into the SWF and interact with it:

Embed in the Timeline *The most common choice is to integrate the video clip as an embedded video within the Timeline. If you're using the video clip for linear playback in the Timeline, importing the video into the Timeline is the most appropriate method.*

Embed as a movie clip *When you work with embedded video, a best practice is to place video inside a movie clip instance, because you have the most control over the content. The video's Timeline plays independently from the main Timeline. You do not have to extend your main*

Timeline by many frames to accommodate the video, which can make working with your FLA file difficult.

Embed as a graphic symbol Embedding a video clip as a graphic symbol means that you cannot interact with the video using ActionScript (typically you use graphic symbols for static images and to create reusable pieces of animation that are tied to the main Timeline). For this reason, you'll rarely want to embed a video as a graphic symbol. For more information, see *Types of symbols*.

Import the video clip directly onto the Stage (and the Timeline) or as a library item. By default, Flash places the video you import on the Stage. If you prefer that the video be imported only into the library, deselect the Place Instance on Stage check box.

If you're creating a simple video presentation with linear narration and little to no interaction, accept the default setting and import the video to the Stage. If, however, you want to create a more dynamic presentation, are working with multiple video clips, or intend to add dynamic transitions or other elements using ActionScript, import the video into the library. Once a video clip is in the library, you can customize it by converting it into a MovieClip object that can be more easily controlled using ActionScript.

By default, Flash expands the Timeline to accommodate the playback length of the video clip you are embedding.

(Optional) If you want to edit the video clip using the Video Import wizard, select the Edit Video First check box.

The Video Import wizard includes basic video editing options that let you crop and trim video clips. If you want to edit your video clip prior to embedding it in the Timeline, select this option.

(Optional) If the video clip is not yet encoded in the FLV format, select a Flash Video encoding profile.

For information on encoding profiles appropriate to your intended application, see [Selecting a video encoding profile](#).

Click Finish to close the Video Import wizard and complete the video import procedure.

The Video Import wizard encodes your video into the FLV format, and embeds the video into the SWF file. The video is displayed either on the Stage or in the library depending on the embedding options you chose.

In the Property inspector (Window > Properties), give the video clip an instance name, and make any modifications to the video clip's properties that you might need.

1.20 PUBLICAR CONTEÚDOS

When you're ready to deliver Macromedia Flash Basic 8 and Flash Professional 8 content to an audience, you can publish it for playback. By default, the Publish command creates a Flash SWF file and an HTML document that inserts your Flash content in a browser window. The Publish command also creates and copies detection files for Flash 4 and later. If you change publish settings, Flash saves the changes with the document. You can create publish profiles to name and save various configurations for the Publish Settings dialog box, in order to quickly publish documents a variety of ways. After you create a publish profile, you can export it for use in other documents, or for use by others working on the same project. For more information, see Using publish profiles.

To publish a Flash document, you select publish file formats and file format settings with the Publish Settings dialog box. Then you publish the Flash document using the Publish command. The publishing configuration that you specify in the Publish Settings dialog box is saved with the document. You can also create and name a publish profile so that the established publish settings are always available.

Depending on the options you specify in the Publish Settings dialog box, the Publish command creates the following files:

The Flash SWF file

Alternate images in a variety of formats that appear automatically when Flash Player is not available (GIF, JPEG, PNG, and QuickTime)

The supporting HTML document(s) required to show SWF content (or an alternate image) in a browser and control browser setting

Three HTML files (if you keep the default, Detect Flash Version, selected): the detection file, the content file, and the alternate file

Stand-alone projector files for Windows and Macintosh computers and QuickTime videos from Flash content (EXE, HQX, or MOV files, respectively) NOTE

To alter or update a SWF file created with the Publish command, you must edit the original Flash document and then use the Publish command again to preserve all authoring

information. Importing a Flash SWF file into Flash removes some of the authoring information.

PUBLICAR – PREFERENCES E FORMATOS

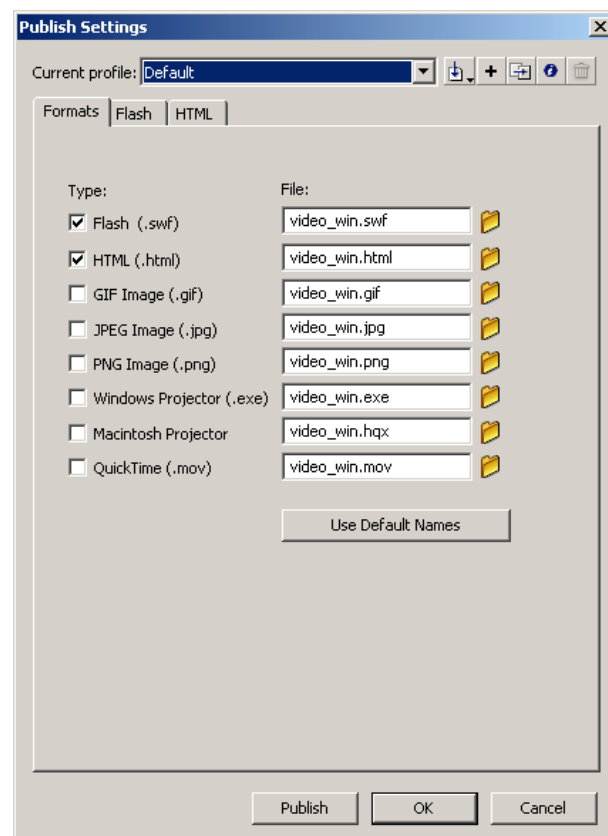


FIGURE 46 - PUBLISH PREFERENCES WINDOW

CRIAR UM PROJECTOR

FS COMMAND

`fscommand(command:String, parameters:String) : Void`

Lets the SWF file communicate with either Flash Player or the program that is hosting Flash Player, such as a web browser. You can also use the fscommand() function to pass messages to Macromedia Director, or to Visual Basic (VB), Visual C++, and other programs that can host ActiveX controls.

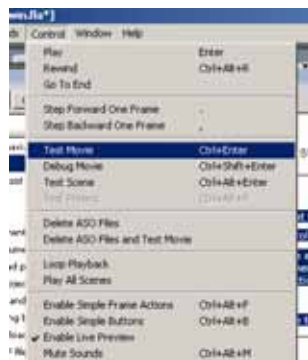
The `fscommand()` function lets a SWF file communicate with a script in a web page. However, script access is controlled by the web page's `allowScriptAccess` setting. (You set this attribute in the HTML code that embeds the SWF file--for example, in the `PARAM` tag for Internet Explorer or the `EMBED` tag for Netscape. When `allowScriptAccess` is set to "never", a SWF file cannot access web page scripts. With Flash Player 7 and later, when `allowScriptAccess` is set to "always", a SWF file can always access web page scripts. When `allowScriptAccess` is set to "sameDomain", scripting is allowed only from SWF files that are in the same domain as the web page; scripting is always allowed with previous versions of Flash Player. If `allowScriptAccess` is not specified in an HTML page, the attribute is set by default to "sameDomain" for SWF files of version 8 and later, and to "always" for SWF files of version 7 and earlier.

Usage 1: To use `fscommand()` to send a message to Flash Player, you must use predefined commands and parameters. The following table shows the values that you can specify for the `fscommand()` function's command and parameters parameters. These values control SWF files that are playing in Flash Player, including projectors. (A projector is a SWF file saved in a format that can run as a stand-alone application--that is, without Flash Player.)

Command	Parameter	Purpose
quit	None	Closes the projector.
fullscreen	true or false	Specifying true sets Flash Player to full-screen mode. Specifying false returns the player to normal menu view.
allowscale	true or false	Specifying false sets the player so that the SWF file is always drawn at its original size and never scaled. Specifying true forces the SWF file to scale to 100% of the player.
showmenu	true or false	Specifying true enables the full set of context menu items. Specifying false hides all of the context menu items except About Flash Player and Settings.
exec	Path to application	Executes an application from within the projector.
trapallkeys	true or false	Specifying true sends all key events, including

accelerator keys, to the onClipEvent(keyDown/keyUp) handler in Flash Player.

TESTING MOVIES



Test the SWF file

Save and test the document to ensure that the dynamic text loads correctly.

1. Select File > Save; then select Control > Test Movie.
2. In the SWF file window, text from the external text file should appear in the dynamic text field that you created. (If the text does not appear as expected, check that you entered the instance name correctly: newFeatures_txt. Also check that you saved your copy of the practice file in the same folder as the original text_start file.)
3. Type in the input text fields.
4. When you finish testing the file, close the SWF file window.

THE BANDWIDTH PROFILER AND SIMULATING A DOWNLOAD

Testing document download performance

Flash Player attempts to meet the frame rate you set; the actual frame rate during playback can vary on different computers. If a document that is downloading reaches a particular frame before the frame's required data has downloaded, the document pauses until the data arrives.

To view downloading performance graphically, you can use the Bandwidth Profiler, which shows how much data is sent for each frame according to the modem speed you specify. The Bandwidth Profiler is divided into two panes. The left pane shows information about the document, the download settings, the state, and streams, if any are included. The right pane shows information about individual frames in the document.

In simulating the downloading speed, Flash uses estimates of typical Internet performance, not the exact modem speed. For example, if you select to simulate a modem speed of 28.8 Kbps, Flash sets the actual rate to 2.3 Kbps to reflect typical Internet performance. The profiler also compensates for the added compression support for SWF files, which reduces the file size and improves streaming performance.

When external SWF files, GIF and XML files, and variables are streamed into a player by using ActionScript calls such as loadMovie and getUrl, the data flows at the rate set for streaming. The stream rate for the main SWF file is reduced based on the reduction of bandwidth caused by the additional data requests. It's helpful to test your document at each speed and on each computer that you plan to support. This helps you ensure that the document doesn't overburden the slowest connection and computer for which it is designed.

You can also generate a report of frames that are slowing playback and then optimize or eliminate some of the content in those frames. For more information, see [Optimizing Flash documents](#).

To change the settings for the SWF file created using the Test Movie and Test Scene commands, use File > Publish Settings. For more information, see [Publishing Flash documents](#).

To test download performance:

Do one of the following:

1. Select Control > Test Scene or Control > Test Movie.
2. If you test a scene or document, Flash publishes the current selection as a SWF file using the settings in the Publish Settings dialog box. (See [Publishing Flash documents](#).) The SWF file opens in a new window and begins playing immediately.
3. Select View > Download Settings, and select a download speed to determine the streaming rate that Flash simulates: 14.4 Kbps, 28.8 Kbps, 56 Kbps, DSL, T1 or a user setting. To enter a custom user setting, select Customize.

When viewing the SWF file, select View > Bandwidth Profiler to show a graph of the downloading performance.

The left side of the profiler displays information about the document, its settings, its state, and streams, if any are included in the document.

The right section of the profiler shows the Timeline header and graph. In the graph, each bar represents an individual frame of the document. The size of the bar corresponds to that frame's size in bytes. The red line beneath the Timeline header indicates whether a given frame streams in real time with the current modem speed set in the Control menu. If a bar extends above the red line, the document must wait for that frame to load.

Close the test window to return to the authoring environment.

After you set up a test environment using the Bandwidth Profiler, you can open any SWF file directly in test mode. The file opens in a Flash Player window, using the Bandwidth Profiler and other selected viewing options.

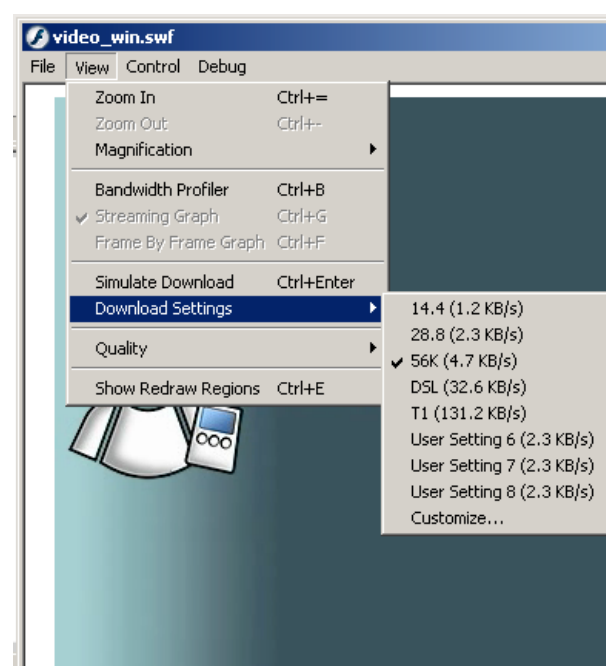


FIGURE 47- BANDWIDTH PROFILER

GENERATING A SIZE REPORT

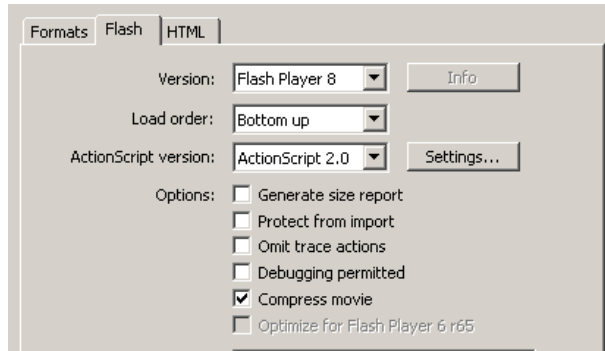


FIGURE 48 - SIZE REPORT

THE FLASH PLAYER DETECTION KIT

Configuring publish settings for Flash Player detection

You can configure your document to detect your users' Flash Player version. If you select *Detect Flash Version* in the *Publish Settings* dialog box, your SWF is embedded in a web page that includes Flash Player detection code. If the detection code finds an acceptable version of Flash Player installed on the end user's computer, the SWF file plays as designed. If an end user does not have the version of Flash needed to view the SWF, an HTML page with a link from which to download the latest version of Flash Player is displayed. Flash Player detection is available only for publish settings set to Flash Player 4 or later, and for SWF files embedded in the Flash Only or Flash HTTPS templates.

Flash Player 5 and later are installed on 98% of Internet-connected computers, making Flash Player detection a reasonable method by which to ensure that end users have the correct version of Flash installed with which to view your content.

To enable Flash Player detection:

1. Select **File > Publish Settings**, and select the **HTML** tab.
2. Select either the **Flash Only** or **Flash HTTPS** template from the **Template** pop-up menu. The **Flash Only** and **Flash HTTPS** templates support the new single-page HTML detection kit. Selecting either of these templates enables the **Detect Flash Version** check box and the version number text boxes.
3. Select the **Detect Flash Version** check box.
4. (Optional) You can use the **Major Revision** and **Minor Revision** text boxes to specify precise revisions of Flash Player. For example, you

might specify Flash Player version 7.0.2 if it provided a feature specific to displaying your SWF file.

When you publish your SWF file, Flash creates a single HTML page in which to embed the SWF and the Flash Player detection code. If an end user does not have the version of Flash you've specified to view the SWF, an HTML page with a link from which to download the latest version of Flash Player is displayed.

2 ACTIONSSCRIPT 2.0 ESSENCIAL (~6HRS)

2.1 INTRODUÇÃO AO ACTIONSSCRIPT 2.0

O QUE É O ACTIONSSCRIPT?

*Macromedia Flash Basic 8 and Macromedia Flash Professional 8 are the professional standard authoring tools for producing high-impact web experiences. **ActionScript is the language you use to add interactivity to Flash applications, whether your applications are simple animated SWF files or more complex rich Internet applications.** You don't have to use ActionScript to use Flash, but if you want to provide basic or complex user interactivity, work with objects other than those built into Flash (such as buttons and movie clips), or otherwise turn a SWF file into a more robust user experience, you'll probably want to use ActionScript.*

The main features of ActionScript 2.0 include the following:

Familiar object-oriented programming (OOP) model *The primary feature of ActionScript 2.0 is a familiar model for creating object-oriented programs. ActionScript 2.0 implements several object-oriented concepts and keywords such as class, interface, and packages that will be familiar to you if you've programmed with Java.*

Strict data typing *ActionScript 2.0 also lets you explicitly specify data types for variables, function parameters, and function return types. For example, the following code declares a variable named `userName` of type `String` (a built-in ActionScript data type, or class).*

```
var userName:String = "";
```

Compiler warnings and errors *The previous two features (OOP model and strict data typing) enable the authoring tool and compiler to provide compiler warnings and error messages that help you find bugs in your applications faster than was previously possible in Flash.*

ACTIONSSCRIPT E EVENTOS

*In Macromedia Flash Basic 8 and Macromedia Flash Professional 8, **ActionScript code is executed when an event occurs**: for example, when a movie clip is loaded, when a keyframe on the timeline is entered, or when the user clicks a button. **Events can be triggered either by the user or by the system.** Users click mouse buttons and press keys; the system triggers events when specific conditions are met or processes completed (the SWF file loads, the timeline reaches a certain frame, a graphic finishes downloading, and so on).*

When an event occurs, you write an **event handler** to respond to the event with an action. Understanding when and where events occur will help you to determine how and where you will respond to the event with an action, and which ActionScript tools to use in each case.

Events can be grouped into a number of categories:

Mouse and keyboard events (User Controlled Events), which occur when a user interacts with your Flash application through the mouse and keyboard;

Clip events (Automatic Events), which occur within movie clips; and frame events, which occur within frames on the timeline.



FIGURE 49 - MOUSE EVENTS

Mouse and keyboard events

A user interacting with your SWF file or application triggers mouse and keyboard events. For example, when the user rolls over a button, the `Button.onRollOver` or `on(rollOver)` event occurs; when the user clicks a button, the `Button.onRelease` event occurs; if a key on the keyboard is pressed, the `on(keyPress)` event occurs. You can write code on a frame or attach scripts to an instance to handle these events and add all the interactivity you desire.



FIGURE 50 – CLIPEVENTS

Clip events

Within a movie clip, you may react to a number of clip events that are triggered when the user enters or exits the scene or interacts with the scene by using the mouse or keyboard. You might, for example, load an external SWF file or JPG image into the movie clip when the user enters the scene, or allow the user's mouse movements to reposition elements in the scene.

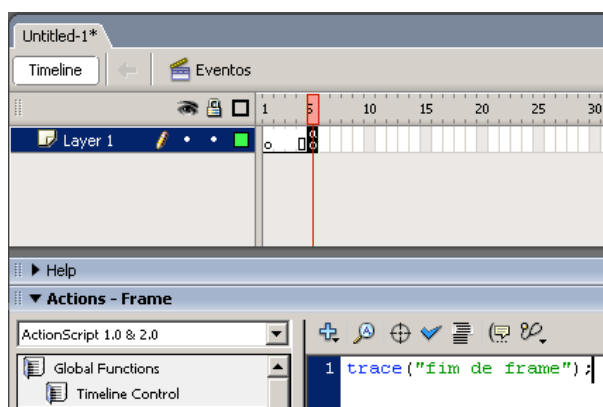


FIGURE 51 – EVENTOS AUTOMÁTICOS

Frame events

On a main or movie clip timeline, a system event occurs when the playhead enters a keyframe--this is known as a frame event. Frame events are useful for triggering actions based on the passage of time (moving through the timeline) or for interacting with elements that are currently visible on the Stage. When you add a script to a keyframe, it is executed when the keyframe is reached during playback. A script attached to a frame is called a frame script.

One of the most common uses of frame scripts is to stop the playback when a certain keyframe is reached. This is done with the `stop()` function. You select a keyframe and then add the `stop()` function as a script element in the Actions panel.



When you've stopped the SWF file at a certain keyframe, you need to take some action. You could, for example, use a frame script to dynamically update the value of a label, to manage the interaction of elements on the Stage, and so on.

PAINEL DE ACÇÕES (ACTIONS PANEL)

You use the Actions panel to create ActionScript in a Flash document (a FLA file). The Actions panel consists of three panes, each of which supports you in creating and managing scripts.

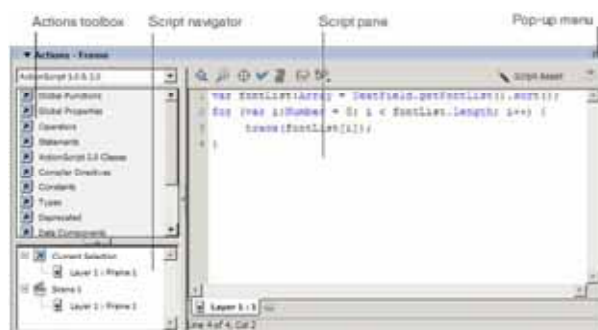


FIGURE 52 - PAINEL DE ACÇÕES

Actions toolbox Use the Actions toolbox to browse a categorical list of ActionScript language elements (functions, classes, types, and so on) and then insert them into the Script pane. You can insert a script element into the Script pane either by double-clicking or dragging it directly into the Script pane. You can also add language elements to your scripts by using the Add (+) button on the Actions panel toolbar. For more information, see *About the Actions panel and Script window toolbars*.

Script navigator The Script navigator displays a hierarchical list of Flash elements (movie clips, frames, and buttons) that contain scripts. Use the Script navigator to move quickly between all the scripts in your Flash document.

If you click an item in the Script navigator, the script associated with that item appears in the Script pane and the playhead moves to that position on the timeline. If you double-click an item in the Script navigator, the script gets pinned (locked in place).

Script pane *The Script pane is where you type your code. The Script pane provides you with tools to create scripts in a full-featured editor (called the ActionScript editor) that includes code syntax formatting and checking, code hinting, code coloring, debugging, and other features that simplify creating scripts.*

SYNTAXE E LINGUAGEM FUNDAMENTAL

Learning ActionScript syntax and statements is like learning how to put together words to make sentences, which you can then put together into paragraphs. ActionScript can be as simple. For example, in English, a period ends a sentence; in ActionScript, a semicolon ends a statement. In the ActionScript language, you can type a stop() action to stop the playhead of a movie clip instance or a SWF file from looping. Or you can write thousands of lines of code to power an interactive banking application. As you can see, ActionScript can do very simple or very complex things.

The ActionScript language is made up of the built-in classes that make up the ActionScript language. You need to use correct ActionScript syntax to form statements so the code compiles and runs correctly in Flash. In this case, syntax refers to the grammar and spelling of a language that you program with. The compiler cannot understand incorrect syntax, so you see errors or warnings displayed in the Output panel when you try to test the document in the test environment. Therefore, syntax is a collection of rules and guidelines that help you form correct ActionScript.

A statement is an instruction you give the FLA file to do something, such as to perform a particular action. For example, you can use a conditional statement to determine whether something is true or exists. Then you might execute actions that you specify, such as functions or expressions, based on whether the condition is true or not. The if statement is a conditional statement and evaluates a condition to determine the next action that should occur in your code.

```
// if statement
if (condition) {
    // statements;
}
```

Expressions, different from statements, are any legal combination of ActionScript symbols that represent a value. Expressions have values, while values and properties have types. An expression can consist of operators and operands, values, functions, and procedures.

For example, the following code is an expression:

$x + 2$

In the previous expression, x and 2 are operands and $+$ is an operator.

The way you format your ActionScript also determines how maintainable your code is. For example, it's difficult to read the logic of a FLA file that doesn't contain indents or comments, or contains inconsistent formatting and naming conventions. When you indent blocks of ActionScript (such as loops and if statements), the code is easier to read and debug if you encounter problems.

DOT SYNTAX E TARGET PATHS

In ActionScript, you use a dot (.) operator (dot syntax) to access properties or methods that belong to an object or instance on the Stage. You also use the dot operator to identify the target path to an instance (such as a movie clip), variable, function, or object.

A dot syntax expression begins with the name of the object or movie clip, followed by a dot, and it ends with the element you want to specify.

To control a movie clip, loaded SWF file, or button, you must specify a target path. Target paths are hierarchical addresses of movie clip instance names, variables, and objects in a SWF file. In order to specify a target path for a movie clip or button, you must assign an instance name to the movie clip or button. You name a movie clip instance by selecting the instance and typing the instance name in the Property inspector. Or you can specify the instance name with code if you create the instance using ActionScript. You can use the target path to assign an action to a movie clip or to get or set the value of a variable or property.

VISIBILIDADE DAS INSTÂNCIAS (SCOPE)

When you nest instances, the movie clip that nests a second movie clip is known as the parent to the nested instance. The nested instance is known as the child instance. The main Stage and main timeline are essentially a movie clip themselves, and can therefore be targeted as such.

You can target parent instances and parent timelines using ActionScript. When you want to target the current timeline, you use the `this` keyword. For example, when you target a movie clip called `myClip` that's on the current main timeline, you would use

```
this.myClip.
```

Optionally, you can drop the `this` keyword, and just use

```
myClip
```

```
trace("me: " + this);
```

```
this._parent.stop();
```

If you're familiar with Flash and ActionScript, you've probably noticed people using the `_root` scope. The `_root` scope generally refers to the main timeline of the current Flash document. You should avoid using the `_root` scope unless it's absolutely necessary. You can use relative target paths instead of `_root`. If you use `_root` in your code, you can encounter errors if you load the SWF file into another Flash document.

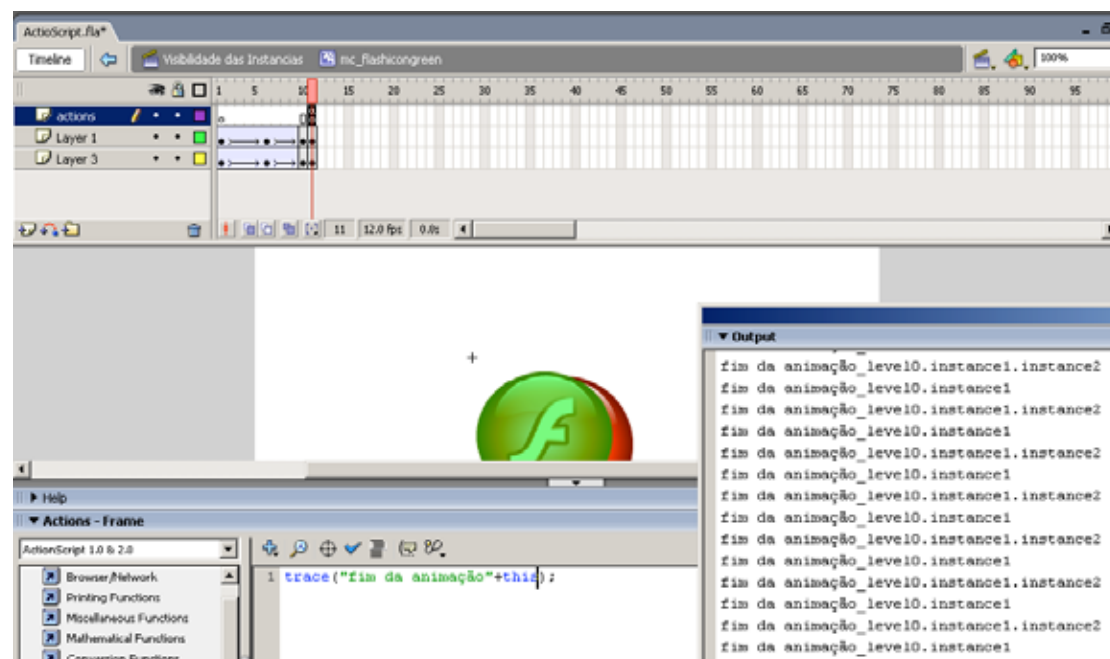


FIGURE 53 - SCOPE DAS INSTÂNCIAS

2.2 ADDING ACTIONSCRIPT

TIPOS DE DADOS E VARIÁVEIS

Data refers to the numbers, strings, and other information that you can manipulate within Flash. Using data is usually essential when you create applications or websites. You also use data when you create advanced graphics and script-generated animation, and you might have to manipulate values that you use to drive your effects.

A data type describes a piece of data and the kinds of operations that you can perform on it. You store data in a variable. You use data types when creating variables, object instances, and function definitions to assign the type of data you're working with. You use many different data types when you write ActionScript.

ActionScript has numerous basic data types that you will probably use frequently in your applications. ActionScript also has core classes, such as Array and Date, that are considered complex or reference data types.

In ActionScript 2.0, you can assign data types to variables when you declare them. You can convert one data type to another at runtime using one of the following conversion functions: Array(), Boolean(), Number(), Object(), String().

TIPOS DE DADOS SIMPLES (PRIMITIVOS) E COMPLEXOS

You can divide all the different data type values into two main categories: primitive or complex.

A primitive value (or primitive data type) is a value that ActionScript stores at the lowest level of abstraction, which means that operations on the primitive data types are generally faster and more efficient than operations carried out on complex data types. The following data types all define a set of one or more primitive values:

Boolean, null, Number, String, and undefined.

A complex value (or complex data type) is a value that is not a primitive value and that references the primitive values. Often, these are called reference data types. Complex values belong to the Object data type or a data type that is based on the Object data type. Data types that define sets of complex values include Array, Date, Error, Function, and XML. For more information on these complex data types, see their entries in the ActionScript 2.0 Language Reference.

Variables that contain primitive data types behave differently in certain situations than those containing complex types. For more information, see Using variables in a project.

ActionScript has the following basic data types that you can use in your applications:

<i>Data type</i>	<i>Description</i>
<i>Boolean</i>	<i>Primitive. The Boolean data type consists of two values: true and false. No other values are valid for variables of this type. The default value of Boolean variable that has been declared but not initialized is false. For more information, see Boolean data type.</i>
<i>MovieClip</i>	<i>Complex. The MovieClip data type lets you control movie clip symbols using the methods of the MovieClip class. For more information, see MovieClip data type.</i>
<i>null</i>	<i>Primitive. The null data type contains the value null. This value means no value—that is, a lack of data. You can assign the null value in a variety of situations to indicate that a property or variable does not have a value assigned to it. The null data type is the default data type for all classes that define complex data types. An exception to this rule is the Object class, which defaults to undefined. For more information, see null data type.</i>
<i>Number</i>	<i>Primitive. This data type can represent integers, unsigned integers, and floating point numbers. To store a floating point number, you should include a decimal point in the number. Without the decimal point, the number is stored as an integer. The Number data type can store values from Number.MAX_VALUE (very high) to Number.MIN_VALUE (very low). For more information, see ActionScript 2.0 Language Reference and Number data type.</i>
<i>Object</i>	<i>Complex. The Object data type is defined by the Object class. The Object class serves as the base class for all class definitions in ActionScript, and it lets you arrange objects inside each other (nested objects). For more information, see Object data type.</i>
<i>String</i>	<i>Primitive. The String data type represents a sequence of 16-bit characters that might include letters, numbers, and punctuation marks. Strings are stored as Unicode characters, using the UTF-16 format. An operation on a String value returns a new instance of the string. For more information, see String data type.</i>
<i>undefined</i>	<i>Primitive. The undefined data type contains one value: undefined. This is the default value for instances of the Object class. You can only assign a value of undefined to variables that belong to the Object class. For more information, see undefined data type.</i>
<i>Void</i>	<i>Complex. The Void data type contains only one value: void. You use this data type to designate functions that don't return a value. Void is a</i>

	<i>complex data type that references the primitive Void data type. For more information, see Void data type.</i>
--	--

VARIÁVEIS

A variable is a container that holds information.

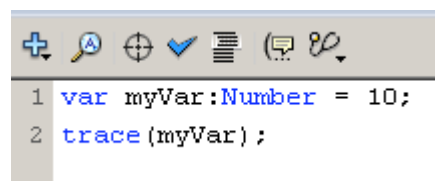
The following ActionScript shows what a variable looks like in ActionScript:

```
var myVariable:Number = 10;
```

This variable holds a numerical value. The use of :Number in the previous code assigns the type of value that variable holds, called data typing.

The container (represented by the variable name) is always the same throughout your ActionScript, but the contents (the value) can change. You can change the value of a variable in a script as many times as you want. When you change the value of a variable while the SWF file plays, you can record and save information about what the user has done, record values that change as the SWF file plays, or evaluate whether a condition is true or false. You might need the variable to continually update while the SWF file plays, such as when a player's score changes in a Flash game. Variables are essential when you create and handle user interaction in a SWF file.

It's a good idea to assign a value to a variable the first time you declare the variable. Assigning an initial value is called initializing the variable, and it's often done on Frame 1 of the Timeline or from within a class that loads when the SWF file begins to play. There are different kinds of variables, which are affected by scope. For more information on different kinds of variables and scope, see About variables and scope.



```
1 var myVar:Number = 10;
2 trace(myVar);
```

FIGURE 54 - VARIÁVEIS E TRACE()

To view the value of a variable, use the `trace()` statement to send the value to the Output panel. Then, the value displays in the Output panel when you test the SWF file in the test environment.

Para um uso correcto das variáveis deve ter-se em conta os seguintes procedimentos:

Declaring variables

Assigning values

Naming variables

Variables and scope

```
4 var myVar2:Number; //Declarar (definir tipo de) variável (dar nome e visibilidade)
5 myVar2 = 20; // Inicializar a variavel = Atribuir valores;
```

FIGURE 55 - DECLARAR, DEFINIR, NOMEAR E INICIALIZAR VARIÁVEIS

VISIBILIDADE DAS VARIÁVEIS (SCOPE)

Assim como a visibilidade das instâncias, *a variable's scope refers to the area in which the variable is known (defined) and can be referenced. The area in which the variable is known might be within a certain timeline or inside a function, or it might be globally known throughout the entire application.*

There are three types of variable scopes in ActionScript:

Global variables and functions are visible to every timeline and scope in your document.

Therefore, a global variable is defined in all areas of your code.

Timeline variables are available to any script on that timeline.

Local variables are available within the function body in which they are declared

(delineated by curly braces). Therefore, local variables are only defined in a part of your code.

Global variables

Global variables and functions are visible to every timeline and scope in your document. To declare (or create) a variable with global scope, use the `_global` identifier before the variable name and do not use the `var =` syntax. For example, the following code creates the global variable `myName`:

```
var _global.myName = "George"; // Incorrect syntax for global variable
_global.myName = "George"; // Correct syntax for global variable
```

However, if you initialize a local variable with the same name as a global variable, you don't have access to the global variable while you are in the scope of the local variable, as shown in the following example:

```
_global.counter = 100; // Declares global variable
trace(counter); // Accesses the global variable and displays 100
function count():Void {
    for (var counter:Number = 0; counter <= 2; counter++) { // Local
variable
        trace(counter); // Accesses local variable and displays 0 through 2
    }
}
count();
trace(counter); // Accesses global variable and displays 100
```

EXPRESSÕES, DECLARAÇÕES E OPERADORES (OPERACIONAIS, ESTRUTURAIS E TERMINAIS)

A statement is an instruction you give the FLA file to do something, such as to perform a particular action. For example, you can use a conditional statement to determine whether something is true or exists. Then your code might execute actions that you specify, such as functions or expressions, based on whether the condition is true or not.

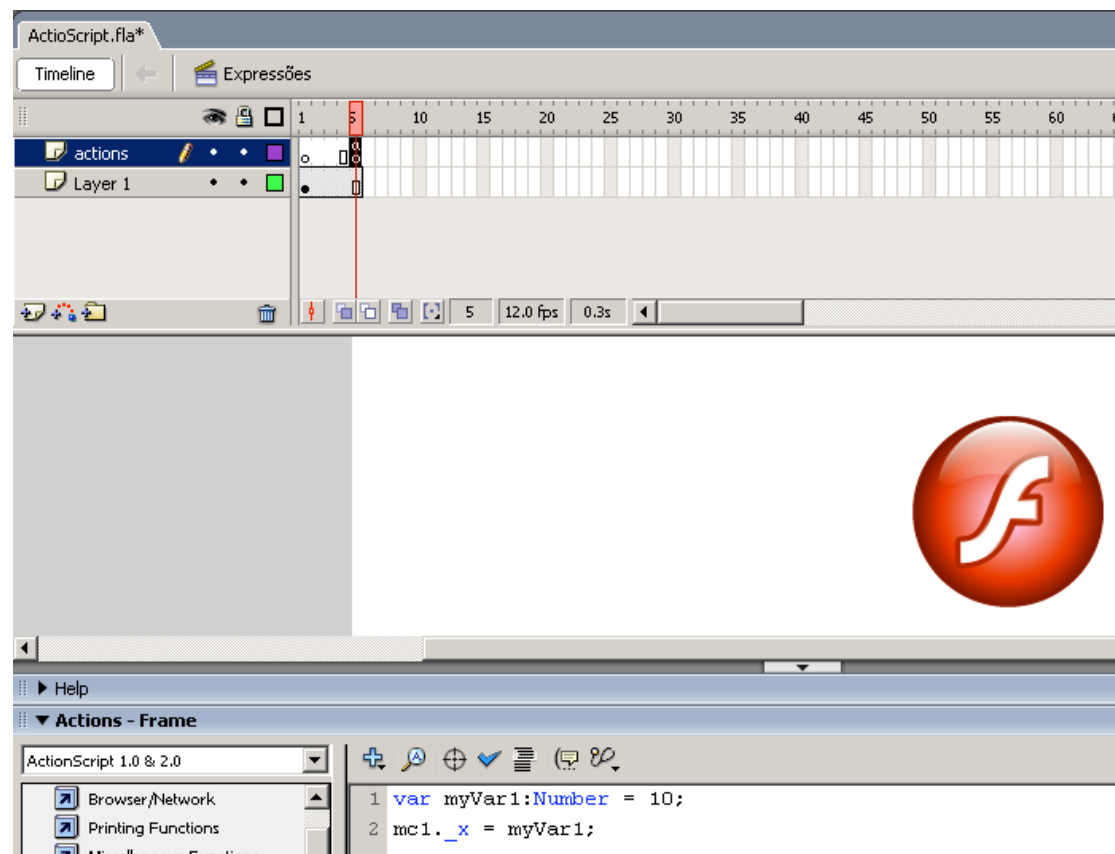


FIGURE 56 - EXPRESSÃO SIMPLES

For example, the if statement is a conditional statement and evaluates a condition to determine the next action that should occur in your code.

```
// if statement
if (condition) {
    // statements;
}
```

Another example is the return statement, which returns a result as a value of the function in which it executes.

There are many different ways for you to format or write ActionScript. Y

Assign variables as separate statements. Consider the following ActionScript example:

```
var myNum:Number = (a = b + c) + d;
```

This ActionScript embeds an assignment within the code, which is difficult to read. If you assign variables as separate statements, it improves readability, as the following example shows:

```
var a:Number = b + c;  
var myNum:Number = a + d;
```

OPERADORES

Operators are characters that specify how to combine, compare, or change values in an expression. An expression is any statement that Flash can evaluate and that returns a value. You can create an expression by combining operators and values or by calling a function.

*For example, a mathematical expression uses numerical operators to manipulate the values you use. Examples of operator characters are +, <, *, and =. An expression consists of operators and operands, and they are any legal combination of ActionScript symbols that represent a value. An operand is the part of your code that the operator performs actions on. For example, in the expression $x + 2$, x and 2 are operands and $+$ is an operator.*

Operadores Numéricos

You use numeric operators to add, subtract, divide, and multiply values in ActionScript. You can perform different kinds of arithmetic operations. One of the most common operators is the increment operator, commonly formed as $i++$. There are more things you can do with this operator.

You can add the increment before (preincrement) or after (postincrement) an operand.

```
// example one  
var firstScore:Number = 29;  
if (++firstScore >= 30) {  
    // should trace  
    trace("Success! ++firstScore is >= 30");  
}  
  
// example two  
var secondScore:Number = 29;  
if (secondScore++ >= 30) {  
    // shouldn't trace  
    trace("Success! secondScore++ is >= 30");  
}
```

Operadores Relacionais

The relational operators take two operands, compare their values, and return a Boolean value. All of the operators in this table have equal precedence:

<i>Operator</i>	<i>Operation performed</i>
<	<i>Less than</i>
>	<i>Greater than</i>
<=	<i>Less than or equal to</i>
>=	<i>Greater than or equal to</i>
instanceof	<i>Checks prototype chain</i>
in	<i>Checks for object properties</i>

Operadores de Equivalência

The equality operators take two operands, compare their values, and return a Boolean value. All of the operators in this table have equal precedence:

<i>Operator</i>	<i>Operation performed</i>
==	<i>Equality</i>
!=	<i>Inequality</i>
===	<i>Strict equality</i>
!==	<i>Strict inequality</i>

Operadores Lógicos

You use logical operators to compare Boolean values (true and false) and then return a Boolean value based on the comparison. For example, if you have two operands that evaluate to true, the logical AND (&&) operator returns true. Or if one or both of the operands evaluate to true, the logical OR (||) operator returns true.

The logical operators take two operands and return a Boolean result. The logical operators differ in precedence and are listed in the table in order of decreasing precedence:

<i>Operator</i>	<i>Operation performed</i>
&&	<i>Logical AND</i>
	<i>Logical OR</i>

Operadores Lógicos (Bitwise)

The bitwise logical operators take two operands and perform bit-level logical operations. The bitwise logical operators differ in precedence and are listed in the table in order of decreasing precedence:

<i>Operator</i>	<i>Operation performed</i>
&	<i>Bitwise AND</i>
^	<i>Bitwise XOR</i>
	<i>Bitwise OR</i>

CONDIÇÕES

You use conditions to determine whether something is true or exists, and then you can optionally repeat an action (using loops), or execute actions that you specify, such as functions or expressions, based on whether the condition is true or not. For example, you can determine whether a certain variable is defined or has a certain value and execute a block of code based on the result. Also, you could change the graphics within your Flash document based on what time the user's system clock is set to or on the weather in the user's current location.

To perform an action depending on whether a condition exists, or to repeat an action (create loop statements), you can use if, else, else if, for, while, do while, for..in, or switch statements.

ESTRUTURA IF

The if..else conditional statement lets you test a condition and then execute a block of code if that condition exists or execute an alternative block of code if the condition does not exist.

For example, the following code tests whether the value of x exceeds 20, generates a trace() statement if it does, or generates a different trace() statement if it does not:

```
if (x > 20) {
    trace("x is > 20");
} else {
    trace("x is <= 20");
}
```

If you do not want to execute an alternative block of code, you can use the if statement without the else statement.

The if..else statement in Flash is similar to the if statement. For example, if you use the if statement to validate that a user's supplied user name and password matches a value stored in a database, then you might want to redirect the user based on whether the user name and password are correct. If the login is valid, you can redirect the user to a welcome page using the if block. However, if the login was invalid, you can redirect the user to the login form and display an error message using the else block.

```
// create a string that holds AM/PM based on the time of day.
var amPm:String;
// no parameters pass to Date, so returns current date/time.
var current_date:Date = new Date();
// if current hour is greater than/equal to 12, sets amPm string to "PM".
if (current_date.getHours() >= 12) {
    amPm = "PM";
} else {
    amPm = "AM";
}
trace(amPm);
```

EXERCÍCIO IF

Desenhar um MovieClip animado cuja coordenada horizontal seja igual à do rato quando o cursor estiver para além da metade do ecrã

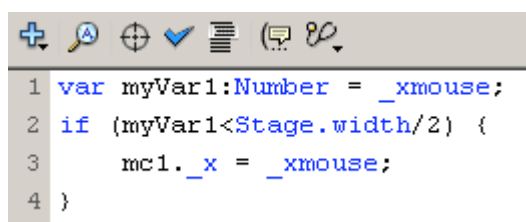


FIGURE 57 - EXERCÍCIO IF

```
var myVar1:Number = _xmouse;
if (myVar1<Stage.width/2) {
    mc1._x = _xmouse;
}
```

ESTRUTURA SWITCH

The switch statement creates a branching structure for ActionScript statements. Similar to the if statement, the switch statement tests a condition and executes statements if the condition returns a value of true.

When you use a switch statement, the break statement instructs Flash to skip the rest of the statements in that case block and jump to the first statement that follows the enclosing switch statement. If a case block doesn't contain a break statement, a condition called "fall through" occurs. In this situation, the following case statement also executes until a break statement is encountered or the switch statement ends. This behavior is demonstrated in the following example, where the first case statement doesn't contain a break statement and therefore both of the code blocks for the first two cases (A and B) execute.

All switch statements should include a default case. The default case should always be the last case on a switch statement and should also include a break statement to prevent a fall-through error if another case is added. For example, if the condition in the following example evaluates to A, both the statements for case A and B execute, because case A lacks a break statement. When a case falls through, it does not have a break statement, but includes a comment in the break statement's place, which you can see in the following example after case A. Use the following format when you write switch statements:

```
switch (condition) {  
  case A :  
    // statements  
    // falls through  
  case B :  
    // statements  
    break;  
  case Z :  
    // statements  
    break;  
  default :  
    // statements  
    break;  
}
```

EXERCÍCIO SWITCH

Criar uma filme que em cada vez que se carregue numa tecla o script avalia a tecla e determina se foi igual à tecla para a qual produz “Bingo” na janela de Output, caso contrário, “Tenta outra vez”

```
var listenerObj:Object = new Object();
listenerObj.onKeyDown = function() {
    // Use the String.fromCharCode() method to return a string.
    switch (String.fromCharCode(Key.getAscii())) {
        case "A" :
            trace("you pressed A");
            break;
        case "a" :
            trace("you pressed a");
            break;
        case "E" :
        case "e" :
            /* E doesn't have a break statement, so this block executes if you
            press e or E. */
            trace("you pressed E or e");
            break;
        case "I" :
        case "i" :
            trace("you pressed I or i");
            break;
        default :
            /* If the key pressed isn't caught by any of the above cases,
            execute the default case here. */
            trace("you pressed some other key");
    }
};
Key.addListener(listenerObj);
```

ITERAÇÕES (ESTRUTURA FOR)

*Repeating actions using loops. **ActionScript can repeat an action a specified number of times or while a specific condition exists. Loops let you repeat a series of statements when a particular condition is true.** There are four types of loops in ActionScript: for loops, for..in*

loops, while loops, and do..while loops. Each type of loop behaves somewhat differently, and each one is useful for different purposes.

Most loops use some kind of counter to control how many times the loop executes. Each execution of a loop is called an iteration. You can declare a variable and write a statement that increases or decreases the variable each time the loop executes. In the for action, the counter and the statement that increments the counter are part of the action.

<i>Loop</i>	<i>Description</i>
for loops	<i>Repeat an action using a built-in counter.</i>
for..in loops	<i>Iterate over the children of a movie clip or object.</i>
while loops	<i>Repeat an action while a condition exists.</i>
do..while loops	<i>Similar to while loops, except the expression evaluates at the bottom of the code block, so the loop always runs at least once.</i>

The most common type of loop is the for loop, which loops over a block of code a predefined number of times. For example, if you have an array of items, and you want to perform a series of statements on each item in the array, you would use a for loop and loop from 0 to the number of items in the array. Another type of loop is the for..in loop, which can be very useful when you want to loop over each name/value pair within an object and then perform some type of action. This can be very useful when you are debugging your Flash projects and want to display the values that load from external sources, such as web services or external text/XML files. The final two types of loops (while and do..while) are useful when you want to loop over a series of statements but you don't necessarily know how many times you need to loop. In this case you can use a while loop that loops as long as a certain condition is true.

ActionScript can repeat an action a specified number of times or while a specific condition exists. Use the while, do..while, for, and for..in actions to create loops. This section contains general information on these loops. See the following procedures for more information on each of these loops.

The for loop lets you iterate over a variable for a specific range of values. A for loop is useful when you know exactly how many times you need to repeat a series of ActionScript statements. This can be useful if you want to duplicate a movie clip on the Stage a certain number of times or to loop over an array and perform a task on each item in that array. A for loop repeats an action using a built-in counter. In a for statement, the counter and the statement that increments the counter are all part of the for statement. You write the for statement using the following basic format:

```
for (init; condition; update) {
```

```
// statements;
}
```

*You must supply three expressions to a for statement: a variable that is set to an initial value, a conditional statement that determines when the looping ends, and an expression that changes the value of the variable with each loop. For example, the following code loops five times. The value of the variable *i* starts at 0 and ends at 4, and the output are the numbers 0 through 4, each on its own line.*

```
var i:Number;
for (i = 0; i < 5; i++) {
    trace(i);
}
```

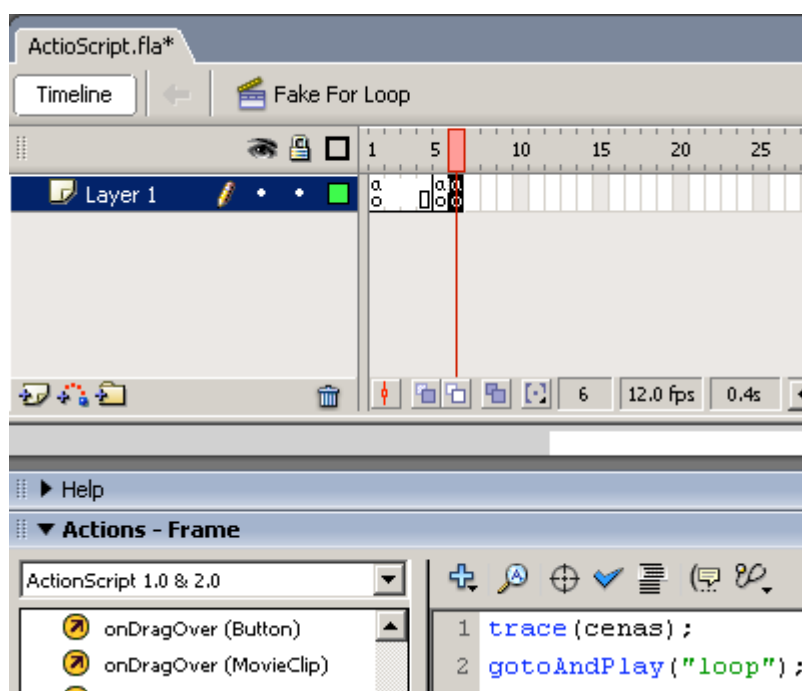


FIGURE 58 - LOOP FOR FALSO (TIMELINE)

FUNÇÕES E MÉTODOS

Understanding functions is important when you're writing ActionScript, creating classes, and using methods. There are several different kinds of functions that you'll work with.

Methods and functions are blocks of ActionScript code that you can reuse anywhere in a SWF file. You might write your functions in the FLA file or in an external ActionScript file and then

call the function from anywhere within your documents. Methods are merely functions that are located within an ActionScript class definition. You can define functions to execute a series of statements on passed values. Your functions can also return values. After a function is defined, it can be called from any timeline, including a timeline of a loaded SWF file.

If you pass values as parameters to a function, the function can perform calculations using the supplied values. Each function has individual characteristics, and some functions require that you pass certain types or numbers of values. If you pass more parameters than the function requires, the function ignores the extra values. If you don't pass a required parameter, the function assigns the undefined data type to the empty parameters. This can cause errors during runtime. A function can also return values (see Returning values from functions).

The basic syntax for a simple named function is:

```
function traceMe() {  
    trace("your message");  
}  
traceMe();
```

The basic syntax for a simple named function that builds on the previous example by passing a parameter, yourMessage, is:

```
function traceMe(yourMessage:String) {  
    trace(yourMessage);  
}  
traceMe("How you doing?");
```

Alternatively, if you want to pass multiple parameters, you could use the following code:

```
var yourName:String = "Ester";  
var yourAge:String = "65";  
var favSoftware:String = "Flash";  
function traceMe(favSoftware:String, yourName:String, yourAge:String) {  
    trace("I'm " + yourName + ", I like " + favSoftware + ", and I'm " +  
yourAge + ".");  
}  
traceMe(favSoftware,yourName,yourAge);
```

ABOUT BUILT-IN AND TOP-LEVEL FUNCTIONS

As discussed in About functions and methods, a function is a block of ActionScript code that can be reused anywhere in a SWF file. If you pass values as parameters to a function, the function operates on those values. A function can also return values.

You can use functions that are built into the ActionScript language. They might be top level, as described in About types of methods and functions; or the function might be in a built-in class, such as Math or MovieClip, which you use as a method in your application.

You use built-in functions in ActionScript to perform certain tasks and to access information. For example, you can get the number of milliseconds the SWF file has been playing by using `getTimer()`. Or you can get the version number of Flash Player that hosts the file by using `getVersion()`. Functions that belong to an object are called methods. Functions that don't belong to an object are called top-level functions and are found in subcategories of the Global Functions category of the Actions panel.

Some built-in functions require you to pass certain values. If you pass more parameters than the function requires, the extra values are ignored. If you don't pass a required parameter, the empty parameters are assigned the undefined data type, which can cause errors during runtime.

```
trace("my message");
```

When you test the SWF file, my message appears in the Output panel. Two other examples of top-level functions are `setInterval()` and `getTimer()`. The next example shows how to use both of these functions together. Add the following code to Frame 1 of the Timeline:

```
function myTimer():Void {  
    trace(getTimer());  
}  
  
var intervalID:Number = setInterval(myTimer, 100);
```

This code creates a simple timer using `getTimer()`, and uses the `setInterval()` and `trace()` top-level functions to display the number of milliseconds since the SWF file began to play in Flash Player.

WRITING NAMED FUNCTIONS

A named function is a kind of function that you commonly create in your ActionScript code to carry out all kinds of actions. When you create a SWF file, the named functions are compiled first, which means that you can reference the function anywhere in your code, as long as the function has been defined in the current or a previous frame. For example, if a function is defined in Frame 2 of a timeline, you cannot access that function in Frame 1 of the timeline.

The standard format for named functions is as follows:

```
function functionName(parameters) {  
    // function block  
}
```

This piece of code contains the following parts:

***functionName** is the unique name of the function. All function names in a document must be unique.*

***parameters** contains one or more parameters that you pass to the function. Parameters are sometimes called arguments. For more information on parameters, see *Passing parameters to a function*.*

***// function block** contains all of the ActionScript code that's carried out by the function. This part contains the statements that "do stuff." You can put the code that you want to execute here. The **// function block** comment is a placeholder for where your code for the function block would go.*

USING VARIABLES IN FUNCTIONS

Local variables are valuable tools for organizing code and making it easy to understand. When a function uses local variables, it can hide its variables from all other scripts in the SWF file; local variables are invoked in the scope of the body of the function and cease to exist when the function exits. Flash also treats any parameters passed to a function as local variables.

```
var myName:String = "Ester";  
var myAge:String = "65";  
var myFavSoftware:String = "Flash";  
function traceMe(yourFavSoftware:String, yourName:String, yourAge:String) {
```

```
        trace("I'm " + yourName + ", I like " + yourFavSoftware + ", and I'm "
+ yourAge + ".");
    }
    traceMe(myFavSoftware, myName, myAge);
```

PASSING PARAMETERS TO A FUNCTION

Parameters, also referred to as arguments, are the elements on which a function executes its code. (In this book, the terms parameter and argument are interchangeable.) You can pass parameters (values) to a function. You can then use these parameters for processing the function. You use the values within the function block (statements within the function).

Sometimes parameters are required, and sometimes they are optional. You might even have some required and some optional parameters in a single function. If you do not pass enough parameters to a function, Flash sets the missing parameter values to undefined, which may cause unexpected results in your SWF file.

The following function called myFunc() takes the parameter someText:

```
function myFunc(someText:String):Void {
    trace(someText);
}
```

After passing the parameter, you can pass a value to the function when you call the function. This value traces in the Output panel, as follows:

```
myFunc("This is what traces");
```

When you call the function, you should always pass the specified number of parameters unless your function checks for undefined values and sets default values accordingly. The function substitutes the passed values for the parameters in the function definition; if any parameters are missing, Flash sets their value to undefined. You regularly pass parameters into functions when you write ActionScript code.

To pass a variable number of parameters to a function:

```
function addNumbers(a:Number, b:Number, c:Number):Number {
    return (a + b + c);
}
```

```
trace(addNumbers(1, 4, 6)); // 11
trace(addNumbers(1, 4)); // NaN (Not a Number), c equals undefined
trace(addNumbers(1, 4, 6, 8)); // 11
```

If you don't pass enough parameters to the addNumbers function, the missing arguments are assigned a default value of undefined. If you pass too many parameters, the excess parameters are ignored.

Select Control > Test Movie to test the Flash document.

Flash displays the following values: 11, NaN, 11.

RETURNING VALUES FROM FUNCTIONS

You use the return statement to return values from functions. The return statement specifies the value that is returned by a function. The return statement returns the result of an evaluation as a value of the function in which an expression executes. The return statement returns its result immediately to the calling code.

```
function sqr(myNum:Number):Number {
    return myNum * myNum;
}
```

Some functions perform a series of tasks without returning a value. The next example returns the processed value. You are capturing that value in a variable, and then you can use that variable within your application.

```
function getArea(width:Number, height:Number):Number {
    return width * height;
}
```

The getArea() function takes two parameters, width and height.

Type the following code after the function:

```
var area:Number = getArea(10, 12);
trace(area); // 120
```

The getArea() function call assigns the values 10 and 12 to the width and height, respectively, and you save the return value in the area instance. Then you trace the values that you save in the area instance.

2.3 INTRODUÇÃO AOS OBJECTOS E VECTORES (ARRAYS)

VECTORES (ARRAYS)

An array is an object whose properties are identified by numbers representing their positions in the structure. Essentially, an array is a list of items. It's important to remember that each element in an array doesn't have to be the same data type. You can mix numbers, dates, strings, and objects and even add a nested array at each array index.

The following example is a simple array of month names.

```
var myArr:Array = new Array();  
myArr[0] = "January";  
myArr[1] = "February";  
myArr[2] = "March";  
myArr[3] = "April";
```

The previous array of month names can also be rewritten as follows:

```
var myArr:Array = new Array("January", "February", "March", "April");
```

Or, you can use shorthand syntax, as follows:

```
var myArr:Array = ["January", "February", "March", "April"];
```

An array is like a structure for data. An array is like an office building, where each floor contains a different piece of data (such as accounting on floor 3, and engineering on floor 5). As such, you can store different kinds of data in a single array, including other arrays. Each floor of this building can contain multiple kinds of content (executives and accounting might share floor 3).

An array contains elements, which are equivalent to each floor of the building. Each element has a numeric position (the index), which is how you refer to each element's position in the array. This is similar to how each floor in a building has a floor number. Each element can either hold a piece of data (which could be a number, string, Boolean value, or even an array or object) or be empty.

You can also control and modify the array itself. For example, you might want to move the engineering department to the basement of the building. Arrays let you move values around, and they let you change the size of the array (say, renovate the building and add more floors or remove floors). As such, you can add or remove elements and move values to different elements.

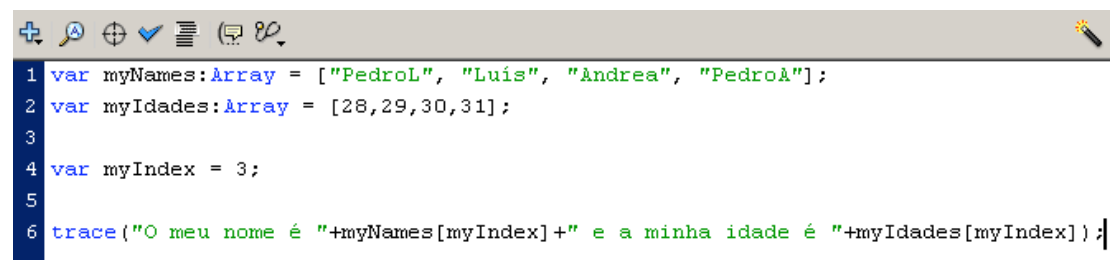
Therefore, the building (the array) contains floors (the elements), which are numbered floors (the index), and each floor contains one or more departments (the values).

The location of an item in an array is called the index. All arrays are zero-based, which means that the first element in the array is [0], the second element is [1], and so on.

```
// define a new array
var myArr:Array = new Array();
// define values at two indexes
myArr[1] = "value1";
myArr[0] = "value0";
// iterate over the items in the array
var i:String;
for (i in myArr) {
    // trace the key/value pairs
    trace("key: " + i + ", value: " + myArr[i]);
}
```

EXERCÍCIO ARRAY

Escrever as idades e os nomes das pessoas na sala e devolver o nome próprio e a idade própria.



```
1 var myNames:Array = ["PedroL", "Luís", "Andrea", "PedroA"];
2 var myIdades:Array = [28,29,30,31];
3
4 var myIndex = 3;
5
6 trace("O meu nome é "+myNames[myIndex]+" e a minha idade é "+myIdades[myIndex]);
```

FIGURE 59 - EXERCÍCIO ARRAY

```
var myNames:Array = ["PedroL", "Luís", "Andrea", "PedroA"];
var myIdades:Array = [28,29,30,31];
```

```
var myIndex = 3;
```

```
trace("O meu nome é "+myNames[myIndex]+" e a minha idade é  
"+myIdades[myIndex]);
```

2.4 MOVIE CLIPS E BOTÕES

MOVIE CLIPS

Movie clips are like self-contained SWF files that run independently of each other and the timeline that contains them. For example, if the main timeline has only one frame and a movie clip in that frame has ten frames, each frame in the movie clip plays when you play the main SWF file. A movie clip can, in turn, contain other movie clips, or nested clips. Movie clips nested in this way have a hierarchical relationship, where the parent clip contains one or more child clips.

You can name movie clip instances to uniquely identify them as objects that can be controlled with ActionScript. When you give a movie clip instance an instance name, the instance name identifies it as an object of the MovieClip class type. You use the properties and methods of the MovieClip class to control the appearance and behavior of movie clips at runtime.

You can think of movie clips as autonomous objects that can respond to events, send messages to other movie clip objects, maintain their state, and manage their child clips. In this way, movie clips provide the foundation of component-based architecture in Macromedia Flash Basic 8 and Macromedia Flash Professional 8. In fact, the components available in the Components panel (Window > Components) are sophisticated movie clips that are designed and programmed to look and behave in certain ways.

CONTROLAR MOVIE CLIPS COM ACTIONSCRIPT

You can use global ActionScript functions or the methods of the MovieClip class to perform tasks on movie clips. Some methods of the MovieClip class perform the same tasks as functions of the same name; other MovieClip methods, such as `hitTest()` and `swapDepths()`, don't have corresponding function names.

```
myMovieClip.play();  
parentClip.childClip.gotoAndPlay(3);
```

In the first statement, `play()` moves the playhead in the `myMovieClip` instance. In the second statement, `gotoAndPlay()` sends the playhead in `childClip` (which is a child of the instance `parentClip`) to Frame 3 and continues to move the playhead.

Global functions that control a timeline have a target parameter that let you specify the target path to the instance that you want to control. For example, in the following script `startDrag()` targets the instance the code is placed on and makes it draggable:

```
my_mc.onPress = function() {  
    startDrag(this);  
};  
my_mc.onRelease = function() {  
    stopDrag();  
};
```

The following functions target movie clips: `loadMovie()`, `unloadMovie()`, `loadVariables()`, `setProperty()`, `startDrag()`, `duplicateMovieClip()`, and `removeMovieClip()`. To use these functions, you must enter a target path for the function's target parameter to indicate the target of the function.

The following `MovieClip` methods can control movie clips or loaded levels and do not have equivalent functions: `MovieClip.attachMovie()`, `MovieClip.createEmptyMovieClip()`, `MovieClip.createTextField()`, `MovieClip.getBounds()`, `MovieClip.getBytesLoaded()`, `MovieClip.getBytesTotal()`, `MovieClip.getDepth()`, `MovieClip.getInstanceAtDepth()`, `MovieClip.getNextHighestDepth()`, `MovieClip.globalToLocal()`, `MovieClip.localToGlobal()`, `MovieClip.hitTest()`, `MovieClip.setMask()`, `MovieClip.swapDepths()`.

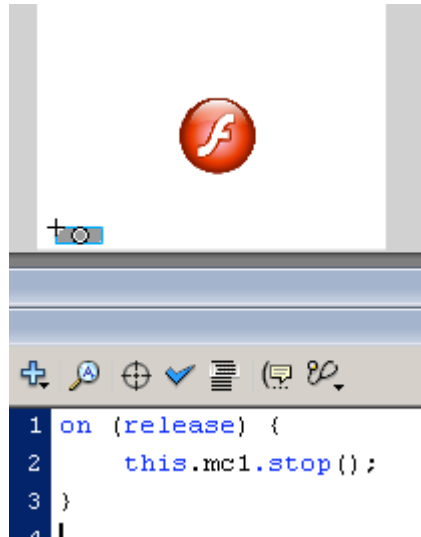


FIGURE 60 - TARGETING MOVIECLIPS

EXERCÍCIO

Criar um Scroll de um campo de texto (ver 1.16)

Parar um MovieClip animado (ver 1.16)

DESAFIO

Criar um Preloader (ver 1.16)

LOADING AND UNLOADING SWF FILES

To play additional SWF files without closing Flash Player, or to switch SWF files without loading another HTML page, you can use one of the following options:

The global loadMovie() function or loadMovie() method of the MovieClip class.

The loadClip() method of the MovieClipLoader class.

Use loadMovie() to do any of the following:

Play a sequence of banner ads that are SWF files by placing a loadMovie() function in a container SWF file that sequentially loads and unloads SWF banner files.

Develop a branching interface with links that lets the user select among several SWF files that are used to display a site's content.

Build a navigation interface with navigation controls in level 0 that loads content into other levels. Loading content into levels helps produce smoother transitions between pages of content than loading new HTML pages in a browser.

For more information on loading SWF files, see Loading external SWF and image files.

EXERCÍCIO

Criar um Slideshow (IDE) com um Preloader e carrega-lo dinamicamente.

DESAFIO

- Combinar os exercícios anteriores todos num simples ficheiro sob o tema autobiográfico:
- Menu de Popup – cada secção faz o loadMovie do movieClip para a área principal.
 - Secções: Início; Fotografias; Texto; ActionScript (especial).
 - Secção Início: MovieClip de Animação de Tweens.
 - Secção de Fotografia: Slideshow controlável;
 - Secção de Texto: Texto controlável em Scroll;
 - Secção de ActionScript em branco (a preencher na próxima sessão).
- Cada MovieClip relativo à Secção deverá ser gravado com o nome [omeuprimeironome]+home, [omeuprimeironome]+foto, [omeuprimeironome]+texto, [omeuprimeironome]+script.

2.5 WORKING WITH TEXT

Dynamic text fields

Loading text

2.6 CREATING BASIC USER INPUT FORMS

2.7 CONTROLLING SOUND

CREATING AND ATTACHING SOUND

LOADING SOUND

START, STOP AND PAUSE

3 DESAFIO PRÁTICO

Criar uma interface gráfica à semelhança do desenvolvido em www.big.dk




```
vinvertida = [this.mc_i, this.mc_h, this.mc_g,  
              this.mc_f, this.mc_e, this.mc_d,  
              this.mc_c, this.mc_b, this.mc_a];  
// Depois criamos uma lista Aleatoria  
// A ordem que atribuímos aos nomes dos moviclips em palco corresponde às posições  
nas listas x e y  
vrandomizada = [this.mc_f, this.mc_i, this.mc_d,  
                this.mc_e, this.mc_a, this.mc_h,  
                this.mc_g, this.mc_c, this.mc_b];  
  
//anim;  
  
// Organizar (atribuindo posições segundo as listas)  
this.btn_ordem.onRelease = function() {  
    trace("Ordenar");  
    vdestino = vordenada;  
    anim = true;  
};  
this.btn_invert.onRelease = function() {  
    trace("Inverter");  
    vdestino = vinvertida;  
    anim = true;  
};  
  
this.btn_random.onRelease = function() {  
    trace("Randomizar");  
    vdestino = vrandomizada;  
    anim = true;  
};  
  
//Correr um ciclo para modificar a posição dos objectos, horizontal e vertical  
//  
//trace("anim = "+anim);  
  
if (anim) {  
    for (i=0; i<vdestino.length; i++) {
```

```
        dx = vdestino[i]._x-vx[i];  
        dy = vdestino[i]._y-vy[i];  
        vdestino[i]._x = vx[i] + dx/3;  
        vdestino[i]._y = vy[i] + dy/3;  
  
        if (dx==0 && dy==0){  
            anim = false;  
        }  
    }  
}
```

BIBLIOGRAFIA E LINKS ÚTEIS

ADOBE Systems Incorporated (a) – *Flash Professional 8 Product Information* [On-line].

Disponível em: <URL:

<http://www.adobe.com/products/flash/flashpro/productinfo/faq/#item-1-1>>.

ADOBE Systems Incorporated (b) – *Bitmap vs. Vector-Based Graphics* [On-line].

Disponível em: <URL:

http://www.adobe.com/education/webtech/CS2/unit_graphics1/gb_print.htm>.

ADOBE Systems Incorporated (c) – [Vários Tutoriais On-line]. Disponível em: <URL:

<http://www.adobe.com/support/documentation/en/flash/>>.

BESLEY, Kristian; *et al* – *Foundation ActionScript for Flash 8*. Friends of Ed: Birmingham, 2006. ISBN-10 1590596188.

BESLEY, Kristian; BHANGAL, Sham – *Foundation Flash 8*. Friends of Ed: Birmingham, 2005. ISBN-10 1590595424.

CARDOSO, Jorge – *Sebenta de Programação Multimédia* [Em linha]. UCP: Porto, 2006. 16 Fev 2006 Disponível na WWW: <URL: <http://teaching.jotgecardoso.org/pm>>.

DESIGNWORKS.COM – *Vector graphics vs. Raster graphics: Pythagoras vs. Seurat*. [On-line]. Disponível em: <URL: http://www.design-works.com/resources/vector_and_raster_graphics.htm>.

FELLEISEN, Bruce; FINDLER, Robert; *et al* – *How to Design Programs: An Introduction to Programming and Computing* [Em linha]. MIT Press: Cambridge, MA, 2001 [Actual. em Set. 2003] Disponível na WWW: <URL: <http://www.htdp.org/>>. ISBN-10 0-262-06218-6, ISBN-13 978-0-262-06218-3.

LEURS, L. - *Bitmap versus vector graphics* [On-line]. Disponível em: <URL:

<http://www.prepressure.com/image/bitmapvector.htm>>.

MENDES, António José; MARCELINO, Maria José – *Fundamentos de Programação em Java 2*. FCA Editora: Lisboa, 200?. ISBN 972-722-423-7

NAGEL, David – *First Look: Macromedia Flash Professional 8 New video technologies lead beefed-up feature set* [On-line]. Disponível em: <URL: <http://mediadesigner.digitalmedianet.com/articles/viewarticle.jsp?id=33950-1>>.

NETO, Joao Pedro – *Programação, Algoritmos e Estruturas de Dados*. Escolar: Lisboa, 2004. ISBN 972-592-179-8