

# Conhecendo o ActionScript 3.0 para o desenvolvimento de jogos utilizando o Adobe Flash CS3.

**Filipe Ghesla Silvestrim**

AUGRS – Adobe User Group Rio Grande do Sul  
Porto Alegre – RS – Brasil

[contato@filipesilvestrim.com](mailto:contato@filipesilvestrim.com)

***Resumo.** Este documento serve como uma fundamentação teórica para o mini-curso, no pré-evento GDS, o qual abordará fatores importantes sobre a criação de jogos on-line e offline, utilizando a tecnologia Adobe Flash.*

*Tal mini-curso se aterá em introduzir o ActionScript 3.0 - a mais nova linguagem de programação para Flash - e abordar tópicos relacionados na criação de games.*

## 1. O Autor

Filipe Ghesla Silvestrim atua como instrutor de tecnologia Flash desde 2004, tendo atuado em centros de treinamentos e em clientes in company.

Possui certificação Adobe Flash 8 Professional Certified.

É Fundador e *Manager* do Grupo de Usuários oficial da Adobe no Rio Grande do Sul.

Cursando do curso de Jogos Digitais na UNISINOS.

Atua no desenvolvimento principalmente de objetos de aprendizagem virtual e jogos educativos (faz parte do núcleo desenvolvimento de um jogo educativo de química no Centro de Super Computação da UFRGS).

Possui um *blog* na área de tecnologia ActionScript, sendo a URL <http://www.filipesilvestrim.com>.

## 2. A Comunidade

O Adobe User Group Rio Grande do Sul – AUGRS - é um grupo de usuários destinado a obter uma maior integração e cooperação entre profissionais de Internet que utilizam produtos e tecnologias Adobe System – After Effects, Captivate, ColdFusion, Director, Dreamweaver, Flash, Flash Media Server, Flash Remoting, Flex, Illustrator, Photoshop, Premiere entre outras; sem deixar de lado tecnologias de apoio como PHP, MySQL, Javascript, CSS.

Este grupo – uma organização sem fins lucrativos - tem como foco e meta uma campanha intitulada ‘Incentivo aos Estudos’, na qual profissionais da área se mobilizam criando artigos – disponibilizados no *website* do grupo, tirando dúvidas – que são

sanadas em nossa lista de discussões - e ministrando palestras gratuitas – nos encontros do grupo. Constituindo, assim um ambiente integrado de aprendizagem.

O grupo não se prende a níveis de conhecimento de seus usuários; tendo uma amplitude muito grande entre nichos/áreas de atuação e níveis de conhecimento. Sendo participantes iniciantes na área de web, programadores avançados, gerentes de TI e curiosos – com o intuito de aprender novas tecnologias.

Mais informações em [www.augrs.com](http://www.augrs.com).

### **3. Descrição**

Dentre os principais objetivos, o mini-curso se propõe a apresentar o desenvolvimento de jogos utilizando o ActionScript 3.0 lançando mão do paradigma OOP.

A principal barreira para novos programadores ao começarem com a programação em Flash é o quesito de que toda a lógica de interatividade do Flash é baseada em eventos e esse ponto será um tópico de suma relevância na apresentação dessa linguagem de programação para um público de pessoas que queiram começar nessa tecnologia. Por tal motivo eventos em ActionScript 3.0 (AS3) será o tópico com segunda maior relevância no mini-curso.

Técnicas de otimização de jogos desenvolvidos com o Adobe Flash CS3 serão abordadas a fim de esboçar a interação entre interface e linguagem de programação.

Será apresentado e debatido - levando em conta as principais técnicas; classes e abordagens dos objetos visuais e sua interatividade com o código.

Serão apresentadas API's desenvolvidas em AS3, tais como motores de física e 3D.

### **4. Sumário do MiniCurso**

Este curso possui como objetivo principal o de introduzir o ActionScript 3.0 no desenvolvimento de jogos de forma que os participantes aprendam a sintaxe dessa linguagem através de um paradigma Orientado a Objetos.

O curso possui três principais módulos na abordagem do conteúdo.

O primeiro módulo atêm-se em explicar o que é o ActionScript, a sua evolução junto a tecnologia Flash, porquê o mesmo é indicado no desenvolvimento de games e que razões temos para utilizar a versão 3 dessa linguagem.

O segundo módulo foca-se em introduzir aos Desenvolvedores de Jogos essa linguagem de forma básica voltada para o desenvolvimento de games.

O terceiro módulo é destinado a uma abordagem mais aprofundada em técnicas de desenvolvimento de jogos.

### **5. Módulo 1 (o ActionScript)**

#### **5.1. Breve Histórico do Flash**

Em Dezembro de 1996 a Macromedia, com a aquisição da empresa FutureWave fez com que o *software* de animação vetorizada Future Splash Animator tornara-se Macromedia Flash 1.0.

Mas existe muito mais história antes do lançamento do Flash 1. Vamos voltar um pouco no tempo.

O Flash começou com o sonho de Jonathan Gay de se tornar um arquiteto. Passando por dificuldades para ver seus desenhos de casas em uma forma final Jonathan começa a programar para Apple II e logo descobre que escrevendo programas pode-se desenhar algo, construir e ver como funciona e como interage com o usuário. Enfim assim ele conseguia ver seus desenhos em uma forma final.

O primeiro programa que fora finalizado (escrito em Basic) fora uma cópia de Space Invaders. A partir de então ele começou a programar em Pascal e utilizou tal linguagem para escrever o seu primeiro editor gráfico o Super Paint.

Jonathan Gay começou a sua vida profissional ainda no segundo grau. Participando de algumas reuniões do Macintosh Users Group ele conhece o organizador do grupo Charlie Jackson que estava planejando criar uma empresa de desenvolvimento de *software* para Macintosh com o nome de Silicon Beach Software.

A partir de então Jonathan desenvolveu Airborne, o primeiro jogo para Macintosh game que utilizava sons digitais e animações com suavidade. Sendo assim por um tempo uma *kill application*.

Logo após vieram jogos como Dark Castle e Beyond Dark Castle. Desenvolver jogos tornou-se um dos elementos cruciais na sua aprendizagem como programador.

A partir de então ele começou a trabalhar no desenvolvimento do Super Paint II implementando desenhos em estilo PostScript. Após isso, Jonathan graduou-se na faculdade e começou a trabalhar tempo integral na Silicon Beach Software, desenvolvendo assim *softwares* gráficos desenvolvidos em C e utilizando uma *framework* com paradigma OOP.

Então em janeiro de 1993, Charlie Jackson, Jonathan Gay, and Michelle Welsh criaram uma pequena empresa de *software* chamada FutureWave Software e assim criaram o seu primeiro produto o SmartSketch. Uma aplicação de desenho que tinha como principal foco o de tornar a criação grafica em computadores tão simples quanto desenhar em um papel.

Com o início da internet o FutureWave era um potencial como uma ferramenta de animação baseada em desenho vetorial.

Em 1995 FutureWave modificou o SmartSketch adicionando animação *frame-by-frame* e relançando o produto como sendo agora um *software* de animação.

A partir de então foi-se evoluindo a tecnologia Flash com as novas versões do software. De versão em versão ele melhorava a parte gráfica, sonora e etc, mas fora em 1998 com o Flash 3 que a era dos games teria o seu ponto de partida com comandos de interação em *script* (ainda não dava para desenvolver jogos, mas estávamos no caminho).

E fora no ano de 2000 com o lançamento do Flash 5 e com a evolução dos *scripts* para Flash que fora lançada a linguagem de programação para Flash, o ActionScript 1.0. E é neste momento que começamos a ter os primeiros jogos em Flash. Com uma linguagem estruturada e prototipada (sendo possível simular classes e objetos). Essa versão da linguagem continha todos princípios básicos de linguagens web como o Lingo - Macromedia Director - e Java - Sun. Porém esta versão viera deficiente nos quesitos velocidade e poder.

Em setembro de 2003 com o Flash MX 2004 com o Flash Player 7 fora introduzido no mercado o ActionScript 2.0, dando assim aos desenvolvedores um paradigma formalmente OOP (Orientado a Objetos). Estando assim o ActionScript muito mais próximo ao ECMAScript (Javascript também é baseado nessa tecnologia).

E o momento mais glorioso da evolução do Flash até então aconteceu em 2006 o lançamento do ActionScript 3.0

A partir desse momento podemos dizer que jogos em Flash não são mais simplesmente “joguinhos” irmãos dos jogos feitos em Shockwave, pois tal era mais rápido, poderoso e ainda 3D. Porém agora com ActionScript 3.0 o Flash se torna tão poderoso quanto Shockwave. Num pensamento próprio, chego a dizer que atualmente mais poderoso que o Shockwave.

Agora nós possuímos um excelente ambiente de desenvolvimento de jogos 2D, tanto como jogos 3D com a biblioteca PaperVision 3D.

Atualmente existe o Flash Player 9 até mesmo para Linux. Antigas versões do player rodavam em Televisões Web, consoles como o Wii (Flash Player 7). Não muito distante está essa aproximação do Flash Player 9 para diversos dispositivos também.

## **5.2. Flash CS3 – Que estilos de jogos produzir?**

Ok, já sabemos que no Flash, utilizando o ActionScript, conseguimos produzir jogos, mas que estilos de jogos podemos produzir?

A resposta é, todo tipo de jogos, desde MMO RPG 2D até um FPS 3D, mas agora, será que é vantajoso produzir um MMO ou um jogo 3D?

A resposta é depende o tempo que se terá para produzir.

Sendo assim, abordarei aqui os principais estilos de jogos que estão ligados à tecnologia Flash e porque.

### **5.2.1. Jogos Casuais**

Jogos acessíveis e divertidos (estilo fácil de aprender e divertido de se tornar um *expert*), demograficamente neutros, adaptáveis ao fator tempo de jogo (podendo ser jogados desde 5 minutos a 5h) e cross plataforma.

São jogos geralmente definidos por serem jogos para *download* (com menos de 50mb) que possuem versões de teste (*trial* com limitação de tempo ou fases) e compra.

Para esse tipo de jogos serem desenvolvidos eles normalmente utilizam uma plataforma RAD (Desenvolvimento Rápido) e sendo assim o Flash se encaixa perfeitamente ao ponto que programadores e designers.

Normalmente porte de jogos tais como Zuma[1] são desenvolvidos de forma RAD em Flash e postos para vendas em portais de venda como Playfirst.com, Popcapgames.com, GameHouse.com.

Normalmente é um gênero que se assume sendo o Publisher o próprio Desenvolvedor. Sendo assim, este é o meio que a maioria dos novos desenvolvedores possuem para lançarem-se no mercado sem a necessidade de um Publisher.

**Dados:**

Aproximadamente 60 milhões de *downloads* de jogos são feitos por mês.

Em 2008 a American Online Casual Market estima uma meta de arrecadação de \$690 milhões.

Em todo o mundo estima-se que 150 milhões de pessoas jogam jogos gratuitos via internet.

A grande diferença entre jogos complexos e Casual games é a de que os primeiros possuem gráficos e recursos excelentes, porém são os segundos que irão comandar uma maior audiência.

### **5.2.2. Advergames**

Advergame é um estilo de jogos que engloba a idéia de jogos casuais voltados a uma forma de comunicação via *marketing*.

Tal estilo possui mensagens comerciais misturado ao conteúdo do jogo.

Essa interação virtual com uma marca associada no contexto de games diferencia esse tipo de jogos de um simples *banner* na internet.

Dados:

De \$56 milhões em 2005, a industria de advergaming espera gerar \$1.8 bilhões até 2010 [Massive Incorporated] - \$732 milhões [Yankee Group]

50% do público alvo que jogam Advergaming, jogam por um período em média de 25 minutos.

42% dos jogadores dizem que jogam uma ou mais horas de jogos online por semana [Entertainment Software Association].

## **5.3. Flash CS3 e AS3 para jogos**

### **5.3.1. Por que desenvolver jogos em Flash CS3?**

- Suporte Multiplataforma - PC, Mac, Linux;
- Suporte Multibrowser - IE, Firefox, Opera, Safari ...;
- Flash player 9 encontrase em 93.3% dos *browsers*;
- Jogos para *desktop* – AIR;
- Comunidade gigantesca de Desenvolvedores (gurus, forums, ugs ...);
- Uma *suite* poderosa e integrada;

- Developers e Designers Gráfico;
- Gráfico vetoriais + Renderização Bitmaps + Actionscript 3;
- Uma aplicação em Flash pode:
  - Se conectar com um servidor Remoto (LoadVars, XMLSocket);
  - Visualizar gráficos, sons e vídeos embedados no arquivo ou remotos;
  - Interagir com outra aplicação em flash (swf) (localConnection);
- Flash CS3 não força desenvolvedores e designers a trabalharem juntos.

### 5.3.2. Por que desenvolver jogos em Flash CS3 com Actionscript 3.0?

- É uma linguagem OOP e programação poderosa;
- É parecida com Java e C++;
- Desenvolvida para ser facilmente compreendida;
- Muito eficiente em todas as plataformas;
- ActionScript 3.0 é o resultado de anos de desenvolvimento.

### 5.3.3. AS3 – O que mudou? O que temos de novo?

*Atenção! Para quem provinha de programação em ActionScript 2 continue lendo esta sessão, caso contrário pule-a para a próxima.*

Primeiramente comece pensando como AS2 e AS3 sendo duas linguagens completamente diferentes.

Em relação ao AS2, o AS3 é uma linguagem mais tipada, mais complexa de se aprender, não são linguagens compatíveis.

Os código não podem mais ser colocado em instâncias de objetos no palco, mas ainda podem ser colocados na Timeline (mesmo que eu aconselhe a não fazer isso e sim trabalhar somente com classes).

AS3 fora totalmente repensado de uma forma a otimizar o desempenho de sua aplicação (testes evidenciam que o AS3 é 10 vezes mais rápido que o AS2), não tão somente tirando underscore das propriedades dos movieclips, ou transformando variações que antes ocorriam de 0 a 100 para 0 a 1, mas sim toda a arquitetura fora remodelada.

Na nova arquitetura fora incorporado o compilador JIT junto da nova máquina virtual AVM2 (ActionScript Virtual Machine).

MovieClip não é mais a resposta para todas as nossas perguntas.

Esqueça botões e comece a aprender sobre Display Objects: Sprites, Shapes, and Bitmaps, porém não assuste se se ao criar um objeto nada aparecer no palco (ele só está em memória).

Possuímos um modelo de eventos baseado no padrão DOM3.

XML agora ficou muito fácil com a estrutura de acesso através de E4X (ECMAScript para XML).

Binário? Sim, agora temos acesso a binário através da classe `ByteArray`.

Menos memória utilizada (32 bit) com os tipos de sistema `int` e `uint`.

Não se assuste se não achares mais o prefixo “on” em eventos, o prefixo simplesmente deixou de existir com o novo sistema de eventos.

Agora possuímos uma `DisplayList` - sendo a estrutura física do filme - na qual poderemos colocar nossos `Display Objects`.

#### 5.3.4. Razões para utilizar AS3

- AS3 é 10 vezes mais rápido que o AS2;
- Actionscript 3.0 proporciona um código estruturado;
- AS3 possui um curva de aprendizagem rápida;
- AS3 promove reusabilidade;
- AS3 pode ser desenvolvida e trocada de ambientes de desenvolvimento;
- Erros em tempo de execução - para tipagem - acesso impróprio a objetos – `try`, `catch` e `finally`;
- Muito mais fácil para criar novos `Display Objects` (`new MovieClip()`);
- Profundidade relativa a `Display Objects`;
- Padrões de Mercado – ECMAScript3 – E4X – RegExp – DOM3 – Tamarin engine (Mozilla);
- O documento (`swf`) pode possuir uma classe genérica como classe (não mais sendo um `MovieClip`);
- Tipos primitivos: `int`, `uint` e `void`;
- Acesso direto a Objetos através da classe `Dictionary`;
- Variáveis do AS3 podem ser utilizadas dentro do XML;
- Acesso de dados Binário e comunicação via *sockets* Binários;
- Podemos trocar o FPS em tempo de execução;
- Nova API de desenho.

#### 5.4. Designer Vs Developer

A plataforma de desenvolvimento Flash não obriga o trabalho conjunto de Designers e Developers, ao passo que os primeiros não necessitam conhecer métodos, tão pouco objetos e classes que os programadores desenvolveram, tais necessitam somente passar para o Developer os Objetos visuais na Biblioteca de um *swf* ou passar os arquivos como *bitmaps* externos.

##### ***Dicas para Designers:***

- Externalize os elementos audiovisuais sempre que for possível;
- Elementos que necessitam estar dentro do documento é uma ótima prática mantê-los sempre na Biblioteca;

- Agrupe os recursos em documentos separados por funcionalidade;
- Não torne tudo que está na Biblioteca em MovieClip, deixe os elementos na forma primitiva (Bitmap, Mp3, etc...).

#### ***Dicas para Desenvolvedores:***

- Faça Classes;
- Criem sempre uma classe base para o documento (fla);
- Não reinventem a roda, utilizem Classes já prontas.

## **6. Módulo 2 (conhecendo a linguagem em uma abordagem voltada à games)**

### **6.1. Começando com o ActionScript 3.0**

Para começar, escolha um ambiente de desenvolvimento que no nosso caso será o próprio Adobe Flash CS3, porém existem inúmeros ambientes tais como Flex Builder, FlashDevelop ou algum editor de texto a sua escolha.

Se você já possui experiências anteriores com o AS2. Pense como o AS2 e AS3 sendo linguagens completamente diferentes.

ActionScript 3 é Case Sensitivity, ou seja possui diferença entre maiúsculas e minúsculas, por exemplo uma variável Nome e outra nome são coisas completamente diferentes.

Nada mais de códigos na Timeline (no máximo um *frame* para todo *swf*).

A nova organização do ActionScript se dá de uma forma muito parecida a estrutura do Java. Para compilar o código de maneira mais otimizada lançou-se mão da reorganização do código em pacotes (*packages*) e é dessa mesma maneira que temos que começar a programar nossas classes, sempre dentro de *packages*.

**Atenção!** – nomes de Objetos (OOP) no palco passam a se chamar nomes de instâncias.

Funções, laços e condicionais são escritas com a mesma sintaxe de C++ ou Java.

### **6.2. Aonde colocar o código?**

NOTA: Alguns conteúdos serão explicados mais adiante.

A partir de agora com o AS3, somente pode-se colocar os códigos na Timeline, e não mais em instâncias.

**Atenção!** – nas propriedades de publicação do código lembre-se sempre que revisar para qual versão do ActionScript estaremos compilando o nosso arquivo.

### **6.3. Conceitos básicos**

#### ***Case sensitivity***

AS3 é uma linguagem *case sensitive*.

#### ***Fortemente tipada***

Tipar os dados no AS3 é uma opção fortemente recomendada. Ao passo que os dados não forem tipados, tais não estarão sujeitos a erros e os valores se adequarão a tipos de dados primitivos tais como Boolean, String, Number, int e uint.

Todos os valores em AS3, sendo eles primitivos ou complexos, são objetos. O Flash Player trata os tipos primitivos de uma forma especial, sendo que os mesmos se comportam como objetos, porém não requerem a normal sobrecarga associada a criação de objetos.

**Tabela 1. As duas linhas de código são equivalentes**

```
var numero:int = 3;  
var numero:int = new int(3);
```

**Tabela 2. Tipos de dados e os valores padrões.**

Tipo de dados	Valor padrão
String	null
Boolean	false
int	0
uint	0
Number	NaN
Outros tipos	null

NOTA: Propriedades criadas dinamicamente não podem ser tipadas.

### ***Operador ponto (.)***

Tal operador fornece um meio de acesso a métodos e propriedades de objetos.

**Tabela 3. Acessando propriedades e métodos**

```
var filipe:Pessoa = new Pessoa();  
filipe.nomeCompleto = "Filipe Silvestrim";  
filipe.caminhar();
```

A hierarquias das pastas dos pacotes é feita com este operador ao invés de barra(/).

## **6.4. Compilando ActionScript 3 com o Flash CS3**

Para compilar um programa em AS3 utilizando o Flash CS3, primeiramente necessitamos associar a classe principal (podemos pensar como o nosso arquivo Main em c++) com um documento do Flash (.fla).

**Tabela 4. Passo a passo para compilar um programa**

1. No Flash, selecione File → New.
2. Na janela New Document que se abrirá, selecione Flash File (ActionScript 3.0) e clique em OK.
3. Selecione File → Save As.
4. Salve o arquivo .fla com o nome desejado.
5. No painel de propriedades do Flash associe o nome da classe, com o caminho de pacote se o mesmo existir, ao campo Document Class.
6. Para compilar o programa selecione Control → Test Movie ou Ctrl + Enter.
7. Agora ao compilar teremos gerado um arquivo .swf na mesma pasta do .fla e com o mesmo nome.

## 6.5. Entendendo o Display List e o Display Objects

A Display List é a hierarquia de todos os objetos gráficos visualizados em tempo de execução pelo Flash. Em outras palavras, são as “coisas” que são visíveis no palco (MovieClips, Shapes, TextFields, etc..).

Podemos entender a Display List como sendo uma árvore hierárquica na qual possuímos *containers* que armazenam objetos e os mesmo podem armazenar mais *containers*, os quais são chamados de Display Objects, de forma recursiva.

Hierarquicamente a raiz, sendo derivada de um Display Object, desta árvore é uma única instância da classe Stage. Sendo assim, o primeiro filho da instância de Stage do arquivo sempre será uma instância da Document Class do documento (.swf).

A partir de então todos os objetos filhos, não apenas criados, mas também adicionados a instância da Document Class aparecerá na “tela”.

A classe DisplayObject é a classe base entre as classes de *display* e a partir de tal todas as outras de *display* são derivadas.

A principal mudança é a classe Sprite, sendo a classe base de recursos visuais interativos, esta atua como sendo somente estática, muito similar a classe Shape, porém pode conter filhos. Derivando da classe Sprite possuímos, dentre outras, a classe MovieClip sendo adicionado a mesma controle da linha de tempo (*timeline*) para animações.

Tendo em vista o quesito interatividade, todas as Document Class necessitam derivar da classe Sprite.

Para tornar um Display Object visível é necessário adicionar o mesmo a uma instância da classe DisplayObjectContainer (sendo Sprite e Stage derivados de tal classe), caso contrário os objetos nunca serão desenhados no documento.

**Dicas:**

- Utilize os métodos `addChild()` e `addChildAt()` de instâncias da classe `DisplayObjectContainer` para adicionar um `Display Object` a `Display List`, fazendo com que a instância apareça na tela.
- Utilize os métodos `removeChild()` e `removeChildAt()` de instâncias da classe `DisplayObjectContainer` para remover um `Display Object` a `Display List`, fazendo com que a instância seja removida visualmente. Porém perceba que o objeto continuará em memória.

## 6.6. Entendendo o Sistema de Eventos

Manipulação de eventos em AS3 é simplesmente a principal ferramenta de trabalho para o desenvolvimento de aplicativos interativos, sendo assim, games.

No AS3 a classe `EventDispatcher`, que é quem controla toda a parte de manipulação de eventos, agora fora incorporada ao núcleo da linguagem.

Quando o AS3 dispara um evento tendo como alvo um objeto que não faz parte da `Display List` o único objeto que irá receber notificação de tal evento é o próprio alvo; porém, quando possuímos como alvo um objeto que faz parte da `Display List`, tanto como o objeto alvo, quanto todos os objetos que vem antes deste na hierarquia são notificados que o evento ocorrera (para bloquear essa notificação para os outros objetos da hierarquia utilizar o método `stopImmediatePropagation()`).

Na nova arquitetura agora temos a possibilidade de que todo `Display Object` registre ouvintes de eventos que possuam como objetos alvos `Display Objects` na hierarquia descendente da `Display List`. Sendo assim, não fazemos com que todos os objetos na `Display List` escutem a notificação de tal evento.

Conceito básico de implementação:

Primeiramente criamos funções que possuirão o bloco de código que serão executados quando for escutada a notificação de um evento relacionado. Após feito isso necessitamos associar a função criada com os eventos utilizando o método `addEventListener()`, sendo chamado pelo objeto que sofrerá a notificação do evento.

Eventos de escuta de mouse e teclado são as principais formas de interatividade com games.

Para fazer o *game loop* utilizamos (criamos) eventos de `Timer` (disparado de acordo com determinados intervalos de tempo) e de `EnterFrame` (esse tipo de evento é disparado a cada passada de *frame* determinado pelo *frame rate* - *fps* - do flash, sendo assim temos que observar que se possuímos somente 1 *frame* em nosso documento, o `EnterFrame` será disparado a cada variação de *fps*, ou seja, se temos o nosso filme a 30 *fps*, 30 *frames* serão renderizados em 1 segundo e 1 *frame* será executado em 0,0333... segundos).

Ao lançar mão de eventos no AS3 observamos que o código de disparo dos eventos não estão referenciados de forma explícita. Isso ocorre pelo fato de que no AS3 todas as classes derivam de `EventDispatcher`, para que assim tenhamos avesso a eventos de disparo e escuta de eventos.

***Métodos da classe `EventDispatcher`:***

- `addEventListener()`: Passamos por parâmetro o evento que iremos manipular e a função que irá ser executada quando o evento ocorrer.
- `removeEventListener()`: Remove a manipulação de eventos adicionados a `listeners` utilizando o `addEventListener`.
- `dispatchEvent()`: Envia o evento relacionado a todos os *listeners* relacionados, podendo-se assim criar eventos customizados.
- `hasEventListener()`: Determina quando um objeto possui ou não *listeners* relacionados a certo evento.
- `willTrigger()`: Muito parecido com o `hasEventListener`, porém tal método checa se o atual objeto tão como todos outros objetos que poderiam ser afetados pela propagação do evento relacionado.

Quando estamos trabalhando com a propagação de eventos, isto ocorre em três fases:

- **Capturing phase**: Representa a fase da propagação do evento originada pelo pai do objeto que originara o evento, ou seja, todos eventos começam pelo Stage e são propagados de ordem descendendo na hierarquia dos Display Objects na Display List até chegar ao alvo.
- **Target phase**: É a fase na qual o evento chega até o alvo ou ao objeto pelo qual o evento fora originado.
- **Bubbling phase**: É a fase inversa a *Capturing phase*, voltando na hierarquia de Display Objects até chegar ao Stage.

## 6.7. Estratégias e técnicas de programação em AS3

Não otimize prematuramente ou obsessivamente o código antes do mesmo estar pronto. Codifique de acordo com os padrões e faça uso excessivo de OOP e Design Patterns.

Sempre tipe o código, sendo ele um objeto de classes primitivas ou próprias.

Utilize variáveis locais, minimizando assim o tempo de acesso durante a execução do aplicativo.

Os tipos `int` e `uint` são seus amigos e tornam a execução do código menor e armazenam menor peso em memória.

Utilize menos `Try/Catch` e mais condicionais.

Faça teste de unidade em suas classes.

Comente o código e coloque *tags* como `TODO` e `FIXME`.

Desenvolva o seu jogo totalmente em Classes com uma classe principal como sendo a Class Document do `.fla` ou utilize no máximo 1 *frame* dentro do `.fla`.

Caso você não saiba como começar a programar o seu jogo como um todo, divida o jogo em vários pedaços e comece a programar as partes de forma separada (preferencialmente em classes).

Faça sempre protótipos antes de começar a programar

Utilize sempre uma boa IDE, aqui utilizamos o Flash CS3, porém temos ainda o FDT3, Moxie e Flex SDK para Eclipse.

Utilize o tipo Number quando for necessário fazer cálculos matemáticos, o tipo int quando estiver trabalhando com índices de array (não utilize uint, pois isso falha no indexOf()).

O acesso a objetos agora é feito de uma forma quase que instantânea ao passo que utilizarmos a classe Dictionary, pois tal aponta direto o objeto e não faz um laço de procura.

Utilize um software de controle de versões tal como o Subversion (<http://subversion.tigris.org>).

Minimize o número de objetos no palco.

Se quiser deixar um objeto invisível, não coloque o alpha em 0 ou torne-o invisível e sim remova-o da Display List.

Remova os *listeners* ou os objetos de referência após ter acabado o que eles deveriam ter feito.

Antes de “fechar” o game para entrega remova todos os trace() de dentro do código.

Não utilize Display Objects quando for armazenar dados que não sejam visuais.

Se estivermos trabalhando com muitos cálculos, considere que multiplicação trabalha mais rápido que divisão (  $x/2$  se torna  $x*0.5$ ).

Fazer *cast* em um dado com o tipo int trabalha mais rápido do que utilizar Math.floor().

## 6.8. Animação

Para realizar animações, é necessário que haja uma variação nas propriedades de um elemento visual de acordo com a passagem do tempo. Necessitando-se assim um estado inicial e um estado final.

Dessa maneira necessitaremos modificar propriedades de instâncias visuais de acordo com a passagem do tempo, isso via programação ocorrerá de duas maneiras ou com eventos de EnterFrame ou Timer.

## 6.9. Física

Ao passo que necessitamos um maior realismo em nossas aplicações, lançamos mão de simulações de física que são aplicações de forças (valores) em objetos.

Ao trabalhar com física acabamos entrando no mérito de vetores, sendo estes grandezas aplicadas a propriedades de objetos.

Vetores podem ser entendidos por uma abstração de uma linha que provém de um ponto inicial até um ponto final, indicando uma direção, sentido e tamanho que significa a grandeza (o valor aplicado).

Definimos assim um comportamento e a partir de tal são geradas as animações.

### 6.9.1. Bibliotecas

Para AS3 existem duas bibliotecas de física mais conhecidas:

- APE (Actionscript Physics Engine) - <http://www.cove.org/ape>
- The Fisix Engine - <http://www.fisixengine.com>

## 6.10. Detecção de Colisão

Para o desenvolvimento de jogos, necessitamos crucialmente detectar colisões entre objetos visuais para que desse modo possamos fazer com que quando for detectada a colisão alguma reação ocorra.

Para que isso seja possível existem diversas técnicas e algumas delas estão listadas abaixo.

### 6.10.1. Objeto

Detecção de colisão entre duas instâncias de DisplayObject, avaliando assim se os objetos se interseccionam ou se sobrepõem.

Para essa técnica utilizamos o método hitTestObject() que testa a colisão entre a *bouding box* dos objetos.

### 6.10.2. Ponto

Detecção de colisão entre uma instância de DisplayObject e um ponto (posição x e y especificadas) especificado.

Para tal detecção a instância de DisplayObject possui o método hitTestPoint(), sendo que os dois primeiro parâmetros são as posições do ponto em x e y, respectivamente, e como terceiro parâmetro passamos um valor booleano indicando se desejamos trabalhar com detecção via *bouding box* (false) ou pixel-a-pixel (true).

### 6.10.3. Distância

Quando estamos trabalhando com objetos visuais circulares, ou queremos fazer o teste de colisão por *bouding spheres*, testamos via a formula de Pitágoras se a distância entre dos dois objetos é menor que a soma dos seus raios; caso a resposta for positiva os mesmos estão em colisão.

### 6.10.4. Objetos sem Bouding Box

Para testar colisão entre objetos sem utilizar *bouding box* ou que não sejam circulares utilizamos uma classe que testa colisão pixel-a-pixel utilizando métodos e propriedades da classe do Flash BitmapData (classe para manipulação de imagens a nível de pixel).

A classe inicialmente fora criada em AS2 por Grant Skinner ([http://www.gskinner.com/blog/archives/2005/08/flash\\_8\\_shape\\_b.html](http://www.gskinner.com/blog/archives/2005/08/flash_8_shape_b.html)) e portada para AS3 por Frederik Humblet (<http://labs.boulevard.be>).

## 6.11. XML

Com a nova estrutura de manipulação de XML baseada em E4X fornece uma maneira simples de acesso aos conteúdos utilizando a sintaxe do ponto (.). Podemos também acessar um elemento diretamente referenciando elementos pelo nome. O XML pode ser escrito *inline* no AS3.

As classes principais para manipulação de XML utilizando E4X são as classes XML (contendo o nodo pai do xml o qual contém seus nodos filhos) e XMLList (contém uma coleção ou uma lista de instâncias ou nodos contidos no nodo pai do xml).

Para o desenvolvimento de jogos, os quais lançam mão de customização externa, como a criação de um *tile map*, utiliza-se xml também com intuito de estruturar a ordem de animação de *cutscenes* e para tornar mais fácil a modificação e o porte do jogo para outras línguas.

## **7. Módulo 3 (técnicas no desenvolvimento de jogos)**

### **7.1. Partículas**

Para uma maior imersão utiliza-se a replicação de objetos visuais os quais possuem comportamentos físicos. Em games obtemos efeitos orgânicos que implicam em um jogo mais imersivo, para isso utiliza-se muito efeitos de partículas de fumaça, fogo, vapor, chuva, etc...

Sistemas de partículas podem variar de dúzias a centenas de partículas com propriedades e comportamentos específicos.

Antigamente efeitos de partículas não podiam passar de dúzias, mas agora com o advento do AS3 possuímos muito mais performas, sendo assim, podemos elevar o nível de realismo em jogos feitos em flash.

Para realizar tal sistema no AS3 lançamos mão da Display List e de tipos específicos de objetos DisplayObject associados a objetos do tipo BitmapData para uma manipulação a nível de píxel, permitindo com que apliquemos filtros a nível da matriz de projeção RGBA de uma imagem através da classe ColorMatrixFilter.

### **7.2. Double Buffer**

A fim diminuir o tempo de acesso a memória para fazer a projeção de elementos visuais, utiliza-se a técnica conhecida como Double Buffer a qual permite que carreguemos a próxima imagem, que irá ser mostrada visualmente, em memória enquanto a imagem atual está sendo mostrada.

No AS3 isso acontece quando criamos uma instância de BitmapData, mas não a adicionamos a Display List enquanto a instância atual de um objeto que esteja na Display List seja mostrada. Carregando a imagem inicialmente em memória, sem ser adicionada a Display List reduz bastante o tempo de criação e acesso de memória.

Utiliza-se os métodos lock() e unlock() da classe BitmapData para bloquear ou desbloquear uma imagem para que qualquer objeto que referencie a instância de BitmapData que possui essa imagem não atualize ou atualize, respectivamente, as mudanças ocorridas na imagem associada a instância de BitmapData.

### **7.3. API's para games**

- corelib

*Descrição:* Classes para Criptografia - MD5 Hash e SHA1 Hash, Manipulação de Imagens - Encodificações JPG e PNG, Serealização - JSON,

Universal Resource Identifiers, Serviço Remoto, Classes de Utilidades e muitas outras.

*URL:* <http://code.google.com/p/as3corelib/>

- AS3 Data Structures For Game Developers

*Descrição:* API de estrutura de dados para Game Developers.

*URL:* <http://lab.polygonal.de/ds/>

- mecheye-as3-libraries

*Descrição:* Um conjunto de bibliotecas para o desenvolvimento de jogos.

*URL:* <http://code.google.com/p/mecheye-as3-libraries/>

- as3soundeditorlib

*Descrição:* Biblioteca para edição de som.

*URL:* <http://code.google.com/p/as3soundeditorlib/>

- asinmotion

*Descrição:* Biblioteca de animação.

*URL:* <http://code.google.com/p/asinmotion/>

- Tweeners

*Descrição:* Biblioteca de animação.

*URL:* <http://code.google.com/p/tweener/>

- FZip

*Descrição:* Biblioteca que carrega arquivos ZIP, extrai e descompacta os arquivos internos.

*URL:* <http://codeazur.com.br/lab/fzip/>

- Paradox (Em fase final de criação)

*Descrição:* Game Engine para FPS 3D.

*URL:* <http://animasinteractive.com/propaganda/2007/10/08/paradox-the-flash-based-first-person-3d-engine-faq/>

- Zen Bullets

*Descrição:* Game Engine para jogos isométricos 2D.

*URL:* <http://www.zenbullets.com/isometric/>

- WiiFlash

*Descrição:* Biblioteca para integrar o Wiimote com Flash.

*URL:* <http://wiiflash.bytearray.org/>

- FWiiDom

*Descrição:* Biblioteca para integrar o Wiimote com Flash.

*URL:* <http://www.fwiidom.org/>

#### **7.4. 3D**

O Flash por si só não possui o eixo Z, porém agora com a nova arquitetura do AS3, é possível uma simulação do eixo Z com muita eficiência de código.

No atual momento existem três principais bibliotecas:

- PaperVision 3D - <http://blog.papervision3d.org/>
- Away 3D - <http://www.away3d.com/>
- Sandy - <http://www.flashesandy.org/>

## **8. Presente e Futuro**

### **8.1. AIR**

Ele é um runtime multi-plataforma que permite que desenvolvedores criem aplicações RIA (Rich Internet Application) em Flash, Flex, HTML, JavaScript, Ajax.

Pode-se desenvolver tanto aplicações *client-side* (podendo acessar diretivas do sistema) quanto *server-side*.

### **8.2. AIF Toolkit**

No MAX (congresso anual de tecnologias Adobe) de 2007, o pessoal da Adobe anunciou que o time da linguagem de processamento gráfico AIF, que possui o codinome “Hydra”, fará parte do desenvolvimento do Flash Player 10.

OK, mas o que isso indica?

Indica que o pessoal do Flash agora poderá usufruir de processamento de imagem feitos em GPU (placas de vídeo), fazendo com que possamos ter otimizações de áudio e vídeo em tempo real, mais poder de processamento de imagens em nossos *scripts* (essa tecnologia já é utilizada no AfterEffects CS3) e arquivos mais leve.

Alguns dos benefícios dessa linguagem incluem:

- Sintaxe familiar, baseada em GLSL, a qual baseia-se em C;
- Permite que o mesmo filtro rode tanto em GPU, quanto em CPU;
- Abstrai a complexidade de se programar direto em Hardware de forma Heterogênea;
- Podemos criar os nossos próprios BitmapFilters;
- Qualidade Adobe de processamento de Imagens.

### 8.3. Flash Player 10, codinome Astro

Um avançado e poderoso cliente *runtime* o qual tem como novidades em sua próxima versão: avançada renderização e processamento em *layout* de textos (linguagens bidirecional, *scripts* complexos; multi colunas; quebras de linhas; tabelas), efeitos em 3D (até que enfim o eixo Z), e filtros, *blend modes* e efeitos customizáveis (criados a partir da linguagem Hydra).

### 8.3. Flash 10

A próxima geração do Flash, Flash 10, codinome “Diesel”, terá um novo modelo de animação por *timeline* na qual não necessitaremos mais utilizar keyframes - necessitamos somente mexer o objeto, especificando o início e fim da animação, e então poderemos modificar a linha de animação e modificar as transições de um ponto ao outro através de uma curva de *bezier*.

Outra grande novidade é a adição de *bones* (ossos) as animações bem como ferramentas de edição 3D fazem e após tal passo podemos manipular os *bones* por AS3.

## 9. Referências

Peters, Keith (2007) Foundation ActionScript 3.0 Animation.

Lott, Joey, Schall, Darron e Peters Keith (2006) ActionScript 3.0 Cookbook.

Skinner, Grant (2007) “50 Reasons AS3 Kicks Ass”, <http://gskinner.com/talks/50reasonsAS3/>, Outubro.

International Games Developers Association (2007), Casual Games, <http://www.igda.org/casual>, Outubro

Adobe Labs (2007), AIF, AIR, Astro, [http://labs.adobe.com/wiki/index.php/Main\\_Page](http://labs.adobe.com/wiki/index.php/Main_Page), Novembro.

Adobe Developer Connection (2007), ActionScript Technology Center, <http://www.adobe.com/devnet/actionscript/>, Outubro.