UNIP – Universidade Paulista

CURSO DE CIÊNCIA DA COMPUTAÇÃO

CAMPUS ARARAQUARA

Samuel Lopes Grigolato

CONTEÚDO DINÂMICO E INTERATIVO COM HTML5

Araraquara, Dezembro de 2012.

Samuel Lopes Grigolato

CONTEÚDO DINÂMICO E INTERATIVO COM HTML5

Monografia desenvolvida durante as disciplinas “Trabalho de Curso I” e “Trabalho de Curso II”, apresentada ao Curso de Ciência da Computação da Universidade Paulista, campos Araraquara, como pré-requisito para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Leandro Carlos Fernandes

Araraquara, Dezembro de 2012.

Samuel Lopes Grigolato

Conteúdo Dinâmico e Interativo com HTML5

Monografia desenvolvida durante as disciplinas “Trabalho de Curso I” e “Trabalho de Curso II”, apresentada ao Curso de Ciência da Computação da Universidade Paulista, campos Araraquara, como pré-requisito para a obtenção do título de Bacharel em Ciência da Computação, sob a orientação do Professor Leandro Carlos Fernandes.

Data de aprovação: 03 de Dezembro de 2012

Banca Examinadora

Professor (a): Leandro Carlos Fernandes

Mestre em Ciências ICMC/USP, UNIP

Professor (a): Danielle B. Colturato

Mestre em Ciências ICMC/USP, UNIP

Professor (a): Marcelo Criscuolo

Mestre em Ciências ICMC/USP, UNIP

Ao futuro da Internet.

Agradeço aos meus amigos, sócios, familiares e, principalmente, minha namorada, por tolerarem pacientemente meus intermináveis dias de dedicação à pesquisa.

“Extrair informação da Internet é como pegar um drinque de um hidrante”

Mitchell Kapor (tradução nossa)

RESUMO

A quinta versão do protocolo HTML disponibiliza o componente Canvas, uma nova maneira de desenvolver dinamismo para a Internet. Este trabalho avalia até que ponto esse componente substitui ou ultrapassa a forma convencional de desenvolvimento desse tipo de conteúdo. Essa avaliação pode auxiliar na decisão de futuros investimentos na área, seja em treinamento de pessoal ou em aquisição de tecnologia. O trabalho é composto de comparações entre o Adobe Flash e o Canvas, respectivamente, uma das formas convencionais e a forma proposta. Essas comparações são por fim utilizadas como insumo para uma série de discussões, comparando as tecnologias em quesitos como produtividade e curva de aprendizado, entre outros. No final têm-se resultados favoráveis para a utilização do Canvas tão logo amadureça a especificação HTML5.

Palavras-chave: Desenvolvimento Web; Usabilidade; Redes de Computadores.

ABSTRACT

Title: *“Dynamic and Interactive Content with HTML5”*

The fifth version of the HTML protocol brings the Canvas component, a new way to develop dynamic content to the Internet. This paper evaluates how this component replaces the conventional way to develop this kind of content. This evaluation may support future investment decisions on people capacitation or technology acquisition. The paper is composed of comparisons between Adobe Flash and Canvas, respectively, one of the conventional ways and the proposed way. These comparisons are finally digested in a series of discussions, comparing both technologies in topics like productivity and learning curve. Finally we are presented to favorable results to the use of Canvas, as soon as the HTML5 specification stabilizes.

Keywords: *Web Development; Usability; Computer Networks.*

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HTML	Hyper Text Markup Language
WWW	World Wide Web
W3C	World Wide Web Consortium
CSS	Cascading Style Sheet
PNG	Portable Network Graphics
DOM	Document Object Model

SUMÁRIO

1. INTRODUÇÃO

Vivemos em um planeta que passa por mudanças constantemente, sejam mudanças naturais ou provocadas pelo homem. Dentre as consequentes de ação humana, podemos citar os avanços tecnológicos.

Compondo os avanços tecnológicos das últimas décadas, destaca-se o surgimento dos computadores e, ainda mais recentemente, a aparição da Internet. Hoje em dia é certamente impossível imaginar o dia-a-dia da sociedade sem a grande rede.

Inicialmente projetada para servir como meio de publicação de documentos de hipertexto, através do protocolo Hypertext Transfer Protocol (HTTP), a Internet obteve tamanho sucesso que em poucos anos o projeto inicial não atendia mais os anseios dos produtores de conteúdo eletrônico.

Dentre as vertentes revolucionárias do projeto inicial da Internet, citaremos a evolução da interatividade das páginas, a qual era notavelmente limitada até então. A construção de extensões[1] para os navegadores[2] foi a maneira encontrada para atender as necessidades dos usuários mais exigentes quanto a usabilidade[3] das suas aplicações.

Como exemplo dessas extensões, podemos citar o Adobe Flash e o Microsoft Silverlight. Ambos têm por objetivo aumentar o dinamismo das páginas. Até o surgimento da versão número 5 da especificação HTML (a qual será descrita em mais detalhes nos próximos parágrafos), essas soluções reinavam absolutas no mercado de interatividade na Internet.

Como sabemos, todo esforço para substituição de um conceito existente é alimentado por seus problemas. Com as extensões de interatividade (Flash, Silverlight, etc.) não é diferente. Alguns desses problemas identificados com a abordagem até então dominante desse aspecto da Internet são:

---

[1] Componentes que adicionam funcionalidades a um Software.

[2] Também chamados de Agentes de Usuário, são Softwares utilizados para navegar pelas páginas através do protocolo HTTP.

[3] Usabilidade é a forma como o usuário “sente” o sistema, ou seja, o conjunto de ações necessárias para manipulá-lo e visualizá-lo.

- Rastro de instalação: a necessidade de instalação dessas extensões na máquina do usuário foi motivo suficiente para afugentar os mais leigos por muito tempo. No entanto, nas versões mais recentes dos navegadores esse processo é fácil e exige o mínimo de esforço dos usuários, diminuindo o valor deste argumento.
- Interação com os outros componentes: a arquitetura das soluções desenvolvidas nessas extensões dificulta (mas não impossibilita) a comunicação com os outros componentes de tela.
- Especialização dos profissionais: como os editores gráficos e linguagens utilizadas para o desenvolvimento dos componentes interativos são diferentes dos padrões nativos das páginas existe a necessidade de capacitar os profissionais que irão desenvolver e manter esse conteúdo.

Antes de introduzir o conceito que veio para competir com as extensões de navegador, faz-se necessário discutir sobre a especificação Hyper Text Markup Language (HTML). Essa especificação, mantida por uma organização chamada World Wide Web Consortium (W3C), dita todas as regras para a transmissão de páginas HTML. Em outras palavras, essa especificação regula o intermédio entre os servidores de conteúdo eletrônico (que produzem o código das páginas HTML) e os agentes de usuários (que traduzem esse conteúdo e apresentam o resultado para o usuário).

A existência de tal especificação facilita a construção de páginas que funcionem em diversos navegadores simultaneamente, e que o conhecimento necessário para o desenvolvimento fique centralizado em um único conceito ao invés de vários conceitos proprietários com mesmo fim.

Atualmente, a W3C vem amadurecendo a quinta versão da especificação HTML, cujas novidades servem de combustível para este trabalho de pesquisa. Dentre os principais novos aspectos da especificação, podemos citar:

- Suporte a elementos de vídeo;
- Suporte a elementos de áudio;
- Suporte a uma série de controles de captura de dados, como seletor de cor, seletor do tipo *faixa*, seletor de data, etc.

- Componente que permite a apresentação de elementos gráficos.

O componente gráfico citado na lista acima, que podemos chamar de Canvas, é o núcleo de toda a discussão que será desenvolvida neste trabalho. Este componente promete ser uma alternativa a altura para o mercado de conteúdo dinâmico e altamente interativo para a Internet, e é exatamente este ponto que pretendemos demonstrar (ou refutar).

Para ser efetivo neste objetivo, este trabalho será composto por comparações entre a ferramenta Adobe Flash e o componente Canvas, das mais básicas até a análise de um caso de uso completo, sempre alicerçando os argumentos em demonstrações práticas dos conceitos em ambas as ferramentas.

No início do desenvolvimento, trataremos sobre requisitos de configuração, analisaremos um clássico *Olá, Mundo!* em ambas as tecnologias e veremos como desenhar elementos geométricos simples.

Na parte intermediária, discutiremos sobre a utilização de elementos textuais em nossos aplicativos, bem como importação e manipulação de imagens externas. Além disso, conceitos de animação serão apresentados.

Nas seções finais antes das considerações, teremos a abordagem da interação com o usuário, e por último o estudo de caso de um jogo chamado Pong (sim, o clássico).

Para concluir, serão apresentados diversos critérios comparativos, juntamente com sua importância para a comparação desejada entre as ferramentas. Por último será apresentada uma conclusão para a questão principal, *moto* do trabalho: “O Canvas pode substituir as soluções tradicionais para o desenvolvimento de conteúdo dinâmico?”.

2. DESENVOLVIMENTO

O capítulo principal deste trabalho é composto por diversas seções, cada uma abordando um aspecto da criação de aplicações dinâmicas. O objetivo de cada seção é apresentar o conceito que será abordado de forma genérica, especificar uma aplicação e explicar, passo a passo, como desenvolvê-la em Flash e utilizando o componente Canvas.

Todas essas explicações servirão de insumo para as considerações finais, que discorrerão sobre a viabilidade da substituição da tecnologia original de mercado (Flash) pela novidade (Canvas).

2.1. OLÁ, MUNDO!

Nesta seção seremos apresentados aos requisitos de instalação e configuração necessários para:

- Executar aplicações dinâmicas;
- Desenvolver novos aplicativos.

Note que durante todo o trabalho, ao citar exemplos como os acima enumerados, estaremos nos referindo as duas ferramentas em comparação, ou seja, tanto o Adobe Flash quanto o componente Canvas.

Depois de nos habituarmos com a infraestrutura de desenvolvimento e execução, analisaremos passo a passo o desenvolvimento de um simples *Olá, Mundo!* em ambas as tecnologias.

2.1.1. Executando Aplicações Dinâmicas

Para que nosso trabalho de comparação tenha valor, é necessário que tenhamos conhecimento do resultado que esperamos obter com a utilização das ferramentas estudadas. Para isso, o primeiro passo é conseguir visualizar as aplicações desenvolvidas.

Começaremos explorando os requisitos necessários para visualização de aplicativos desenvolvidos na ferramenta Adobe Flash na sua versão 11 (a mais

recente no momento da escrita deste trabalho). Na Tabela 1 encontram-se os *softwares* e requisitos mínimos para execução da ferramenta[4]. Uma especificação mais abrangente dos requisitos pode ser encontrada em Chun (2010, p. 7).

Antes de visualizarmos uma aplicação de exemplo, precisamos instalar o *software* necessário, que consiste em:

- Um sistema operacional suportado;
- Um navegador suportado;
- O Adobe Flash Player, disponibilizado pela fabricante no endereço: http://get.adobe.com/br/flashplayer/

Note que nas versões mais recentes dos navegadores, o Adobe Flash Player já vem instalado por padrão, neste caso ao acessar o endereço acima aparecerá uma notificação informando que o navegador já está pronto para executar aplicações desenvolvidas em Flash.

Item	Versão/Quantidade Mínima
Navegador	• Internet Explorer 7+ • Mozilla Firefox 4+ • Google Chrome (qualquer versão) • Safari 5+ • Opera 11+
Sistema Operacional	• Windows XP (x86) • Windows Server 2003 (x86) • Windows Server 2008 (x86) • Windows Vista (x86) • Windows 7 (x86 ou x64) • Mac OS X v10.6 • Max OS X v10.7

[4] Para mais detalhes, visite a página do fabricante no seguinte endereço: http://www.adobe.com/products/flashplayer/tech-specs.html

• Red Hat Enterprise Linux 5.6+ • OpenSUSE 11.3+ • Ubuntu 10.04+

Tabela 1 Requisitos mínimos para execução do Adobe Flash 11

Depois de instalar o Adobe Flash Player, iremos executar uma aplicação de exemplo, chamada *Olá, Mundo!*. Para isso, existem duas maneiras:

- Abrir o arquivo *CAP01.olamundo/CAP01.olamundo.flash.html*[5].
- Abrir o arquivo *CAP01.olamundo/CAP01.olamundo.flash.swf*. É importante observar que para abrir este arquivo talvez seja necessário utilizar a opção "Abrir" do navegador desejado, pois normalmente esse tipo de arquivo não está associado com um programa específico (o que é necessário para que o clique duplo funcione como opção de execução).

Discutiremos as diferenças de cada uma dessas formas no próximo tópico.

Agora que conseguimos visualizar as aplicações desenvolvidas em Flash, faremos o mesmo processo para as aplicações em Canvas.

Para executar aplicações desenvolvidas em Canvas, diferentemente do demonstrado para o Flash, não é necessário instalar nenhum *software* adicional. Isso ocorre devido à natureza do Flash, que em sua concepção é uma extensão do navegador. O Canvas é parte integrante do HTML5, que por sua vez é uma especificação que os agentes de usuário (navegadores) devem atender. Portanto, para executar aplicações em Canvas, basta possuir um navegador que suporte a versão 5 da especificação HTML.

Existem várias formas de verificar se o seu navegador suporta (e até que ponto) a especificação HTML5. Na Internet podemos encontrar diversos sites que apresentam as chamadas *Matrizes de Compatibilidade*, como por exemplo: http://caniuse.com/. Neste site, ao pesquisar pelo termo *Canvas*, nos é

[5] Todos os recursos utilizados neste trabalho foram disponibilizados como anexos e podem ser utilizados para qualquer fim.

apresentada uma matriz de suporte bem intuitiva. A Figura 1 mostra a matriz de compatibilidade do Canvas obtida no momento da elaboração deste trabalho.

Todos os exemplos apresentados neste trabalho (independente da tecnologia) foram testados no navegador Google Chrome 21, no entanto devem funcionar de forma análoga nos navegadores que suportam a especificação do Canvas e o Flash Player 11. Vale observar que a especificação HTML5 é recente e passa por constantes melhorias, o que compromete a vida útil dos exemplos produzidos para o trabalho.

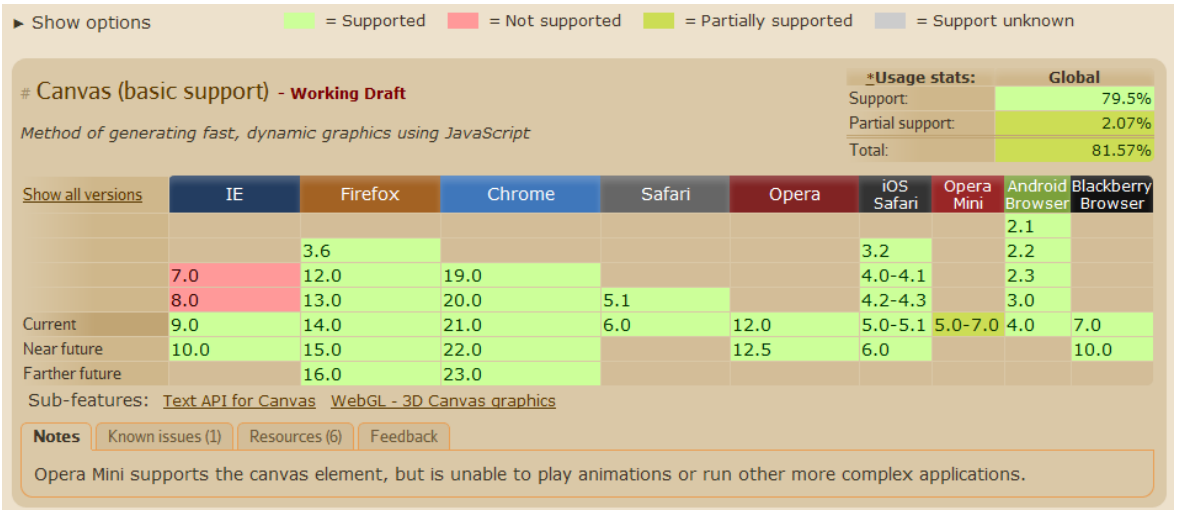

Figura 1 Matriz de compatibilidade do Canvas

Depois de garantir que possuímos um navegador com suporte ao elemento Canvas, basta abrirmos o arquivo *CAP01.olamundo/CAP01.olamundo.canvas.html* para visualizarmos uma aplicação semelhante à apresentada em Flash. A Figura 2 mostra essa aplicação.

Figura 2 Aplicação de exemplo em Canvas

2.1.2. Desenvolvendo Novos Aplicativos

Agora que já sabemos como executar as aplicações desenvolvidas em ambas as tecnologias, podemos começar a desenvolver novas aplicações. Para isso será necessário outro conjunto de ferramentas, que podemos chamar de *Ambiente de Desenvolvimento* dessas aplicações.

Começaremos configurando nosso ambiente de desenvolvimento para aplicações Flash. Como é uma biblioteca proprietária, é natural que a ferramenta necessária para desenvolver aplicações deste tipo seja uma ferramenta paga. Para codificar em Flash é necessário possuir uma cópia do aplicativo Adobe Flash, atualmente em sua versão CS5. Além desse *software*, será necessário um editor HTML (como o Notepad++[6]). Vale observar que Chun (2010, p. 362) [em nota] sugere a utilização do *software* Adobe Dreamweaver para edição de arquivos HTML. A Figura 3 mostra a tela inicial do Adobe Flash.

Para desenvolver aplicações em Canvas, não é necessário nenhum *software* proprietário ou externo aos já utilizados para o desenvolvimento de páginas HTML. Todo o código necessário para manipulação do Canvas está contido no próprio arquivo HTML, seja em sua linguagem de marcação ou na sua linguagem de Scripts (JavaScript). Portanto, precisamos apenas de um editor HTML para essa tarefa.

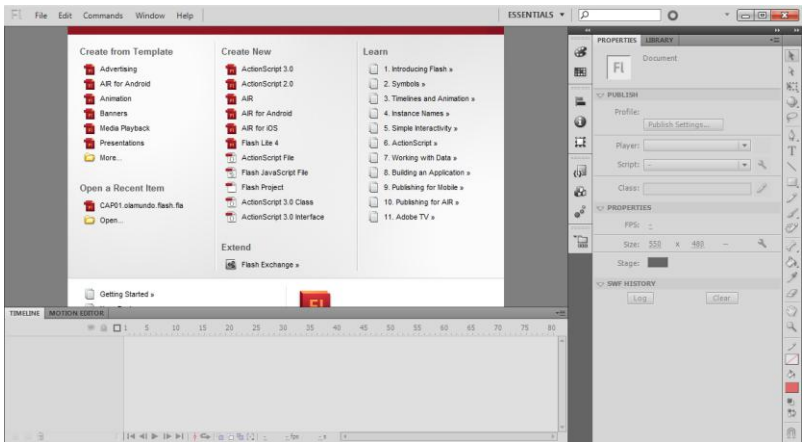

Figura 3 Tela inicial do Adobe Flash CS5

[6] Para mais informações e download do Notepad++ visite o site: http://notepad-plus-plus.org/

2.1.3. Desenvolvendo um Simples Aplicativo

Com os dois ambientes de desenvolvimento configurados, podemos colocar a mão na massa. Desenvolveremos a aplicação mostrada como exemplo no tópico anterior, tanto em Flash quanto em Canvas. Antes de começar a produzir, é importante observamos as características em comum, que desejamos alcançar independente da tecnologia utilizada:

- Trabalharemos em um *quadro* dentro de uma página HTML, que fixaremos o tamanho em *400 pixels* de largura por *230 pixels* de altura.
- O fundo desse quadro será de um tom amarelo claro, para diferenciar do restante da página, que por padrão é branca.
- No centro do quadro, estará escrito o texto *Olá, Mundo!* em vermelho, fonte *Tahoma* e tamanho *50*.

Primeiro, faremos a versão em Flash. Para isso, abra o Adobe Flash, e selecione a opção *New* dentro do menu *File*. Neste ponto serão apresentados diversos tipos e modelos diferentes que podem ser utilizados, para o nosso exemplo utilizaremos o tipo padrão *Action Script 3.0*. Antes de confirmar, altere as propriedades *Width*, *Height* e *Background color* para atender as especificações que apresentamos no parágrafo acima. A Figura 4 mostra a tela com o novo aplicativo criado.

Agora que já temos nosso quadro, precisamos adicionar o texto centralizado. Para isso, selecione a ferramenta de texto (o menu de ferramentas encontra-se na extrema direita da tela, a ferramenta de texto possui como símbolo uma letra *T* em caixa alta). Ao selecionar a ferramenta, a aba de propriedades à direita apresentará as opções disponíveis para manipular textos. Modifique os campos *Family*, *Size* e *Color* para atender as especificações da nossa aplicação.

Depois de configurar a ferramenta, clique em algum ponto do quadro e escreva o texto *Olá, Mundo!*, aperte a tecla ESC para finalizar a escrita.

Temos agora nosso texto, porém, mesmo que tenhamos tentado colocá-lo no centro do quadro, provavelmente não conseguimos obter o centro perfeito na primeira tentativa. Para corrigir esse problema existem ferramentas específicas de alinhamento, que utilizaremos a seguir.

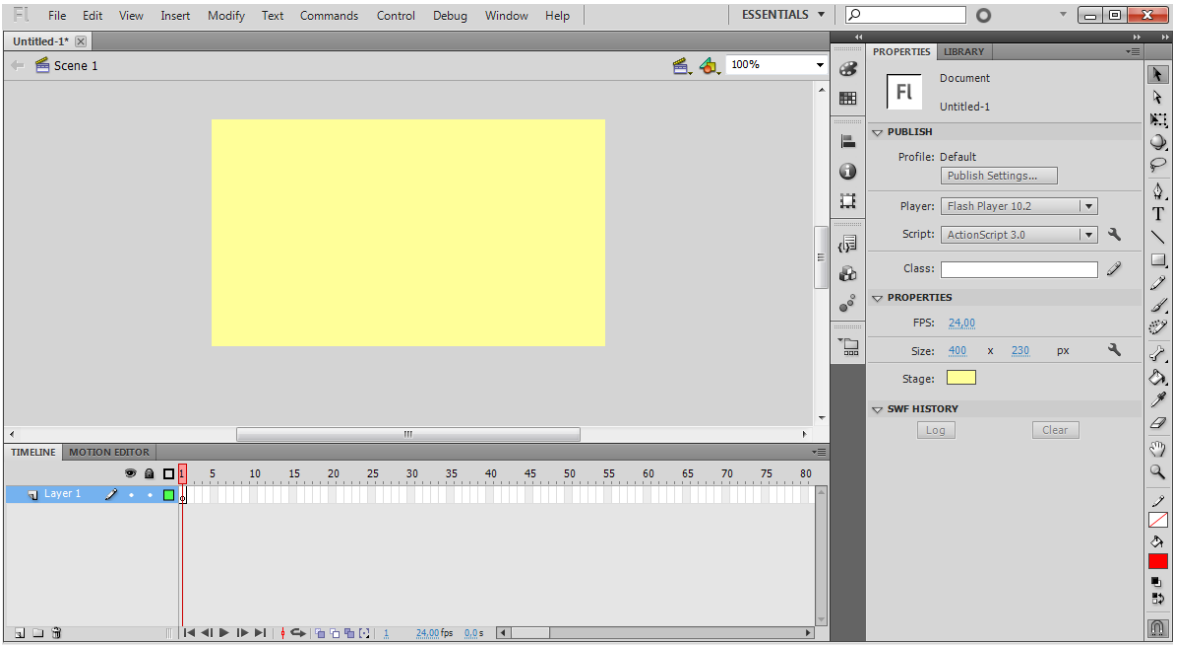

Figura 4 Novo aplicativo em Flash

Para manipular o alinhamento do nosso texto, acione a ferramenta de seleção (primeira ferramenta da barra, simbolizada por um ponteiro de *mouse* escuro), clique sobre o objeto de texto (um contorno azul será apresentado ao redor do texto, simbolizando que a seleção encontra-se neste objeto), e então faça o seguinte:

- Acesse o menu *Modify*;
- Dentro da opção *Align*, assegure que o item *Align to Stage* esteja marcado (se não estiver basta clicar uma vez sobre o item, e acessar o menu novamente) – isso faz com que as operações de alinhamento sejam aplicáveis com relação ao quadro como um todo, e não apenas entre os objetos selecionados;
- Acione a opção *Horizontal Center*;
- Acione a opção *Vertical Center*.

Neste ponto o texto deve estar centralizado em relação ao quadro, conforme a Figura 5.

Nossa aplicação já está construída, porém, como deve ter observado, não sabemos como transportá-la para o navegador, até agora. Para isso, precisamos

compreender como funciona a arquitetura de código fonte e objeto executável do Flash:

- Os aplicativos que desenvolvemos em Flash são armazenados em arquivos de código fonte sob a extensão *FLA*;
- Dentro do Adobe Flash, temos a funcionalidade de exportação, que dentre vários formatos, nos permite exportar os aplicativos para o formato *SWF*;
- Esse formato é acoplado a nossas páginas HTML, através de código específico que executa a extensão de navegador capaz de ler e executar seu conteúdo (Flash Player).

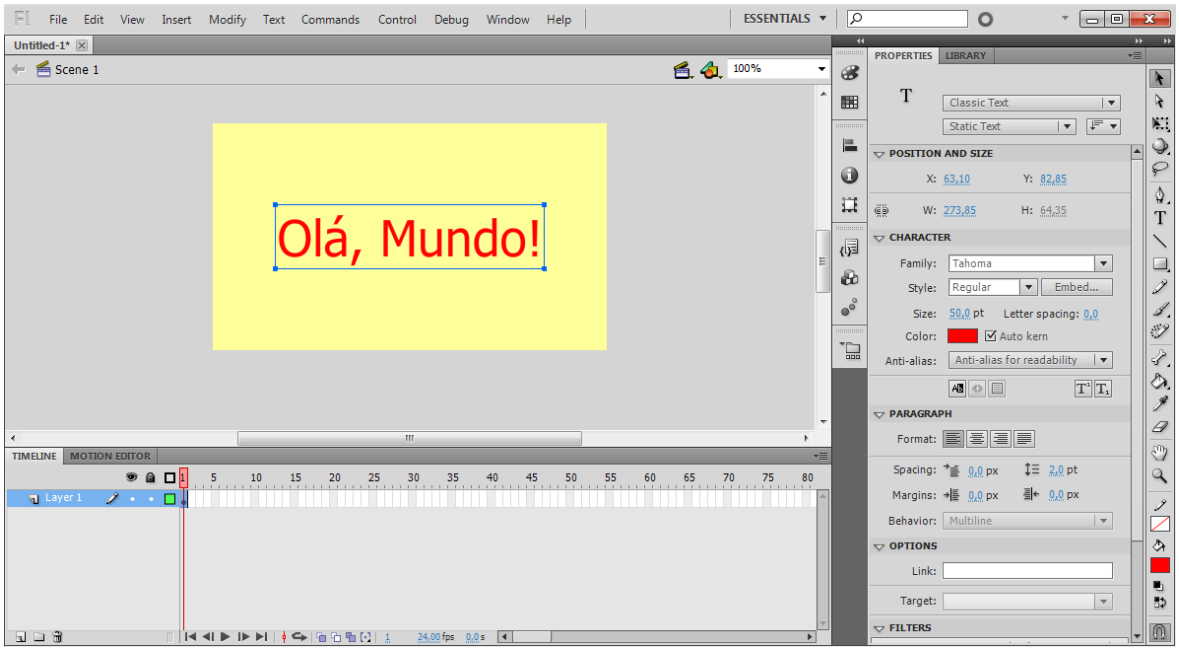

Figura 5 Texto centralizado no quadro

Antes de exportar nosso pequeno aplicativo, vamos armazená-lo em algum lugar para não perdermos o trabalho que já tivemos até agora. Para isso, acione a opção *Save* dentro do menu *File*. Selecione o local e nome desejados (repare na extensão *FLA*) e confirme.

Depois de salvar o aplicativo, acione a opção *Publish Settings* dentro do menu *File*. Essa opção irá apresentar uma janela permitindo que configuremos a maneira que desejamos que nosso aplicativo seja exportado. Dentre as opções,

utilizaremos as opções *Flash (.swf)* e *HTML Wrapper*. Depois de configurar as opções de publicação, acione a opção *Publish* do menu *File*. Essa ação irá gerar dois arquivos no mesmo diretório onde salvamos o arquivo de código fonte, sob as extensões *SWF* e *HTML*.

Chun (2010, p. 362) trata de forma detalhada todas as opções disponíveis ao publicar uma aplicação Flash.

Agora que publicamos nosso trabalho, podemos visualizá-lo, abrindo o arquivo HTML em nosso navegador. Observe que o fundo da página não está branco, conforme esperávamos, mas sim da mesma cor que o fundo do nosso quadro. Isso é uma peculiaridade do publicador *HTML Wrapper* que selecionamos anteriormente, que pode ser contornada da seguinte forma:

- Utilizando o editor HTML escolhido anteriormente, abra o arquivo HTML gerado na publicação;
- Procure a linha de código CSS[7] contendo a instrução `background-color: #ffff99;` (note que `#ffff99` pode estar diferente, dependendo da cor que escolheu na criação do aplicativo);
- Modifique essa instrução, alterando a cor `#ffff99` pela palavra `white`;
- Salve o arquivo e execute-o novamente.

Finalizada a versão em Flash, precisamos desenvolver agora a versão em Canvas. Para isso, criaremos um novo arquivo com extensão HTML, e editaremos seu conteúdo utilizando o editor escolhido na preparação do ambiente de desenvolvimento.

Juntamente com o material disponibilizado com este trabalho, existe um arquivo no caminho *CAP01.olamundo/CAP01.olamundo.templatecanvas.html*. Utilizaremos esse arquivo como base para todos os desenvolvimentos sob essa plataforma. Localize esse arquivo, e copie seu conteúdo para o novo arquivo criado.

[7] Cascading Style Sheet: linguagem de definição de estilos de páginas HTML.

Como a análise das instruções CSS, HTML e JavaScript não fazem parte do escopo deste trabalho, focaremos nas novidades que foram introduzidas especificamente para adicionar o quadro do Canvas na página.

Na linha 42 do nosso arquivo de modelo, existe um elemento do tipo `canvas`. Este elemento foi acrescido na especificação HTML5, e serve para introduzir um quadro nesse ponto específico da página. Podemos também identificar os seguintes atributos:

- `id`: utilizamos esse atributo para referenciar esse quadro do código JavaScript;
- `width`: largura do quadro, em pixels;
- `height`: altura do quadro, em pixels.

Dentro do elemento, observamos um texto alertando o usuário que seu navegador não possui suporte para o elemento Canvas. Os navegadores que não entendem uma determinada marcação simplesmente escrevem seu conteúdo como texto simples na tela, já os que a entendem, no caso do elemento Canvas, desconsideram seu conteúdo textual e o interpretam como um quadro gráfico.

Na linha 27, existe uma chamada ao método `getContext`[8] do elemento `canvas`, que por sua vez foi recuperado através de seu identificador pela chamada ao método `getElementById` do objeto `document`. O método `getContext` recebe um parâmetro, do tipo `string`, que especifica qual o tipo de contexto desejado. Em todos os exemplos desse trabalho utilizaremos o contexto `2d`, porém existem outros, com destaque para o contexto `webgl`, que permite o desenvolvimento de aplicações gráficas intensivas dentro do navegador[9]. Uma detalhada introdução ao contexto `webgl` pode ser encontrada em Rowell (2011, p. 269).

Na linha 31, temos a declaração da função `renderizaTela`, que não possui código até o momento. Dentro dessa função, temos acesso a variável `contexto`, que faz referência ao contexto `2d` obtido do elemento `canvasUm`.

---

[8] Para detalhes dos métodos disponíveis na especificação, visite o site: http://dev.w3.org/html5/2dcontext/

[9] Para detalhes do contexto WebGL, visite o site: http://www.khronos.org/webgl/

Antes de continuar podemos validar a nossa página HTML. Ao abri-la, nada de interessante será apresentado, apenas o fundo branco que já conhecemos. Neste ponto, no entanto, podemos utilizar quaisquer ferramentas de depuração que desejarmos (como o Firebug ou o Console do Chrome) para assegurarmos que nenhum problema de JavaScript está impedindo nosso elemento `canvas` de ser interpretado pelo navegador.

Agora que já temos nosso modelo de aplicação Canvas, precisamos preenchê-lo de forma que fique semelhante à aplicação que desenvolvemos em Flash. O primeiro passo será o preenchimento do fundo do quadro no tom amarelo claro que utilizamos anteriormente. Para isso, preencha a função `renderizaTela`, conforme o Código-fonte 1:

```
function renderizaTela() {

 contexto.fillStyle = "#ffff99";
 contexto.fillRect(0, 0, canvasLargura, canvasAltura);

}
```

Código-fonte 1 Preenchendo o fundo do Canvas

Depois de salvar e executar a página, um quadro amarelo deverá ser apresentado. Para finalizar nossa primeira aplicação, precisamos escrever o texto “Olá, Mundo!” da mesma forma que fizemos em Flash. Como já percebemos, com o elemento Canvas não temos até o momento uma ferramenta de edição visual como o Adobe Flash, então a solução será adicionar mais um trecho de código JavaScript, conforme o Código-fonte 2.

Vamos agora salvar e executar nosso trabalho. Se tudo deu certo, temos que obter a mesma aplicação que visualizamos na Figura 2, onde aprendemos a executar uma aplicação desenvolvida utilizando Canvas.

```
function renderizaTela() {

 contexto.fillStyle = "#ffff99";
 contexto.fillRect(0, 0, canvasLargura, canvasAltura);

 contexto.fillStyle = "#ff0000";
 contexto.font = "50px Tahoma";

 var mensagem = "Olá, Mundo!";
```

```
  var mensagemDimensoes = contexto.measureText(mensagem);
  var mensagemLargura = mensagemDimensoes.width;
  var mensagemAlturaEstimada = 28; // não existe forma de obter esse
valor exato até o presente momento

  var mensagemPosicaoX = (canvasLargura / 2) - (mensagemLargura / 2);
  var mensagemPosicaoY = (canvasAltura / 2) + (mensagemAlturaEstimada /
2);
  contexto.fillText(mensagem, mensagemPosicaoX, mensagemPosicaoY);

}
```

Código-fonte 2 Escrevendo no Canvas

2.2. DESENHANDO ELEMENTOS GEOMÉTRICOS

Na seção anterior vimos como criar nossa primeira aplicação em Flash e utilizando Canvas. Apresentamos os requisitos necessários, ambiente de execução e de desenvolvimento para cada uma das tecnologias.

Agora veremos como desenhar simples elementos geométricos, com cores variadas de contorno e preenchimento. Começaremos pela especificação da nossa aplicação:

- Quadro de *400* por *230 pixels* com fundo amarelo claro;
- Uma linha vermelha, de espessura *5*, cantos arredondados, posicionada do ponto *(200, 10)* até o ponto *(200, 210)*;
- Um retângulo azul claro com bordas azul escuro, posicionado no ponto *(10, 10)* até o ponto *(190, 210)*;
- Um círculo verde claro com bordas verde escuro, com centro no ponto *(300, 100)* e raio *90*;
- Um triângulo laranja com bordas de tom mais escuro que o fundo, passando pelos pontos: *(210, 190)*, *(210, 210)* e *(390, 210)*.

Para auxiliar o entendimento, a Figura 6 apresenta o resultado esperado da aplicação especificada.

Agora que sabemos o que devemos construir, partiremos para a versão em Flash. Abra o Adobe Flash, crie um novo aplicativo utilizando o modelo Action Script 3.0.

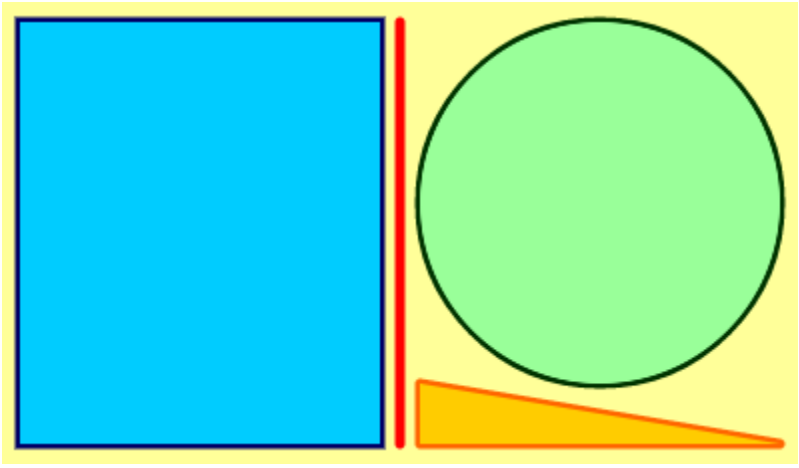

Figura 6 Especificação dos elementos geométricos

Modifique o tamanho do quadro e a cor de fundo da aplicação, conforme vimos no capítulo anterior. Depois disso, selecione a ferramenta de linha (simbolizada por uma linha na diagonal). Na aba de propriedades da ferramenta, altere a cor e a espessura (*stroke*) de acordo com nossa especificação. Por fim, clique e arraste uma linha dentro do quadro, tentando posicioná-la conforme a especificação.

Por mais que tentamos posicionar a linha no lugar correto, dificilmente temos êxito, a menos que sejam efetuados ajustes nas propriedades do objeto.

Para corrigir o posicionamento da linha, acione a ferramenta de seleção, clique sobre a linha recentemente adicionada, ela deve passar a apresentar pontos sobre ela simbolizando a seleção. Com o objeto linha selecionado, na aba referente às suas propriedades, altere os valores *X*, *Y* e *H* (altura) para, respectivamente, *200*, *10* e *210*.

Você poderá notar que a propriedade *W* (largura) não foi mencionada. Isso ocorreu devido ao fato de que em uma linha totalmente vertical, essa propriedade se apresenta desabilitada (com texto na cor cinza ao invés de azul). Se a sua reta não ficou totalmente vertical, ao invés de alterar a propriedade do objeto, o que não irá funcionar, crie outra reta ou então ajuste a mesma utilizando a ferramenta de seleção, clicando em algum dos extremos da reta e arrastando a ponta até que a mesma fique totalmente na vertical. A Figura 7 ilustra essa operação.

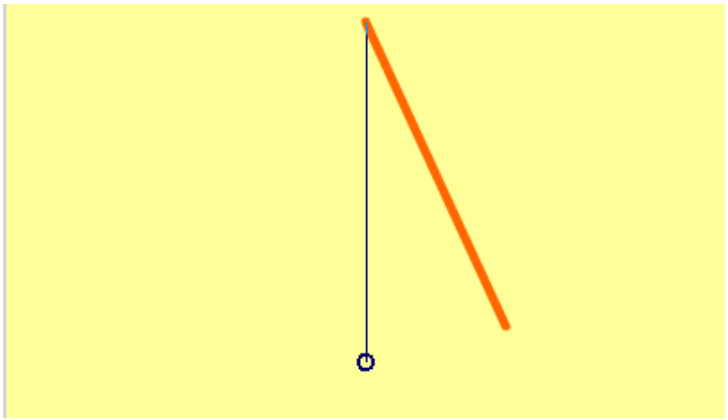

Figura 7 Corrigindo o ângulo da reta

Depois da linha criada, passaremos para o retângulo. Acione a ferramenta de retângulo (simbolizada por um retângulo). Nas propriedades da ferramenta, modifique a cor da borda (simbolizada por um pequeno lápis), de fundo e tamanho da borda (*stroke*). Desenhe um retângulo aproximadamente no local desejado, corrigindo-o depois de forma semelhante à forma que foi feito para a linha, alterando as propriedades *X*, *Y*, *W* e *H* para *10*, *10*, *180* e *210*, respectivamente. Note que a forma de selecionar este objeto difere da seleção de retas, como veremos no próximo parágrafo.

Selecione a ferramenta de seleção, clique em qualquer ponto do preenchimento do retângulo. O mesmo aparentará estar selecionado, porém, se observar em detalhes, perceberá que as bordas não estão selecionadas. Se modificar as propriedades com essa seleção parcial, a borda não sofrerá nenhuma alteração, desfigurando o elemento. Para selecionar o retângulo como um todo, ao invés de clicar no preenchimento do mesmo, efetue uma operação de *laço com o mouse*, selecionando uma área que contenha o preenchimento e as bordas inteiras, conforme a Figura 8 nos mostra. Chun (2010, p. 46) apresenta as formas de seleção disponibilizadas pelo Adobe Flash.

O próximo passo será a criação do círculo. A ferramenta para desenharmos círculos está “escondida” atrás da ferramenta de retângulo. Repare que algumas ferramentas possuem um pequeno triângulo no canto inferior direito de seu ícone. Isso representa a existência de outras ferramentas, de mesma categoria, escondidas sob a mesma. Para visualizar essas ferramentas, clique exatamente sobre o pequeno triângulo (veja a Figura 9 para detalhes).

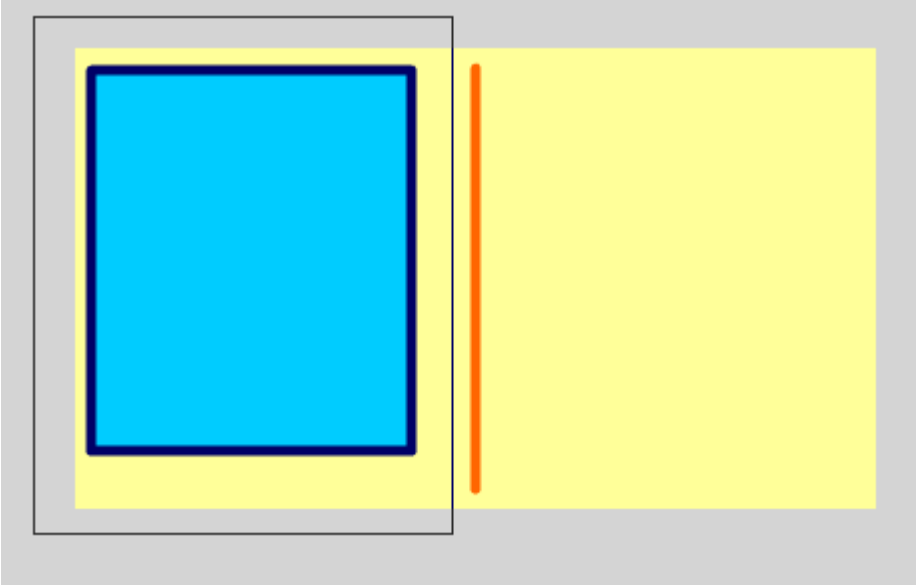

Figura 8 Selecionando um retângulo completo

Uma das ferramentas escondidas do retângulo é a ferramenta de círculos. Selecione-a e altere as propriedades de cor conforme nossa especificação. Depois disso, desenhe o círculo no quadro, modificando suas propriedades *X*, *Y*, *W* e *H* para *210*, *10*, *180* e *180*.

Para finalizar, iremos construir nosso triângulo. Diferentemente das outras estruturas, não temos uma ferramenta especificamente para triângulos, precisamos construí-lo com a combinação de outras ferramentas, no caso, das ferramentas caneta e balde de tinta.

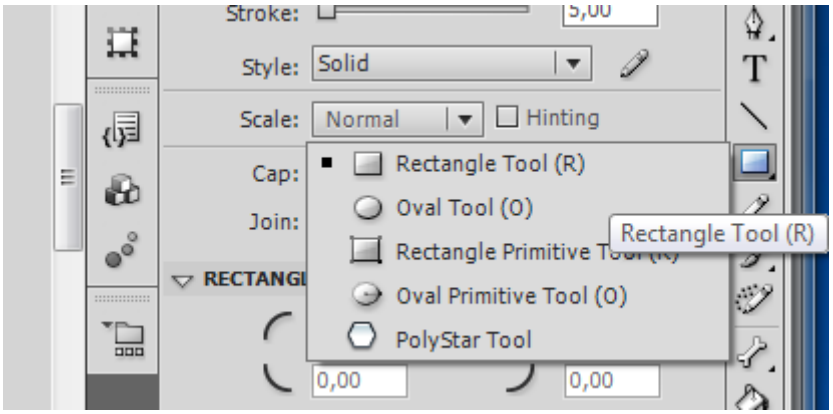

Figura 9 Ferramentas "escondidas" no Adobe Flash

Primeiramente, selecione a ferramenta caneta. Essa ferramenta nos permite desenhar linhas retas demarcando suas extremidades. Nas propriedades da caneta, selecione a correta cor de borda do nosso triângulo (a cor de fundo será tratada mais a frente). Clique, no quadro, uma vez sobre cada ponto onde o triângulo deve ser desenhado. Repare que conforme vamos definindo os pontos, uma linha vai sendo construída ligando-os. Para finalizar o contorno do nosso triângulo, clique novamente sobre o ponto inicial. A Figura 10 mostra o resultado obtido até o momento.

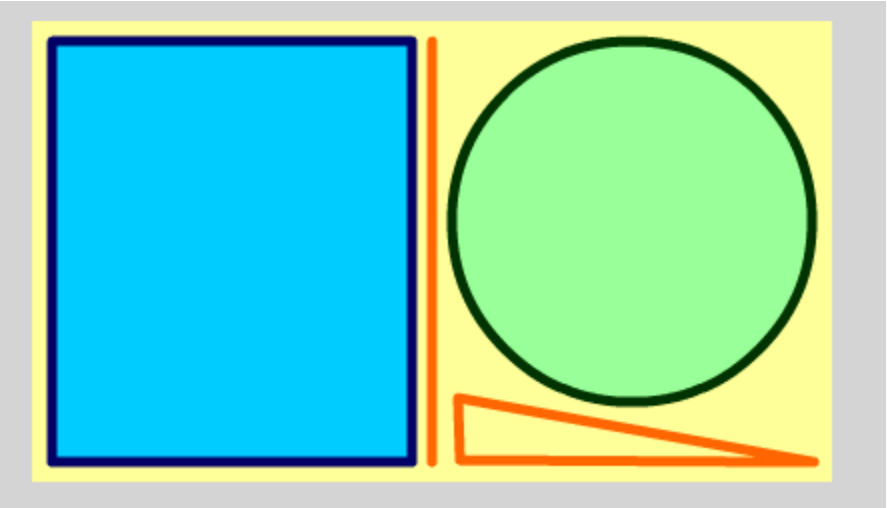

Figura 10 Construindo um triângulo em Flash com a ferramenta caneta

Para preencher o triângulo, selecione a ferramenta balde de tinta. Configure a cor de fundo na aba de propriedades. Por fim, clique em qualquer ponto dentro do triângulo.

Antes de etiquetarmos a aplicação para entrega, precisamos corrigir o nosso triângulo, para que fique na posição correta. Note que nenhuma das duas formas conhecidas até o momento servirá para selecionarmos esse objeto. A seleção da forma *laço com o mouse* irá selecionar também uma parte do nosso círculo, o que não é desejado.

Para selecionarmos esse triângulo completamente, precisaremos combinar a seleção por clique com a tecla *Shift*. Quando selecionamos um objeto juntamente com o acionamento da tecla, a seleção antiga é mantida, somando as seleções.

Modifique as propriedades *X*, *Y*, *W* e *H* do triângulo para *210*, *190*, *180* e *30*. Neste ponto a aplicação deve estar parecida com a Figura 6. Para finalizar, publique-a, adaptando o arquivo HTML para corrigir o fundo da página (como vimos no capítulo anterior) e verifique o trabalho realizado.

Agora desenvolveremos a versão em Canvas da mesma aplicação. Faça uma cópia do nosso modelo de aplicações em Canvas mostrado no capítulo anterior. Execute-a no navegador para garantir que está funcional. Para começar, definiremos cinco funções, uma para cada elemento geométrico e uma para cuidar do fundo da tela. A função `renderizaTela` ficará responsável por chamar essas outras funções na ordem correta. Inicialmente o corpo delas estará vazio. O Código-fonte 3 mostra como estamos até o momento.

```
renderizaTela();

function renderizaTela() {

 renderizaFundoDaTela();
 renderizaLinha();
 renderizaRetangulo();
 renderizaCirculo();
 renderizaTriangulo();

}
```

Código-fonte 3 Esqueleto da aplicação Canvas para elementos simples

Começaremos preenchendo a função `renderizaFundoDaTela`. O objetivo dessa função é pintar o fundo com a cor amarelo claro. Para isso, utilizaremos as propriedades `fillStyle` e `fillRect`, conforme o Código-fonte 4. Fulton (2011, p. 28) elabora essas propriedades.

```
function renderizaFundoDaTela() {
 contexto.fillStyle = "#FFFF99";
 contexto.fillRect(0, 0, canvasLargura, canvasAltura);
}
```

Código-fonte 4 Preenchendo o fundo da tela do Canvas

Depois do fundo preenchido, desenharemos a linha central, utilizando as funções de demarcação de caminho do Canvas. As principais a se considerar são:

- `beginPath`: inicia um novo caminho;
- `moveTo`: move o cursor imaginário para o pixel desejado, sem tracejar uma linha;
- `lineTo`: move o cursor imaginário para o pixel desejado, tracejando uma linha no processo;
- `stroke`: pinta toda a linha demarcada com as chamadas `lineTo` com o estilo de linha atualmente configurado no contexto;
- `fill`: preenche o polígono demarcado pelas linhas imaginárias desenhadas pelo cursor;
- `closePath`: finaliza o processo de marcação.

O Código-fonte 5 mostra como podemos demarcar nossa linha vermelha no quadro.

```
function renderizaLinha() {

 contexto.strokeStyle = "#FF0000";
 contexto.lineWidth = 5;
 contexto.lineCap = "round";

 var linhaPosicaoX1 = 200;
 var linhaPosicaoY1 = 10;
 var linhaPosicaoX2 = 200;
 var linhaPosicaoY2 = 220;

 contexto.beginPath();
 contexto.moveTo(linhaPosicaoX1, linhaPosicaoY1);
 contexto.lineTo(linhaPosicaoX2, linhaPosicaoY2);
 contexto.stroke();
 contexto.closePath();

}
```

Código-fonte 5 Desenhando uma linha no Canvas

Continuaremos o desenvolvimento adicionando o retângulo ao quadro. Para desenhar retângulos, o contexto oferece os métodos `strokeRect` e `fillRect`. O Código-fonte 6 mostra como o retângulo pode ser desenhado.

```
function renderizaRetangulo() {

 contexto.strokeStyle = "#000066";
 contexto.fillStyle = "#00CCFF";

 var retanguloX = 10;
 var retanguloY = 10;
```

```
  var retanguloLargura = 180;
  var retanguloAltura = 210;
  contexto.strokeRect(retanguloX, retanguloY, retanguloLargura,
retanguloAltura);
  contexto.fillRect(retanguloX, retanguloY, retanguloLargura,
retanguloAltura);

}
```

Código-fonte 6 Desenhando um retângulo no Canvas

Para desenhar nosso círculo, utilizaremos a demarcação de caminho já utilizada na linha reta, porém, desenharemos um círculo imaginário através da função `arc` ao invés dos métodos `moveTo` e `lineTo`, traçado em função do ponto central, raio e ângulos iniciais e finais informados como parâmetro. O Código-fonte 7 apresenta como devemos proceder. Essa e várias outras formas geométricas avançadas podem ser encontradas em Fulton (2011, p. 34).

```
function renderizaCirculo() {

  contexto.strokeStyle = "#003300";
  contexto.fillStyle = "#99FF99";

  var circuloX = 300;
  var circuloY = 100;
  var circuloRaio = 90;
  var circuloAnguloInicial = 0;
  var circuloAnguloFinal = (Math.PI / 180) * 360; // conversão de graus
em radianos
  contexto.beginPath();
  contexto.arc(circuloX, circuloY, circuloRaio, circuloAnguloInicial,
circuloAnguloFinal);
  contexto.stroke();
  contexto.fill();
  contexto.closePath();

}
```

Código-fonte 7 Desenhando um círculo no Canvas

Estamos quase finalizando nossa aplicação. Falta desenharmos nosso triângulo. Para esse elemento geométrico combinaremos uma sequência de chamadas a função `lineTo` da já conhecida demarcação de caminho do Canvas. O resultado está apresentado no Código-fonte 8.

```
function renderizaTriangulo() {

  contexto.strokeStyle = "#FF6600";
  contexto.fillStyle = "#FFCC00";
  contexto.lineJoin = "round";
```

```
  var trianguloX1 = 210;
  var trianguloY1 = 190;
  var trianguloX2 = 390;
  var trianguloY2 = 220;
  var trianguloX3 = 210;
  var trianguloY3 = 220;

  contexto.beginPath();
  contexto.moveTo(trianguloX1, trianguloY1);
  contexto.lineTo(trianguloX2, trianguloY2);
  contexto.lineTo(trianguloX3, trianguloY3);
  contexto.closePath();
  contexto.stroke();
  contexto.fill();

}
```

Código-fonte 8 Desenhando um triângulo no Canvas

Salve e execute a aplicação, o resultado obtido deve ser semelhante ao construído utilizando a ferramenta Adobe Flash.

2.3. UTILIZANDO IMAGENS

Na última seção fomos apresentados aos mecanismos de desenho de elementos geométricos simples, tanto em Flash quanto em Canvas. Agora iremos abordar outro tipo, muito utilizado e poderoso, que é o de imagens externas a aplicação.

Dificilmente desenvolveremos uma aplicação, que não seja trivial, apenas com texto e elementos geométricos simples. O logotipo do cliente já justifica a importação dos dados da imagem de outro arquivo, e não a reprodução do mesmo através de linhas, retângulos e triângulos.

A ideia é, através de mecanismos oferecidos pela plataforma em que estamos desenvolvendo, plotar imagens em formatos externos e conhecidos, como, por exemplo, o Portable Network Graphics (PNG).

Apresentada a ideia, vamos começar especificando nossa aplicação, como de costume:

- Quadro de *400* por *230* pixels;
- Fundo verde claro;
- Bola de tênis plotada na posição *(140, 90)*;
- Tábuas de madeira plotadas nas posições *(10, 80)* e *(360, 20)*.

A Figura 11 mostra o resultado esperado para a aplicação.

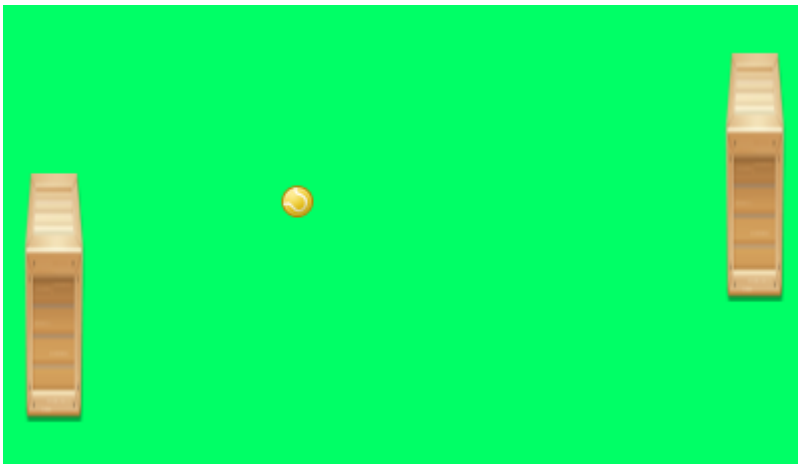

Figura 11 Aplicação utilizando imagens externas

Vamos iniciar desenvolvendo a versão Flash. Inicie o ambiente de desenvolvimento, criando um novo aplicativo Action Script 3.0. Configure o tamanho e o fundo do quadro.

As imagens utilizadas neste trabalho estão disponíveis no diretório *GERAL.imagensdiversas*, que por sua vez encontra-se no material disponibilizado em conjunto com o documento.

Começaremos pela bola de tênis. Acione o menu *File*, item de menu *Import* e por fim a opção *Import to Stage*. Uma janela de seleção de arquivo padrão será apresentada. Localize o diretório citado no parágrafo anterior e selecione o arquivo *bola.png*. Clique em *Abrir*. Note que a bola foi colocada no quadro, no canto superior esquerdo. Para corrigir seu posicionamento, selecione o objeto e altere as propriedades *X* e *Y* para *140* e *90*, respectivamente. A Figura 12 mostra o que obtivemos até o momento.

Agora repita o processo, selecionando a imagem *tabua.png*. Posicione-a no ponto *(10, 80)*. Falta apenas a segunda tábua, porém, ao contrário do que podemos imaginar, o processo não deve ser repetido para a segunda tábua. Antes de colocarmos o terceiro e último objeto no quadro, conheceremos a biblioteca de recursos externos disponibilizada pelo Flash.

Paralelamente a aba *Properties*, existe uma aba chamada *Library*. Acione-a, deverá ser apresentada uma tela semelhante a Figura 13.

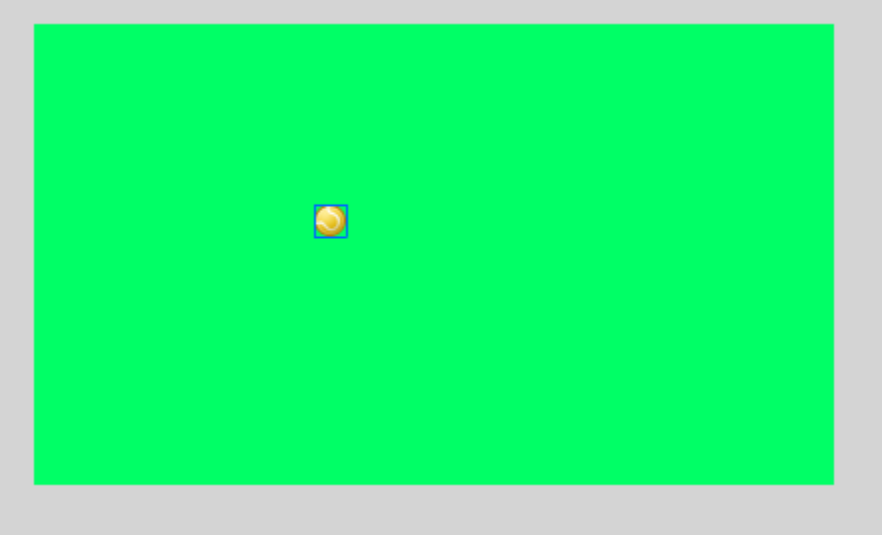

Figura 12 Plotando uma bola de tênis com Flash

Observe que as duas imagens que importamos estão listadas na biblioteca. Para utilizar outras instâncias da mesma imagem, ao invés de importa-las novamente, podemos simplesmente arrastá-las com o mouse, da biblioteca ao quadro. Faça esse processo para plotar a segunda tábua no quadro, posicionando-a no ponto *(360, 20)*. Para detalhes sobre a aba *Library*, consulte Chun (2010, p. 14).

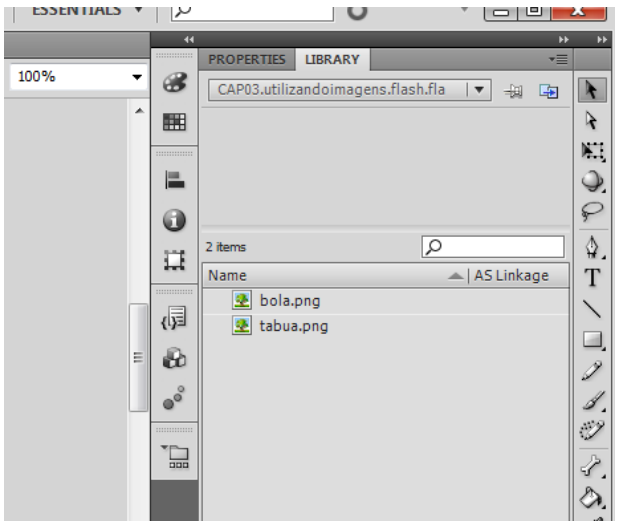

Figura 13 Biblioteca de recursos externos do Flash

Salve a aplicação, execute a rotina de publicação (não se esqueça de corrigir o fundo da página) e abra a página resultante no navegador. Se tudo foi feito corretamente, a aplicação deve estar em conformidade com o apresentado na Figura 11.

Antes de iniciar a versão em Canvas, vale observar uma das principais características do HTML5, que é o reaproveitamento das funcionalidades já oferecidas pelo navegador. Vimos isso, por exemplo, nas propriedades de estilo utilizadas para as fontes e elementos geométricos. A semelhança das estruturas apresentadas com atributos CSS não é mera coincidência, o Canvas foi desenhado para aproveitar outras estruturas, como o CSS e o elemento de imagem do Document Object Model (DOM), cujos benefícios exploraremos agora.

Copie uma versão do modelo de aplicação Canvas. Preencha o método `renderizaTela` com o código apresentado no Código-fonte 9. Execute a aplicação, o fundo do quadro deve ter sido pintado com uma tonalidade clara de verde.

O método que utilizaremos para plotar nossas imagens é o `drawImage`, presente no contexto do Canvas. Uma das opções de passagem de parâmetros para esta função é: um objeto do tipo `Image` e as coordenadas `X` e `Y` referentes ao canto superior esquerdo desejado para a plotagem. As coordenadas nós já possuímos, porém, como obter a instância do tipo `Image`?

```
function renderizaTela() {
  contexto.fillStyle = "#00FF66";
  contexto.fillRect(0, 0, canvasLargura, canvasAltura);
}
```

Código-fonte 9 Pintando o fundo do Canvas de verde claro

A forma que utilizaremos neste trabalho é a criação do objeto através de código JavaScript. Adapte a sua aplicação seguindo o Código-fonte 10, adicionando os pontos destacados. Não se esqueça de corrigir o caminho da imagem para refletir o seu ambiente. Para descobrir outros mecanismos de carga de imagem, refira-se a Fulton (2011, p. 124).

Neste ponto é importante observar a natureza assíncrona da utilização de imagens no Canvas. Se tentarmos mover a chamada da função `drawImage` do método `imagemBolaCarregada` para o método `renderizaTela` acabaremos com comportamentos imprevistos na nossa aplicação. Isso ocorre pois no momento em que atribuímos valor para a propriedade `src` do nosso objeto `Image`, o navegador dispara uma nova requisição HTTP para carrega-la, e não espera o término dessa operação para retornar o controle para a próxima linha de código JavaScript. A única forma de garantirmos a disponibilidade do recurso antes de sua utilização é através do evento `load`, que é disparado apenas quando o navegador termina de realizar o trabalho de carga.

```
var canvasAltura = canvas.height;
var contexto = canvas.getContext("2d");

var bolaImagem = new Image();
bolaImagem.addEventListener('load', imagemBolaCarregada, false);
bolaImagem.src = "../GERAL.imagensdiversas/bola.png";

renderizaTela();

function renderizaTela() {
 contexto.fillStyle = "#00FF66";
 contexto.fillRect(0, 0, canvasLargura, canvasAltura);
}

function imagemBolaCarregada() {
 var bolaPosicaoX = 140;
 var bolaPosicaoY = 90;
 contexto.drawImage(bolaImagem, bolaPosicaoX, bolaPosicaoY);
}

function imagemTabuaCarregada() {
```

Código-fonte 10 Plotando a bola de tênis no Canvas

Salve e execute a aplicação, a bola de tênis deve estar aparecendo no local esperado. Agora passaremos para a plotagem das tábuas. O processo é semelhante, a única diferença é a chamada dupla ao método `drawImage` dentro do método `imagemTabuaCarregada`. Adapte o código conforme o Código-fonte 11. Salve e execute novamente, a aplicação deve estar completa.

```
var contexto = canvas.getContext("2d");

var bolaImagem = new Image();
bolaImagem.addEventListener('load', imagemBolaCarregada, false);
bolaImagem.src = "../GERAL.imagensdiversas/bola.png";
```

```
var tabuaImagem = new Image();
tabuaImagem.addEventListener('load', imagemTabuaCarregada, false);
tabuaImagem.src = "../GERAL.imagensdiversas/tabua.png";

renderizaTela();

function renderizaTela() {
 contexto.fillStyle = "#00FF66";
 contexto.fillRect(0, 0, canvasLargura, canvasAltura);
}

function imagemBolaCarregada() {
 var bolaPosicaoX = 140;
 var bolaPosicaoY = 90;
 contexto.drawImage(bolaImagem, bolaPosicaoX, bolaPosicaoY);
}

function imagemTabuaCarregada() {

 var tabua1PosicaoX = 10;
 var tabua1PosicaoY = 80;
 contexto.drawImage(tabuaImagem, tabua1PosicaoX, tabua1PosicaoY);

 var tabua2PosicaoX = 360;
 var tabua2PosicaoY = 20;
 contexto.drawImage(tabuaImagem, tabua2PosicaoX, tabua2PosicaoY);

}
```

Código-fonte 11 Plotando as tábuas no Canvas

2.4. ANIMANDO OBJETOS

Agora que já sabemos como colocar nossos objetos em cena, podemos partir para um conceito mais avançado e, por conseguinte, mais interessante. Como inserir movimento, ou seja, animação, aos nossos objetos?

Para demonstrar esse conceito, utilizaremos a mesma bola de tênis que trabalhamos na seção anterior. Desenvolveremos uma aplicação com as seguintes características:

- Quadro de *400* por *230* pixels, com fundo verde claro;
- Bola de tênis em movimento, iniciando no ponto *(20, 115)* e finalizando no ponto *(360, 115)*;
- Bola de tênis em rotação constante no sentido horário, com duração de 1 segundo por volta completa;
- Toda a animação deverá ocorrer em um intervalo de tempo de 5 segundos.

A Figura 14 ilustra o nosso objetivo para esta seção. A reta vermelha demarca o caminho que a bola deverá percorrer durante a execução da animação.

Conhecida a aplicação, partiremos para a versão em Flash. Antes de colocarmos a mão na massa, faz-se necessário elaborarmos um pouco sobre as duas formas que essa animação pode ser desenvolvida em Flash.

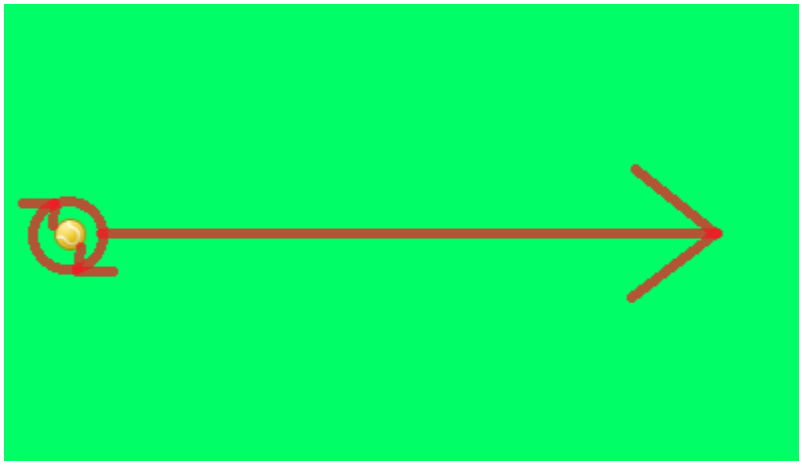

Figura 14 Especificação da animação com a bola de tênis

A primeira forma é a animação sem a necessidade de elaboração de *scripts*. Essa animação baseia-se somente na utilização de *tweens*. Um *tween* consiste de uma série de definições de *keyframes*, que podem definir novas posições, escalas, rotações e até mesmo mutações ponto-a-ponto (também chamados de *morph tweens*) para o objeto animado. Chun (2010, p. 102) explica em detalhes os conceitos que utilizaremos aqui, em especial a linha do tempo (*timeline*) da aplicação Flash.

A segunda forma, mais complexa e mais flexível, é a animação utilizando blocos de scripts. Para animar dessa forma, introduziremos a utilização de *Timers* via Action Script. Codificaremos um temporizador para animar nossa bola a cada fração de milissegundo, simulando exatamente os *frames* da animação via *tween*.

Quando utilizar uma ou outra animação? A resposta é: depende. Dada uma situação que requer uma animação, precisamos analisar se essa animação poderá de alguma forma ser impactada pela interação com o usuário da nossa aplicação, se sim, então utilizamos animação via Action Script, se não, então

ficamos com a animação via *tweens*, que é mais rápida de ser desenvolvida. A Tabela 2 Escolha da estratégia de animação em Flash tenta esclarecer esse conceito com exemplos.

Situação	Tween/ Script	Justificativa
Balançar de folhas de uma árvore, constante durante toda a execução.	Tween	Nenhuma interação com o usuário impacta neste balançar.
Movimentação do personagem principal de um jogo através de eventos de teclado.	Script	O processamento de entradas do usuário é feito através de código.
Mensagem de congratulação devido a uma conquista.	Tween	Por mais que o gatilho da animação seja estritamente dependente de interação com o usuário, uma vez disparada, a animação é constante e não sofre impacto, logo justifica a utilização de um Tween.

Tabela 2 Escolha da estratégia de animação em Flash

Apresentadas as estratégias, desenvolveremos a aplicação especificada em cada uma delas. Primeiro iremos utilizar tweens. Crie uma nova aplicação utilizando o modelo Action Script 3.0. Dimensione e configure o fundo da tela de acordo com nossa especificação no início da seção. Importe a bola de tênis posicionando-a no ponto *(20, 115)*.

Neste ponto precisamos transformar nossa bola em um *Symbol* do tipo *Movie Clip*. Todo objeto que será alvo de animação ou de interação via script deve ser um Symbol. Chun (2010, p. 74) apresenta o conceito em detalhes.

Para transformar nossa bola, clique sobre ela com o botão direito do mouse e acione a opção *Convert to Symbol*. Na janela suspensa, preencha o campo *Name*

com o valor *Bola Rodando*. Confirme, o novo Symbol deve ter sido acrescido na aba *Library*.

Agora que temos nosso Symbol, podemos aplicar um *tween* de posicionamento, para fazer com que ele se desloque para a direita, conforme desejamos. Para isso, precisaremos adicionar *frames* a nossa animação, o que pode ser feito na aba *Timeline*.

A especificação pede que nossa animação dure 5 segundos. No entanto, nossa Timeline trabalha com *frames*. Quantos *frames* devem ser criados para que nossa animação possua 5 segundos? A resposta é dada em função de outro conceito, o atributo *Frames Per Second* (FPS) da nossa aplicação. Essa configuração dita quantos frames serão apresentados ao usuário da aplicação a cada segundo. A Figura 15 mostra onde fica essa propriedade na nossa Timeline. O valor padrão de FPS em uma aplicação Flash é 24.

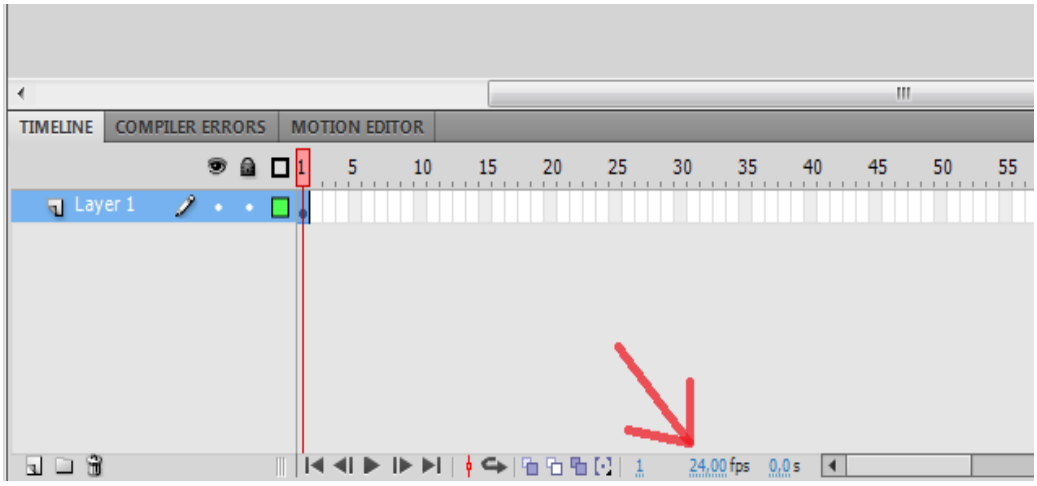

Figura 15 FPS da aplicação Flash

Agora que sabemos quantos frames deve possuir nossa animação, podemos cria-los. Para isso, procure a posição *120* (*5* vezes *24* frames, ou seja, *5* segundos de animação), clique com o botão direito no quadrado diretamente à direita da posição e acione a opção *Create Frame*. Repare que foi criada uma faixa de tonalidade cinza até o *frame 120*.

Faremos agora com que nossa bola de tênis se movimente até o outro lado do quadro. Para isso utilizaremos um *Motion Tween*. Clique com o botão direito

em qualquer ponto da faixa de *frames 0-120* e acione a opção *Create Motion Tween.* A faixa de *frames* deve estar sendo apresentada com uma tonalidade de azul.

Toda animação em Flash utilizando tweens baseia-se na definição de *keyframes*. Um *keyframe* é um *frame* especial, utilizado para mostrar ao Flash como nós, os desenvolvedores, desejamos que os objetos se apresentem em determinado momento em nossa aplicação. Toda transformação entre dois *keyframes* é calculada pelo Flash, respeitando nossas especificações. Clique sobre o *frame* número 120 (cuidado para não selecionar a faixa toda, certifique-se de que apenas o quadrado do último *frame* ficou com aparência de objeto selecionado), clique com o botão direito e acione o submenu *Insert Keyframe*. Dentro deste submenu existem várias opções, cada opção representando um tipo diferente de *tween* que o Flash pode realizar entre nossos *keyframes*. Para essa animação com a bola de tênis utilizaremos um *tween* de posição, logo, selecione a opção *Position* no submenu.

O último passo para finalizarmos nossa animação de movimento é reposicionar a bola de tênis no nosso keyframe de posicionamento. Para isso, certifique-se de que o keyframe 120 está ativo na Timeline (a linha vermelha vertical da aba deve estar passando sobre ele, se não estiver, basta clicar sobre o *frame*), e altere a propriedade *X* da bola para o valor *360*. A Figura 16 demonstra como está nosso espaço de trabalho depois dessas alterações.

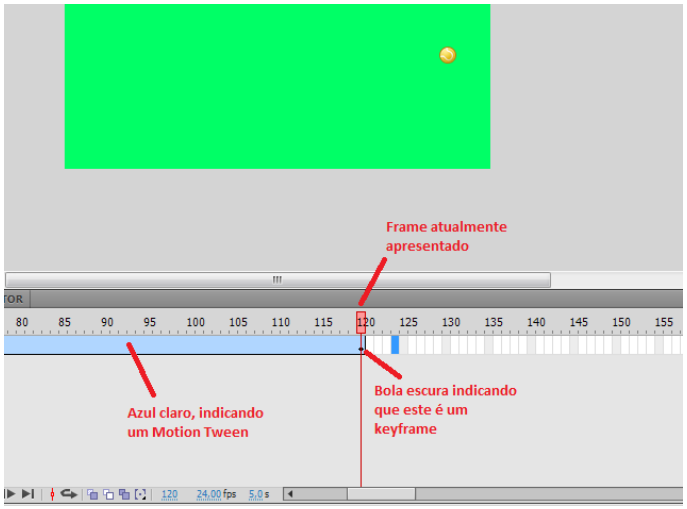

Figura 16 Motion Tween em uma aplicação Flash

Para testar a animação, você pode publicar e abrir o arquivo HTML, ou então, acionar a opção *Test Scene* dentro do menu *Control*. Se tudo foi configurado corretamente a bola deve estar se movendo da esquerda para a direita.

Estamos quase finalizando nossa aplicação, falta apenas à animação de rotação da bola de tênis, em sentido horário. Para desenvolver essa característica precisamos de um novo conceito do Flash, que é a edição dos Movie Clips. No início dessa aplicação nós criamos um Movie Clip a partir da imagem importada da bola de tênis. Agora, para desenvolver a animação de giro, devemos editar a Timeline específica do Movie Clip, e não a principal. Para começar, dê um duplo clique sobre a bola de tênis. A migalha superior da aba da aplicação deve estar indicando que estamos editando o Movie Clip *Bola Rodando*, e não a cena *Scene 1*. Tudo que aplicarmos nessa Timeline será aplicado em todos os objetos criados a partir desse Movie Clip em nossa biblioteca.

Note que a Timeline do Movie Clip está vazia, isso reforça o fato de que não existe conhecimento da cena por parte do Movie Clip. Para continuarmos nosso desenvolvimento, crie uma faixa de *frames* até o frame 24 (lembre-se que em nossa especificação está descrito que a bola deve demorar 1 segundo para cada rotação completa). Clique com o botão direito na faixa de *frames* e selecione a opção *Create Motion Tween*. Uma mensagem deve ter sido apresentada informando que o objeto deve ser convertido para um símbolo antes de poder ser alvo de um *tween*. É por esse motivo que havíamos criado o Movie Clip Bola Rodando, porém, como estamos aplicando um tween dentro da timeline específica deste Movie Clip, temos que criar um novo Movie Clip, uma nova camada de abstração em nossa aplicação. Selecione a imagem da bola, clique com o botão direito sobre a mesma e selecione a opção *Convert To Symbol*. Chame-o de *Bola*. Agora, em nossa Library, temos dois Movie Clips (Bola Rodando e Bola) e um Bitmap (*bola.png*). Tente criar o *tween* novamente, agora deve funcionar.

Depois de criado o Motion Tween, temos duas opções para desenvolver a rotação: via *keyframes* ou via propriedades do *tween*. Como nossa rotação é constante, faremos da segunda forma, aproveitando a propriedade *Rotate*, que serve para indicar quantas vezes o objeto deve ser girado no espaço de tempo

reservado para o *tween*. Clique sobre qualquer *frame* pertencente ao tween em nossa Timeline, depois vá até a aba Properties e modifique o valor da propriedade *Rotate* para *1*. A Figura 17 mostra a aba com a propriedade alterada.

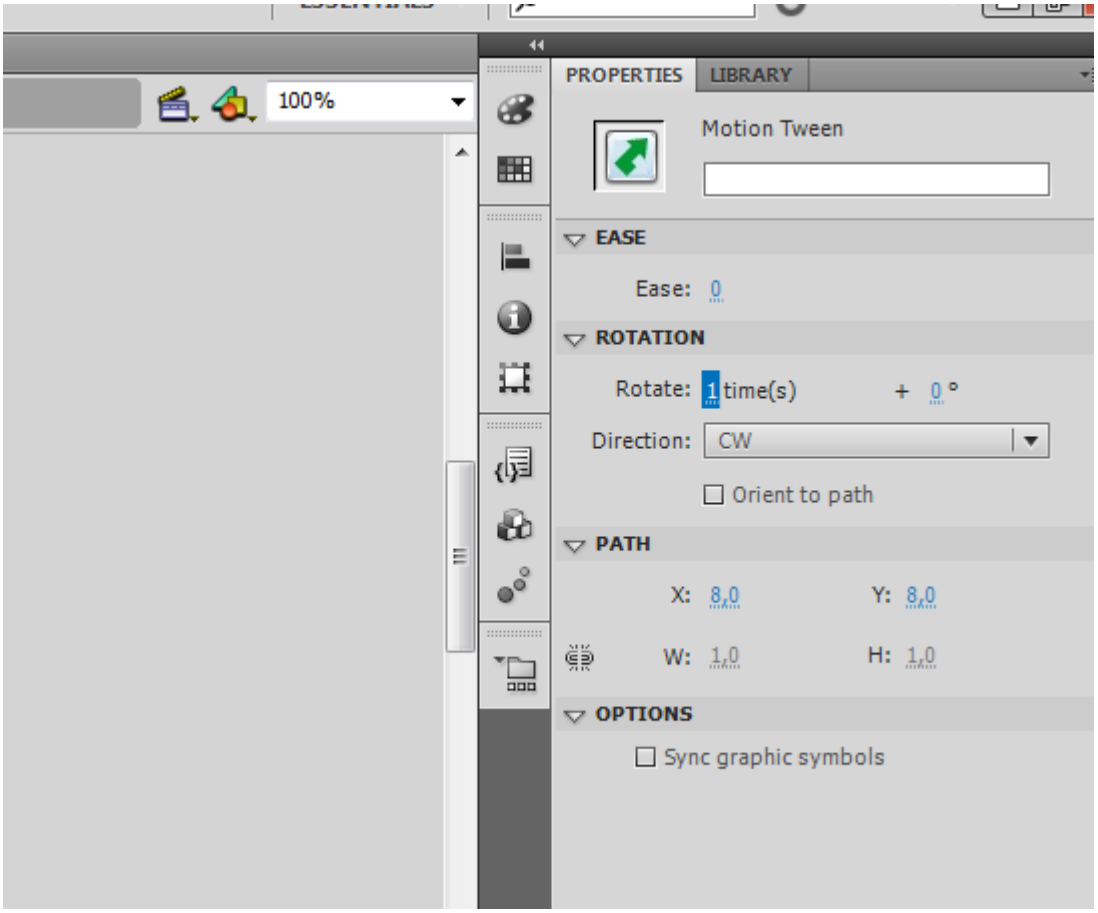

Figura 17 Propriedade Rotate do Motion Tween

Salve a aplicação. Teste-a através da opção Test Scene. Repare que a janela de testes apresenta apenas a animação de rotação que acabamos de desenvolver e não desloca mais a bola da esquerda para a direita. Isso ocorre devido a estarmos dentro do Movie Clip, e não em nossa cena principal. Para resolver basta clicar sobre a etiqueta *Scene 1*, disposta logo atrás da indicação do Movie Clip *Bola Rodando* acima do quadro principal. Após retornar para a cena principal, teste novamente, agora a animação completa deve ser apresentada.

Agora que possuímos uma versão da aplicação utilizando tweens, está na hora de desenvolvermos nossa primeira aplicação em Flash com Action Script. A ideia é controlar a movimentação (e não a rotação) da bola através de código e

temporizadores, ao invés de utilizarmos um tween do tipo posição. O motivo por trás dessa estratégia é que futuramente iremos desejar que determinadas situações (como uma rebatida) faça com que a bola modifique seu curso, o que é facilmente desenvolvido via código, e mais difícil de simular apenas com tweens. Chun (2010, p. 208) discorre sobre Action Script em detalhes.

Crie uma nova aplicação em Flash, execute os mesmos passos realizados na construção da primeira versão do nosso aplicativo animador de bola de tênis, exceto a criação do Motion Tween responsável pela movimentação da bola da direita para a esquerda. O resultado até agora deve ser uma aplicação com as seguintes características:

- Bola parada no ponto *(20, 115)*;
- Bola girando no sentido horário;
- Aba Library contendo dois Symbols (Bola Girando e Bola) e um Bitmap (*bola.png*).

Todo código Action Script é adicionado em um *frame* específico, e é executado assim que a aplicação passa pelo respectivo *frame*. É possível, por exemplo, deixar que a aplicação execute até o *frame* 50, chamar uma função Action Script chamada `stop`, e esperar alguma ação do usuário antes de prosseguir com a execução. Essa ação pode ser um clique em um item de menu, que levará para uma posição específica da Timeline.

No nosso caso não utilizaremos a função `stop`, mas sim criaremos um objeto do tipo `Timer` para que possamos atualizar gradativamente a posição do nosso objeto para a direita, até o ponto desejado.

Para começar, selecione o *frame* número *1* na Timeline. Aperta a tecla de atalho *F9*, a janela suspensa para códigos Action Script será apresentada. Copie o Código-fonte 12 para a caixa de texto.

```
import flash.utils.Timer;
import flash.events.TimerEvent;
import flash.display.MovieClip;

var framesPorSegundo = 24;
var delayTimerBolaMillis = 1000 / framesPorSegundo;
var segundosAnimacao = 5;
```

```
var repeticoesTimerBola = 1000 / delayTimerBolaMillis *
segundosAnimacao;
var timerBola = new Timer(
        delayTimerBolaMillis,
        repeticoesTimerBola);

timerBola.addEventListener(TimerEvent.TIMER, timerBola_onTimer);
timerBola.start();

function timerBola_onTimer(evento: TimerEvent) {

 var xInicial = 20;
 var xDestino = 360;
 var xPercorrido = xDestino - xInicial;
 var pixelsPorTick = xPercorrido / repeticoesTimerBola;

 var bola = MovieClip(this.bola);
 var xNovo = bola.x + pixelsPorTick;
 if (xNovo <= xDestino) {
       bola.x = xNovo;
 } else {
       bola.x = xInicial;
 }
}
```

Código-fonte 12 Timer em Action Script simulando Motion Tween

Sempre que alterar algum trecho de código, verifique a validade léxica e sintática da alteração através da opção *Check Syntax* presente na janela de código. A Figura 18 mostra a localização dessa funcionalidade.

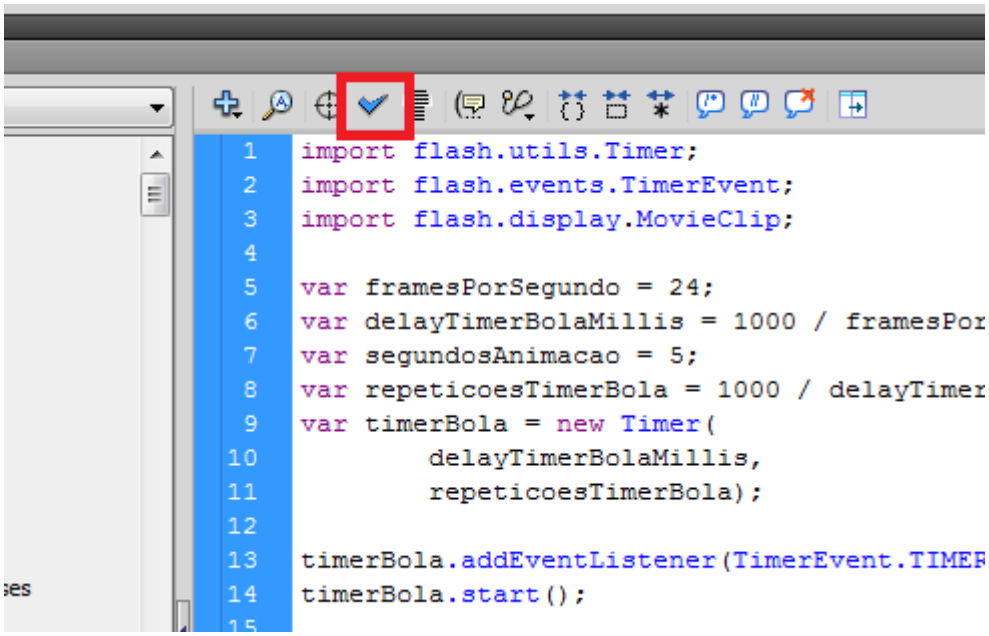

Figura 18 Botão para validar sintaxe do Action Script

Antes de testar o nosso código, temos que efetuar um último ajuste em nossa cena. Note que o código fonte se refere a uma variável `bola` dentro do objeto referenciado pela palavra-chave reservada `this`. Sempre que uma construção desse tipo for processada pelo Flash, o mesmo procurará algum Symbol dentro do objeto dono da Timeline onde o código está escrito (neste caso a cena Scene 1) com o nome solicitado depois do ponto. Esse nome é uma propriedade dos Symbols, *Instance Name*, a primeira da aba Properties dos mesmos. Mude-a para o valor *bola* no nosso objeto. A Figura 19 mostra como isso deve ser feito.

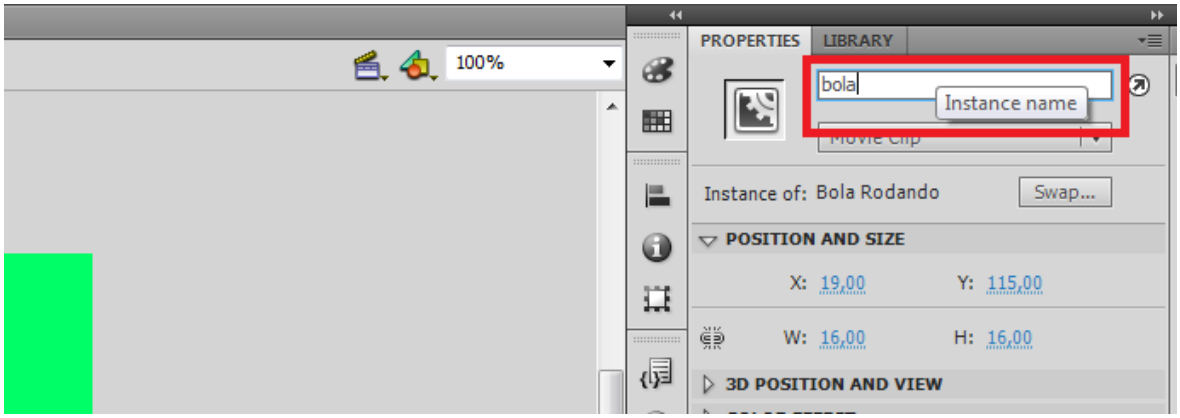

Figura 19 Configurando o nome de instância de um Symbol

Teste a aplicação através da já conhecida funcionalidade Test Scene. Repare que o resultado é o mesmo da aplicação anterior.

Conhecidas as técnicas de animação do Flash, podemos avançar para o desenvolvimento da versão utilizando Canvas. Repare que com Canvas não temos um ferramental visual (ainda) e, portanto, não existe o conceito de *tween* da forma como vimos no Flash. Ficaremos com uma única versão, que como poderá observar nos próximos parágrafos, é muito semelhante à apresentada com Action Script. Fulton (2011, p. 130) possui uma ótima introdução a animações baseadas em Sprites (as quais nós utilizaremos neste trabalho) e conceitos como Timer Loop (ou Main Loop).

Crie uma cópia do modelo de aplicação elaborado no início deste capítulo. Copie e cole o Código-fonte 13 logo abaixo da definição da variável `contexto`.

Esse bloco de código-fonte define uma série de constantes que utilizaremos no decorrer do desenvolvimento.

```
var framesPorSegundo = 24;
var intervaloEntreFrames = 1000 / framesPorSegundo;

var bola = {
 x: 20
 , y: 115
 , anguloEmGraus: 0
};

var bolaAnimacaoSegundos = 5;
var bolaAnimacaoTicks = 1000 / intervaloEntreFrames *
bolaAnimacaoSegundos;
var bolaAnimacaoXInicial = bola.x;
var bolaAnimacaoXDestino = 360;
var bolaAnimacaoXPercorrido = bolaAnimacaoXDestino -
bolaAnimacaoXInicial;
var bolaAnimacaoPixelsPorTick = bolaAnimacaoXPercorrido /
bolaAnimacaoTicks;

var bolaRotacaoSegundos = 1;
var bolaRotacaoTicks = 1000 / intervaloEntreFrames *
bolaRotacaoSegundos;
var bolaRotacaoGrausPorTick = 360 / bolaRotacaoTicks;
```

Código-fonte 13 Constantes utilizadas na aplicação Canvas de animação

Repare na variável `bola`, que definimos como um novo objeto JavaScript com três atributos. É boa prática possuir uma camada de interface (modelo) entre o nosso código de controle (que atualiza o modelo) e o código de visão (que preenche o quadro do Canvas). Na nossa aplicação, a camada de modelo é representada pelo objeto `bola`, a camada de controle será representada por uma função chamada `atualizaModelo` e a camada de visão será a já conhecida função `renderizaTela`.

Agora que já temos definido nosso modelo, podemos continuar com o desenvolvimento, carregando os recursos externos a aplicação. Neste caso temos apenas a imagem *bola.png* como recurso externo a carregar. Para realizar essa carga, adicione o Código-fonte 14 no lugar da chamada da função `renderizaTela`.

```
var bolaImagem = new Image();
bolaImagem.addEventListener('load', imagemBolaCarregada, false);
bolaImagem.src = "../GERAL.imagensdiversas/bola.png";

function imagemBolaCarregada() {
```

```
  setInterval(loopPrincipal, intervaloEntreFrames);
}

function loopPrincipal() {
  atualizaModelo();
  renderizaTela();
}

function atualizaModelo() {
}

function renderizaTela() {
}
```

Código-fonte 14 Carga de recursos e definição do loop principal

Note a chamada da função nativa do JavaScript `setInterval`, que diz para o navegador chamar repetidamente a função `loopPrincipal` a cada `intervaloEntreFrames` milissegundos. A cada iteração (*frame*) da nossa aplicação Canvas, teremos a oportunidade de atualizar o estado da nossa aplicação (dentro do método `atualizaModelo`) e preencher o próximo quadro (função `renderizaTela`).

Começaremos pela atualização do modelo. A ideia é atualizar os atributos `x` e `anguloEmGraus` do nosso objeto `bola` a cada frame, percorrendo a distância necessária e girando o ângulo necessário para concluir a animação nos segundos especificados no início da seção. Quando definimos as constantes da nossa aplicação, entre elas existia uma série de cálculos que definia as variáveis `bolaAnimacaoPixelsPorTick` e `bolaRotacaoGrausPorTick`, esses valores serão utilizados para andar com nosso objeto e girá-lo corretamente a cada *frame*. Atualize o método `atualizaModelo` de acordo com o Código-fonte 15.

```
function atualizaModelo() {

  if (bola.x < bolaAnimacaoXDestino) {
        bola.x = bola.x + bolaAnimacaoPixelsPorTick;
  } else {
        bola.x = bolaAnimacaoXInicial;
  }
  if (bola.anguloEmGraus < 360) {
        bola.anguloEmGraus = bola.anguloEmGraus +
bolaRotacaoGrausPorTick;
  } else {
        bola.anguloEmGraus = 0;
  }

}
```

Código-fonte 15 Atualizando o modelo da aplicação Canvas de animação

Agora que já temos os atributos do modelo atualizados, basta preenchermos o método `renderizaTela`, plotando a imagem da bola na posição e ângulos presentes no nosso objeto de modelo. Para isso, copie o Código-fonte 16 para a aplicação. Salve e execute, o resultado deve ser idêntico ao obtido com as versões em Flash.

```
function renderizaTela() {
 renderizaFundoDaTela();
 renderizaBola();
}

function renderizaFundoDaTela() {
 contexto.setTransform(1, 0, 0, 1, 0, 0);
 contexto.fillStyle = "#00FF66";
 contexto.fillRect(0, 0, canvasLargura, canvasAltura);
}

function renderizaBola() {
 var bolaLargura = bolaImagem.width;
 var bolaAltura = bolaImagem.height;
 var bolaPosicaoX = bola.x;
 var bolaPosicaoY = bola.y;
 var bolaRotacaoEmRadianos = bola.anguloEmGraus * Math.PI / 180;
 contexto.setTransform(1, 0, 0, 1, 0, 0);
 contexto.translate(bolaPosicaoX, bolaPosicaoY);
 contexto.rotate(bolaRotacaoEmRadianos);
 contexto.drawImage(bolaImagem, - bolaLargura / 2, - bolaAltura / 2);
}
```

Código-fonte 16 Plotando a bola de tênis animada em Canvas

Consulte Fulton (2011, p. 41) ou Rowell (2011, p.95) para detalhes referentes aos métodos de transformação do Canvas, como o `setTransform`, o `translate` e o `rotate`.

2.5. INTERAGINDO COM O USUÁRIO

Na seção anterior descobrimos como animar nossos objetos. Já temos conhecimento suficiente para desenvolver grande parte das aplicações dinâmicas para Internet, como por exemplo, banners de propaganda, gráficos, barra superior de um site. Porém, existe outra fatia de aplicações que ainda não conseguimos abordar: as que interagem com o usuário.

Para exemplificar, imagine um jogo (desenvolveremos um completo na próxima seção). É natural identificarmos o jogador executando algum tipo de ação, seja com o mouse ou com o teclado, buscando alcançar um objetivo. Cada

ação disparada pelo jogador é processada pelo jogo que responde alterando o seu espaço universo. Fora do universo dos jogos podemos citar aplicações com navegação por menu ou entrada de dados em formulários.

Nessa seção abordaremos a interação com o usuário. A aplicação que desenvolveremos terá as seguintes características:

- Quadro verde claro, *400* por *230* pixels;
- Tábua plotada inicialmente no ponto *(20, 30)*;
- Ao pressionar a tecla *W*, a tábua deverá andar para cima;
- Ao pressionar a tecla *S*, a tábua deverá andar para baixo;
- Em nenhum momento a tábua poderá estar a menos de *20* pixels de distância da borda do quadro.

A Figura 20 mostra como será o comportamento do nosso aplicativo.

Figura 20 Especificação da interação com o usuário

Vamos iniciar pela versão em Flash. Crie uma nova aplicação, configure o quadro. Importe a imagem da tábua utilizada na seção UTILIZANDO IMAGENS. Transforme-a em um Symbol como fizemos com a bola na seção ANIMANDO OBJETOS, chamando-o de *tabua*. Posicione o objeto (alterando suas propriedades *X* e *Y*) na posição *(20, 30)*.

Antes de passarmos para o Action Script necessário para interagir com o usuário, vale entendermos a estratégia que será utilizada.

A primeira parte da solução será a escuta das teclas pressionadas no teclado. Utilizaremos o método `addEventListener` no objeto `stage`, observando eventos do tipo `KeyboardEvent.KEY_DOWN` e `KeyboardEvent.KEY_UP`. Esses eventos serão utilizados para capturar os acionamentos das teclas *W* e *S* pelo usuário, que por sua vez serão utilizados para atualizar variáveis de modelo chamadas `teclaWPressionada` e `teclaSPressionada`, respectivamente.

A segunda parte da solução consistirá em um `Timer`, que executará *24* vezes por segundo, e a cada execução atualizará a posição da tábua de acordo com as variáveis de modelo.

Chun (2010, p. 214) explica com mais detalhes a criação de eventos em botões. O conceito é exatamente o mesmo para eventos em outros objetos, como em nossa *tabua* ou no *stage* como um todo.

Agora que já sabemos o caminho, podemos codificar essas ideias. Clique sobre o único *frame* de nossa aplicação (na Timeline). Aperte a tecla *F9* para acessar o editor de Action Script. Cole o Código-fonte 17 na área de texto. Valide e teste a aplicação.

```
import flash.display.MovieClip;
import flash.events.KeyboardEvent;
import flash.utils.Timer;
import flash.events.TimerEvent;

var teclaWKeyCode = 87;
var teclaSKeyCode = 83;

var tabuaYMinimo = 10;
var tabuaYMaximo = 90;
var tabuaYDeltaPorTick = 2;
var tabua = MovieClip(this.tabua);

var timerTicksPorSegundo = 24;
var timerDelay = 1000 / timerTicksPorSegundo;
var timer = new Timer(timerDelay);

// modelo da aplicação
var teclaWPressionada = false;
var teclaSPressionada = false;

function onStageKeyDown(e:KeyboardEvent) {
 var keyCode = e.keyCode;
 if (keyCode == teclaWKeyCode) {
        teclaWPressionada = true;
 } else if (keyCode == teclaSKeyCode) {
        teclaSPressionada = true;
 }
}
```

```
function onStageKeyUp(e:KeyboardEvent) {
 var keyCode = e.keyCode;
 if (keyCode == teclaWKeyCode) {
        teclaWPressionada = false;
 } else if (keyCode == teclaSKeyCode) {
        teclaSPressionada = false;
 }
}

function onTimerTick(e:TimerEvent) {
 var tabuaYNovo;
 if (teclaWPressionada) {
        tabuaYNovo = tabua.y - tabuaYDeltaPorTick;
        tabua.y = Math.max(tabuaYNovo, tabuaYMinimo);
 }
 if (teclaSPressionada) {
        tabuaYNovo = tabua.y + tabuaYDeltaPorTick;
        tabua.y = Math.min(tabuaYNovo, tabuaYMaximo);
 }
}

stage.addEventListener(KeyboardEvent.KEY_DOWN, onStageKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onStageKeyUp);

timer.addEventListener(TimerEvent.TIMER, onTimerTick);
timer.start();
```

Código-fonte 17 Action Script para interação com o usuário

Terminada a versão em Flash, podemos passar para o Canvas. Como poderemos observar nos próximos parágrafos, a estratégia de desenvolvimento é a mesma que a utilizada em Flash, o que muda são os objetos que o ambiente (Flash/Canvas) nos provê para executar determinada ação.

Por exemplo, utilizamos o método `addEventListener` no objeto `stage` para adicionar um evento global de escuta no teclado. Para Canvas, a estratégia é a mesma, a diferença é que utilizaremos o método `addEventListener` do objeto `window`. O `Timer` que utilizamos em Flash será desenvolvido com uma chamada ao método `setInterval` do JavaScript, como na seção ANIMANDO OBJETOS. Rowell (2011, p. 158) apresenta técnicas avançadas para manipulação de eventos no Canvas.

Copie o modelo de aplicação desenvolvido na seção OLÁ, MUNDO! para começarmos. Substitua a função `renderizaTela` e a sua chamada pelo Código-fonte 18.

```
var canvasAltura = canvas.height;
var contexto = canvas.getContext("2d");

var teclaWKeyCode = 87;
```

```
var teclaSKeyCode = 83;

var framesPorSegundo = 24;
var intervaloEntreFrames = 1000 / framesPorSegundo;

var tabuaYMinimo = 10;
var tabuaYMaximo = 90;
var tabuaYDeltaPorTick = 2;

var teclasPressionadas = {
 w: false
 , s: false
};

var tabua = {
 x: 20
 , y: 30
};

var tabuaImagem = new Image();
tabuaImagem.addEventListener('load', imagemTabuaCarregada, false);
tabuaImagem.src = "../GERAL.imagensdiversas/tabua.png";

function imagemTabuaCarregada() {
 setInterval(loopPrincipal, intervaloEntreFrames);
}

window.addEventListener("keydown", onKeyDown, true);
window.addEventListener("keyup", onKeyUp, true);

function onKeyDown(e) {
 var keyCode = e.keyCode;
 if (keyCode == teclaWKeyCode) {
        teclasPressionadas.w = true;
 } else if (keyCode == teclaSKeyCode) {
        teclasPressionadas.s = true;
 }
}

function onKeyUp(e) {
 var keyCode = e.keyCode;
 if (keyCode == teclaWKeyCode) {
        teclasPressionadas.w = false;
 } else if (keyCode == teclaSKeyCode) {
        teclasPressionadas.s = false;
 }
}

function loopPrincipal() {
 atualizaModelo();
 renderizaTela();
}

function atualizaModelo() {
 if (teclasPressionadas.w) {
        var tabuaYNovo = tabua.y - tabuaYDeltaPorTick;
        tabua.y = Math.max(tabuaYMinimo, tabuaYNovo);
 }
 if (teclasPressionadas.s) {
        var tabuaYNovo = tabua.y + tabuaYDeltaPorTick;
        tabua.y = Math.min(tabuaYMaximo, tabuaYNovo);
 }
}

function renderizaTela() {
 renderizaFundoDaTela();
 renderizaTabua();
```

```
}

function renderizaFundoDaTela() {
 contexto.setTransform(1, 0, 0, 1, 0, 0);
 contexto.fillStyle = "#00FF66";
 contexto.fillRect(0, 0, canvasLargura, canvasAltura);
}

function renderizaTabua() {
 var tabuaX = tabua.x;
 var tabuaY = tabua.y;
 contexto.drawImage(tabuaImagem, tabuaX, tabuaY);
}
```

Código-fonte 18 Interagindo com o usuário em Canvas

Salve e teste a aplicação, deverá ter obtido o mesmo resultado que em Flash.

2.6. ESTUDO DE CASO: PONG

Na seção anterior nós vimos como capturar ações do usuário e refleti-las em nossos objetos. Nesta seção nós utilizaremos os conceitos abordados até o momento para construir um estudo de caso completo.

O estudo de caso será baseado no famoso jogo Pong, o primeiro videogame com fins lucrativos da história. A Figura 21 mostra a interface do jogo.

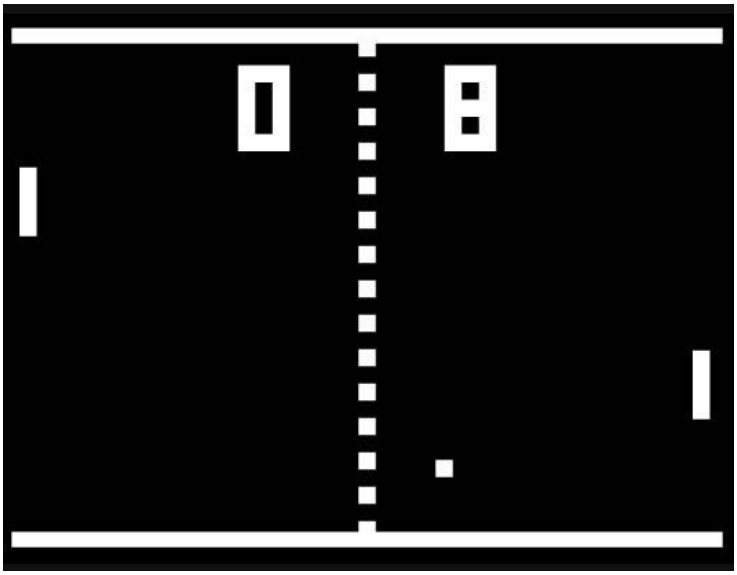

Figura 21 Pong clássico

Como de costume, faremos uma especificação detalhada do nosso objetivo antes de iniciar o desenvolvimento. Nosso Pong deverá:

- Apresentar um placar (utilizando os conceitos da seção OLÁ, MUNDO!);
- Apresentar linha divisória dos campos da esquerda e direita (seção DESENHANDO ELEMENTOS GEOMÉTRICOS);
- Plotar três objetos, duas tábuas (uma de cada lado do campo) e uma bola (seção UTILIZANDO IMAGENS);
- A bola deve se deslocar para a direita e esquerda, e também girar sobre seu eixo no sentido horário (seção ANIMANDO OBJETOS);
- As tábuas devem se deslocar para cima e para baixo de acordo com as ações dos jogadores (teclas W, S, Cima e Baixo, seção INTERAGINDO COM O USUÁRIO).
- A bola deve mudar de direção quando colidir com uma tábua;
- O placar deve ser acrescido caso a bola colida com uma das bordas verticais do quadro;
- A bola deve ser refletida de volta para o campo se colidir com uma das bordas horizontais.

Quando finalizada, nossa aplicação possuirá um aspecto semelhante a Figura 22.

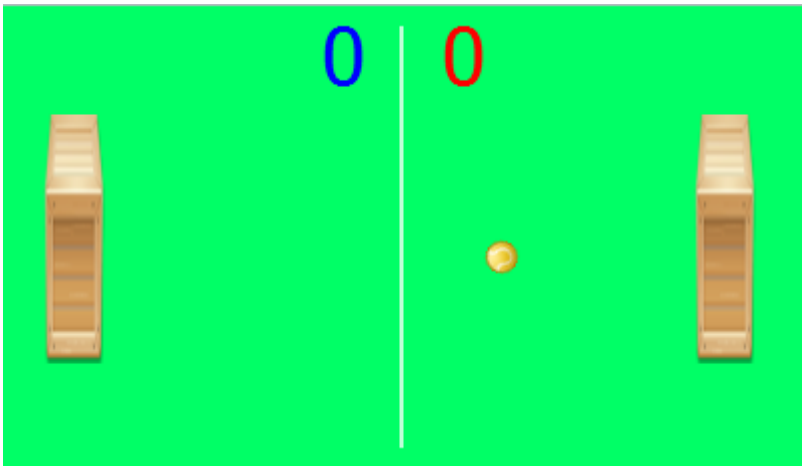

Figura 22 Nossa versão do Pong

Vale observar que nosso intuito aqui não é produzir uma aplicação que apresente um bom nível de jogabilidade, estamos buscando apenas demonstrar a

viabilidade de desenvolvimento deste tipo de aplicação com a tecnologia provida pelo Canvas do HTML5.

Antes de iniciarmos o desenvolvimento, discutiremos sobre um novo conceito que será necessário para o desenvolvimento do nosso Pong, que é a detecção de colisão. Precisamos capturar e processar as colisões da nossa bola com as tábuas e as fronteiras do quadro. Fulton (2011, p. 417) apresenta uma abordagem para o tratamento de colisões. A ideia básica para detectar colisões é a utilização das chamadas *Bounding Boxes*, que consistem na demarcação de polígonos (normalmente retângulos) imaginários ao redor dos nossos objetos, e utilizar algoritmos de cálculo de intersecção entre polígonos para identificar colisões. A Figura 23 exemplifica esse conceito.

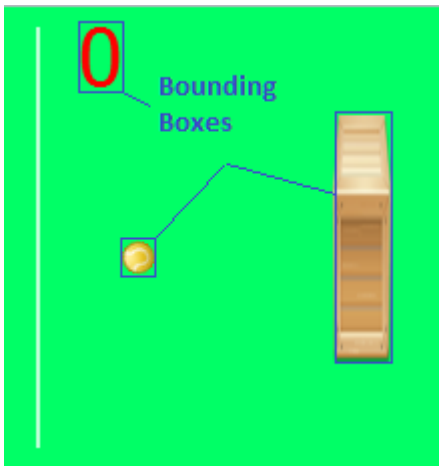

Figura 23 Bounding boxes

Agora podemos começar a desenvolver a versão em Flash. Crie uma nova aplicação utilizando Action Script 3.0. Atualize o fundo, adicione a linha branca no meio do quadro, crie os Symbols das tábuas (chame-os de *tabua1* e *tabua2*) e da bola rolando (chame-o de *bola*). Todas essas ideias já foram abordadas nas seções anteriores.

A primeira novidade se dará na apresentação do placar. Os contadores de pontuação, diferente do nosso texto estático da seção OLÁ, MUNDO!, é dinâmico, e exige um procedimento diferente para seu desenvolvimento.

Acione a ferramenta de texto, vá até a aba *Properties*. Repare que na parte superior da aba existe uma caixa de seleção com a opção *Static Text* selecionada. Essa opção diz ao Flash que esse texto não será alterado no decorrer da execução, permitindo que o motor de publicação execute quaisquer otimizações que achar necessário (como transformar o texto em imagens). Quando precisamos alterar o valor do texto em tempo de execução, essa opção deve ser alterada para *Dynamic Text*, faça isso.

Adicione os campos de pontuação no quadro. Altere a posição, estilo da fonte (utilize a fonte Tahoma) e cor conforme especificado na Figura 22. Troque o nome dos campos de texto para *textoPontuacao1* e *textoPontuacao2*), da mesma forma que fazemos para os Symbols. Agora existe um passo a mais antes de estarmos prontos para dinamizar o conteúdo textual dos objetos.

O Flash, visando suportar portabilidade de fontes, incorpora as fontes utilizadas dentro do arquivo SWF gerado na publicação. Se não o fizesse, as máquinas clientes teriam que possuir as fontes utilizadas na aplicação instaladas localmente. Essa estratégia possui um problema, incorporar muitas fontes acaba elevando o tamanho do arquivo final, dificultando a utilização de usuários com conexões mais lentas. Para contrabalancear esta questão o Flash permite que sejam definidos exatamente os grupos de caracteres (e de quais fontes) que devem ser empacotados junto à aplicação.

Precisamos informar ao Flash a necessidade de empacotamento do conjunto de caracteres numéricos da fonte Tahoma. Para isso, acesse o menu *Text*, item de menu *Font Embedding*. Preencha o campo *Name* com o valor *Números Tahoma*. Escolha no campo *Family* a fonte *Tahoma*. No campo *Character Ranges*, marque a opção *Numerals [0..9]*. Acione a opção +. Pronto, a partir de agora podemos mudar o texto do placar para qualquer composição de caracteres numéricos. A Figura 24 mostra a janela de inclusão de fontes.

Agora que já adicionamos todos os objetos do nosso Pong, falta adicionarmos o código Action Script que dará vida as tábuas e a bola, eventualmente incrementando nosso placar. Comporemos o nosso código com os trechos utilizados nas seções anteriores, e adicionaremos dois novos tratamentos

baseados em colisões: as colisões da bola nas bordas horizontais e nas tábuas, e o acréscimo de pontuação oriundo das colisões com as bordas verticais.

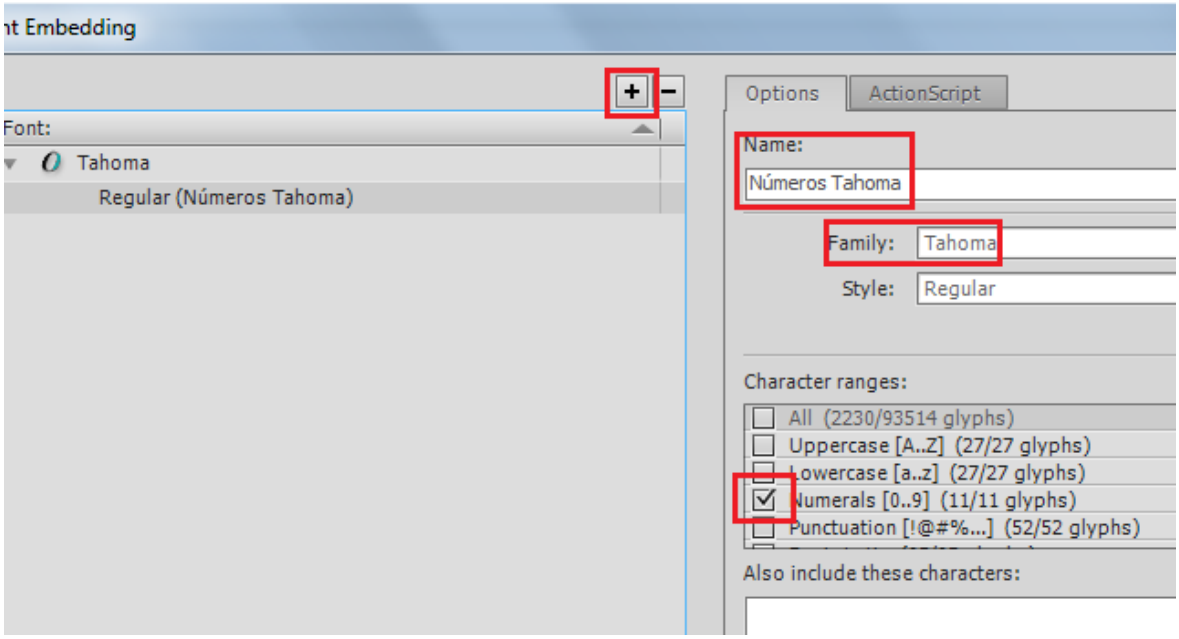

Figura 24 Adicionando fontes dinâmicas na aplicação Flash

Nosso algoritmo de colisão será simples: dado dois retângulos (os *Bounding Boxes*), será verificado se algum dos quatro pontos do primeiro está contido no segundo, caso positivo será considerada uma colisão. Abra a janela de código Action Script do único frame da aplicação e cole o Código-fonte 19.

```
import flash.display.MovieClip;

/* CONSTANTES */
var teclaWKeyCode = 87;
var teclaSKeyCode = 83;
var teclaCimaKeyCode = 38;
var teclaBaixoKeyCode = 40;
var tabuasYMinimo = 10;
var tabuasYMaximo = 90;
var tabuasYDeltaPorTick = 2;
var tabuasTimerTicksPorSegundo = 24;
var tabuasTimerDelay = 1000 / tabuasTimerTicksPorSegundo;
var bolaXInicial = 190;
var bolaYMinimo = 0;
var bolaXMaximo = 400;
var bolaYMaximo = 230;
var bolaVelocidadeX = 3;
var bolaVelocidadeY = 1;
var tabua1 = MovieClip(this.tabua1);
var tabua2 = MovieClip(this.tabua2);

/* MODELO */
```

```
var pontuacaoJogador1 = 0;
var pontuacaoJogador2 = 0;
var teclaWPressionada = false;
var teclaSPressionada = false;
var teclaCimaPressionada = false;
var teclaBaixoPressionada = false;
var bolaVelocidadeAtualX = bolaVelocidadeX;
var bolaVelocidadeAtualY = bolaVelocidadeY;

/* EVENTOS DE TECLADO */

function onStageKeyDown(e:KeyboardEvent) {
 var keyCode = e.keyCode;
 if (keyCode == teclaWKeyCode) {
        teclaWPressionada = true;
 } else if (keyCode == teclaSKeyCode) {
        teclaSPressionada = true;
 } else if (keyCode == teclaCimaKeyCode) {
        teclaCimaPressionada = true;
 } else if (keyCode == teclaBaixoKeyCode) {
        teclaBaixoPressionada = true;
 }
}

function onStageKeyUp(e:KeyboardEvent) {
 var keyCode = e.keyCode;
 if (keyCode == teclaWKeyCode) {
        teclaWPressionada = false;
 } else if (keyCode == teclaSKeyCode) {
        teclaSPressionada = false;
 } else if (keyCode == teclaCimaKeyCode) {
        teclaCimaPressionada = false;
 } else if (keyCode == teclaBaixoKeyCode) {
        teclaBaixoPressionada = false;
 }
}

stage.addEventListener(KeyboardEvent.KEY_DOWN, onStageKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onStageKeyUp);

/* TIMER */

function onTimerTick(e:TimerEvent) {
 atualizaPosicaoTabua(tabua1, teclaWPressionada, teclaSPressionada);
 atualizaPosicaoTabua(tabua2, teclaCimaPressionada,
teclaBaixoPressionada);
 atualizaPosicaoBola();
 processaColisaoBola();
 processaPonto();
}

function atualizaPosicaoTabua(tabua:MovieClip, paraCima:Boolean,
paraBaixo: Boolean) {
 var tabuaYNovo;
 if (paraCima) {
        tabuaYNovo = tabua.y - tabuasYDeltaPorTick;
        tabua.y = Math.max(tabuaYNovo, tabuasYMinimo);
 }
 if (paraBaixo) {
        tabuaYNovo = tabua.y + tabuasYDeltaPorTick;
        tabua.y = Math.min(tabuaYNovo, tabuasYMaximo);
 }
}

function atualizaPosicaoBola() {
 bola.x = bola.x + bolaVelocidadeAtualX;
```

```
 bola.y = bola.y + bolaVelocidadeAtualY;
}

function processaColisaoBola() {
 if (bolaColidiuComTabua(tabua1)) {
        bolaVelocidadeAtualX = bolaVelocidadeX;
 } else if (bolaColidiuComTabua(tabua2)) {
        bolaVelocidadeAtualX = -bolaVelocidadeX;
 }
 if (bola.y <= bolaYMinimo) {
        bolaVelocidadeAtualY = bolaVelocidadeY;
 } else if (bola.y + bola.height >= bolaYMaximo) {
        bolaVelocidadeAtualY = -bolaVelocidadeY;
 }
}

function bolaColidiuComTabua(tabua:MovieClip) {
 return pontoDentroDoRetangulo(bola.x, bola.y, tabua)
        || pontoDentroDoRetangulo(bola.x + bola.width, bola.y, tabua)
        || pontoDentroDoRetangulo(bola.x, bola.y + bola.height, tabua)
        || pontoDentroDoRetangulo(bola.x + bola.width, bola.y +
bola.height, tabua);
}

function pontoDentroDoRetangulo(px:Number, py:Number, tabua:MovieClip)
{
 return (px >= tabua.x && px <= tabua.x + tabua.width)
        && (py >= tabua.y && py <= tabua.y + tabua.height);
}

function processaPonto() {
 if (bola.x <= 0) {
        pontuacaoJogador2++;
        textoPontuacao2.text = pontuacaoJogador2;
        bola.x = bolaXInicial;
 }
 if (bola.x + bola.width >= bolaXMaximo) {
        pontuacaoJogador1++;
        textoPontuacao1.text = pontuacaoJogador1;
        bola.x = bolaXInicial;
 }
}

var timer = new Timer(tabuasTimerDelay);
timer.addEventListener(TimerEvent.TIMER, onTimerTick);
timer.start();
```

Código-fonte 19 Action Script do nosso Pong

Salve e execute a aplicação, deverá ter obtido o resultado especificado no início da seção.

Agora que construímos a versão em Flash, podemos iniciar o desenvolvimento em Canvas. Utilizaremos todos os conceitos vistos até agora, com a adição to teste de colisão dentro da nossa rotina `atualizaModelo`.

Copie um novo modelo de aplicação Canvas, substitua a função `renderizaTela` existente pelo Código-fonte 20, salve e execute. A mesma aplicação será apresentada.

```
/* CONSTANTES */
var framesPorSegundo = 24;
var intervaloEntreFrames = 1000 / framesPorSegundo;
var teclaWKeyCode = 87;
var teclaSKeyCode = 83;
var teclaCimaKeyCode = 38;
var teclaBaixoKeyCode = 40;
var tabuasYMinimo = 10;
var tabuasYMaximo = 90;
var tabuasYDeltaPorTick = 2;
var bolaXInicial = 190;
var bolaYMinimo = 0;
var bolaXMaximo = 400;
var bolaYMaximo = 230;
var bolaVelocidadeX = 3;
var bolaVelocidadeY = 1;
var bolaRotacaoSegundos = 1;
var bolaRotacaoTicks = 1000 / intervaloEntreFrames *
bolaRotacaoSegundos;
var bolaRotacaoGrausPorTick = 360 / bolaRotacaoTicks;

/* MODELO */
var pontuacaoJogador1 = 0;
var pontuacaoJogador2 = 0;
var teclaWPressionada = false;
var teclaSPressionada = false;
var teclaCimaPressionada = false;
var teclaBaixoPressionada = false;
var bolaVelocidadeAtualX = bolaVelocidadeX;
var bolaVelocidadeAtualY = bolaVelocidadeY;
var tabuaImagemCarregada = false;
var bolaImagemCarregada = false;

var tabua1 = { x: 20, y: 50 };
var tabua2 = { x: 345, y: 50 };
var bola = { x: 190, y: 105, anguloEmGraus: 0 };

/* EVENTOS DE TECLADO */

window.addEventListener("keydown", onKeyDown, true);
window.addEventListener("keyup", onKeyUp, true);

function onKeyDown(e) {
 var keyCode = e.keyCode;
 if (keyCode == teclaWKeyCode) {
        teclaWPressionada = true;
 } else if (keyCode == teclaSKeyCode) {
        teclaSPressionada = true;
 } else if (keyCode == teclaCimaKeyCode) {
        teclaCimaPressionada = true;
 } else if (keyCode == teclaBaixoKeyCode) {
        teclaBaixoPressionada = true;
 }
}

function onKeyUp(e) {
 var keyCode = e.keyCode;
 if (keyCode == teclaWKeyCode) {
```

```
        teclaWPressionada = false;
 } else if (keyCode == teclaSKeyCode) {
        teclaSPressionada = false;
 } else if (keyCode == teclaCimaKeyCode) {
        teclaCimaPressionada = false;
 } else if (keyCode == teclaBaixoKeyCode) {
        teclaBaixoPressionada = false;
 }
}

/* CARGA DOS RECURSOS */

var tabuaImagem = new Image();
tabuaImagem.addEventListener('load', imagemTabuaCarregada, false);
tabuaImagem.src = "../GERAL.imagensdiversas/tabua.png";

var bolaImagem = new Image();
bolaImagem.addEventListener('load', imagemBolaCarregada, false);
bolaImagem.src = "../GERAL.imagensdiversas/bola.png";

function imagemTabuaCarregada() {
 tabuaImagemCarregada = true;
 if (tabuaImagemCarregada && bolaImagemCarregada) {
        setInterval(loopPrincipal, intervaloEntreFrames);
 }
}

function imagemBolaCarregada() {
 bolaImagemCarregada = true;
 if (tabuaImagemCarregada && bolaImagemCarregada) {
        setInterval(loopPrincipal, intervaloEntreFrames);
 }
}

/* LOOP PRINCIPAL */

function loopPrincipal() {
 atualizaModelo();
 renderizaTela();
}

/* ATUALIZAÇÃO DE MODELO */

function atualizaModelo() {
 atualizaPosicaoTabua(tabua1, teclaWPressionada, teclaSPressionada);
 atualizaPosicaoTabua(tabua2, teclaCimaPressionada,
teclaBaixoPressionada);
 atualizaPosicaoBola();
 atualizaRotacaoBola();
 processaColisaoBola();
 processaPonto();
}

function atualizaRotacaoBola() {
 if (bola.anguloEmGraus < 360) {
        bola.anguloEmGraus = bola.anguloEmGraus +
bolaRotacaoGrausPorTick;
 } else {
        bola.anguloEmGraus = 0;
 }
}

function processaColisaoBola() {
 if (bolaColidiuComTabua(tabua1)) {
        bolaVelocidadeAtualX = bolaVelocidadeX;
 } else if (bolaColidiuComTabua(tabua2)) {
        bolaVelocidadeAtualX = -bolaVelocidadeX;
```

```
  }
  if (bola.y <= bolaYMinimo) {
        bolaVelocidadeAtualY = bolaVelocidadeY;
  } else if (bola.y + bolaImagem.height >= bolaYMaximo) {
        bolaVelocidadeAtualY = -bolaVelocidadeY;
  }
}

function bolaColidiuComTabua(tabua) {
  return pontoDentroDoRetangulo(bola.x, bola.y, tabua)
        || pontoDentroDoRetangulo(bola.x + bolaImagem.width, bola.y,
tabua)
        || pontoDentroDoRetangulo(bola.x, bola.y + bolaImagem.height,
tabua)
        || pontoDentroDoRetangulo(bola.x + bolaImagem.width, bola.y +
bolaImagem.height, tabua);
}

function pontoDentroDoRetangulo(px, py, tabua) {
  return (px >= tabua.x && px <= tabua.x + tabuaImagem.width)
        && (py >= tabua.y && py <= tabua.y + tabuaImagem.height);
}

function processaPonto() {
  if (bola.x <= 0) {
        pontuacaoJogador2++;
        bola.x = bolaXInicial;
  }
  if (bola.x + bolaImagem.width >= 400) {
        pontuacaoJogador1++;
        bola.x = bolaXInicial;
  }
}

function atualizaPosicaoBola() {
  bola.x = bola.x + bolaVelocidadeAtualX;
  bola.y = bola.y + bolaVelocidadeAtualY;
}

function atualizaPosicaoTabua(tabua, paraCima, paraBaixo) {
  var tabuaYNovo;
  if (paraCima) {
        tabuaYNovo = tabua.y - tabuasYDeltaPorTick;
        tabua.y = Math.max(tabuaYNovo, tabuasYMinimo);
  }
  if (paraBaixo) {
        tabuaYNovo = tabua.y + tabuasYDeltaPorTick;
        tabua.y = Math.min(tabuaYNovo, tabuasYMaximo);
  }
}

/* RENDERIZAÇÃO DA TELA */

function renderizaTela() {
  renderizaFundoDaTela();
  renderizaPontuacao();
  renderizaTabuas();
  renderizaBola();
}

function renderizaFundoDaTela() {

  contexto.setTransform(1, 0, 0, 1, 0, 0);
  contexto.fillStyle = "#00FF66";
  contexto.fillRect(0, 0, canvasLargura, canvasAltura);

  var linhaXInicial = 200;
```

```
 var linhaYInicial = 10;
 var linhaXFinal = 200;
 var linhaYFinal = 220;

 contexto.strokeStyle = "#FFFFFF";
 contexto.beginPath();
 contexto.moveTo(linhaXInicial, linhaYInicial);
 contexto.lineTo(linhaXFinal, linhaYFinal);
 contexto.closePath();
 contexto.stroke();

}

function renderizaPontuacao() {

 var pontuacaoJogador1X = 160;
 var pontuacaoJogador1Y = 40;
 var pontuacaoJogador2X = 220;
 var pontuacaoJogador2Y = 40;

 contexto.setTransform(1, 0, 0, 1, 0, 0);
 contexto.font = "30pt Tahoma";

 contexto.fillStyle = "#00F";
 contexto.fillText(pontuacaoJogador1, pontuacaoJogador1X,
pontuacaoJogador1Y);

 contexto.fillStyle = "#F00";
 contexto.fillText(pontuacaoJogador2, pontuacaoJogador2X,
pontuacaoJogador2Y);

}

function renderizaTabuas() {
 contexto.setTransform(1, 0, 0, 1, 0, 0);
 contexto.drawImage(tabuaImagem, tabua1.x, tabua1.y);
 contexto.drawImage(tabuaImagem, tabua2.x, tabua2.y);
}

function renderizaBola() {
 var bolaRotacaoEmRadianos = bola.anguloEmGraus * Math.PI / 180;
 contexto.setTransform(1, 0, 0, 1, 0, 0);
 contexto.translate(bola.x, bola.y);
 contexto.rotate(bolaRotacaoEmRadianos);
 contexto.drawImage(bolaImagem, - bolaImagem.width / 2, -
bolaImagem.height / 2);
}
```

Código-fonte 20 Pong em Canvas

3. CONSIDERAÇÕES FINAIS

No capítulo anterior nós vimos como desenvolver, em Flash e em Canvas, diversos cenários típicos de aplicações dinâmicas. Utilizaremos essas aplicações como insumo para um trabalho comparativo que será realizado agora, visando descobrir se o Canvas é capaz de substituir o Flash, com base nos seguintes critérios:

- Produtividade;
- Abrangência;
- Facilidade de aprendizado;
- Portabilidade.

Complementando os critérios acima, um *benchmark* de desempenho entre as ferramentas é apresentado no Apêndice A.

Depois de comparar as tecnologias, veremos possíveis pontos de extensão deste trabalho, aspectos que não foram abordados neste e que possuem seu lugar dentro do objetivo proposto.

3.1. A COMPARAÇÃO

Iremos agora comparar as ferramentas nos critérios listados no início do capítulo. Cada subseção irá primeiramente descrever o escopo do critério de forma imparcial, seguindo com parágrafos voltados para as características do Flash e, por último, as características do Canvas que as igualam ou ultrapassam, se houverem.

3.1.1. Produtividade

Este critério busca averiguar se o Canvas permite que as aplicações sejam desenvolvidas com a mesma produtividade quanto com Flash.

O aspecto mais óbvio neste critério é, sem dúvida, a existência de uma ferramenta visual para o Flash. Como vimos desde a seção OLÁ, MUNDO!,

passamos boa parte do tempo de desenvolvimento interagindo com o editor visual Adobe Flash.

Outro ponto importante é a capacidade de reutilizar componentes através de Symbols. Fizemos isso quando criamos dois objetos a partir do mesmo Symbol (tábua) na seção ESTUDO DE CASO: PONG.

Para compor o lado negativo, a possibilidade de desenvolver uma mesma característica de duas formas, como nós vimos na seção ANIMANDO OBJETOS apresenta um problema, pois podemos acabar com código fora do padrão, causando problemas com futuras alterações na aplicação. Imagine se, ao estimar o tempo necessário para mudar o sentido da rotação da bola em função de interação com o usuário, o desenvolvedor considera que a animação está desenvolvida via Action Script, quando ele descobrir que foi feita com Tween terá problemas para desenvolver o solicitado.

Do lado do Canvas, não temos editor visual, o que nos faz começar na última posição. No entanto, se compararmos com a codificação para jogos em C++, veremos que também não existem editores visuais, exceto para a modelagem de objetos 3D. Isso nos leva a crer que a ausência de editor visual (até o momento) é prejudicial apenas para aplicações cujo escopo se restringe a apresentações de imagens, formulários de entrada de dados, navegação via menu, etc. A partir do momento que uma aplicação em Flash precisa de muito código Action Script, a existência de um editor visual pode acabar se tornando prejudicial. De fato, tomamos muito mais tempo para desenvolver a versão em Flash do nosso Pong do que levamos para Canvas.

Quanto à reutilização de componentes, o Canvas possui a vantagem de delegar essa responsabilidade para sua linguagem de scripts. O JavaScript é uma linguagem que permite a simulação de conceitos de orientação a objetos (através de bibliotecas como a Prototype) o que nos deixa no mesmo pé que o Action Script, que também é orientado a objetos. A criação de Symbols em Flash pode ser simulada em Canvas com um bom projeto de Software, como exemplo podemos citar a seção UTILIZANDO IMAGENS, não houve duplicação de código para posicionar as duas tábuas.

Concluindo, no critério Produtividade podemos ver que o Canvas atende ao já obtido com o Flash, ganhando em aplicações mais interativas e intensivas (como jogos) e perdendo nas aplicações menos interativas, como banners animados e menus de sites.

3.1.2. Abrangência

O próximo passo será verificar se o Canvas abrange todo o universo de aplicações abrangidas pelo Flash. Para isso, veremos qual é esse universo e como o Canvas atende cada parte dele.

Através dos capítulos pudemos observar que as aplicações em Flash seguem uma determinada linha de construção: são sempre baseadas em um quadro de tamanho fixo, onde são adicionados vários objetos; existe um eixo de tempo, a Timeline, que permite a adição de animações baseadas em Tweens e em código Action Script; também é possível escutar as ações do usuário e refletir essas ações nos objetos da cena.

Para o Canvas temos o mesmo cenário facilmente simulado através de conceitos de OOP (para demarcar os objetos) e as funções de temporização nativas da linguagem de script. O Canvas vai além, no sentido de facilitar a interação do quadro com o restante da página, visto que tudo é DOM e JavaScript, adicionando um novo nicho de aplicações que com Flash, apesar de possíveis, são mais complicadas, pois envolvem a interação de dois ambientes heterogêneos.

Concluímos que no quesito Abrangência o Canvas supera o Flash.

3.1.3. Facilidade de Aprendizado

Um passo importante antes de desenvolver qualquer aplicação é possuir o pessoal qualificado para o trabalho. Para isso existem duas maneiras, ou se contrata do mercado ou se treina através de programas internos de ensino. Um fator importante na decisão é a facilidade do aprendizado, ou seja, quanto tempo em média será necessário para capacitar o profissional nessa tecnologia? Quais os pré-requisitos necessários para o ensino?

Este quesito é respondido olhando para o objetivo das aplicações em Flash e Canvas. Desenvolvemos uma aplicação dinâmica para disponibilizá-las em uma página HTML, que provavelmente possui outros aspectos dinâmicos mais simples desenvolvidos diretamente em JavaScript, com auxílio de CSS. Independente do conhecimento do profissional nessas tecnologias, o mesmo deve ser exposto para pelo menos dois novos ambientes para desenvolver em Flash: o editor visual Adobe Flash e a sintaxe e API da linguagem Action Script.

Quando falamos em Canvas, a curva de aprendizado cai exponencialmente, visto que o mesmo só precisa aprender a API do Canvas, que é disponibilizada no mesmo JavaScript que ele já está acostumado a utilizar em suas páginas. Sem novos editores e aplicações, o profissional precisará apenas do ferramental que já possui para desenvolvimento Web. De outro ângulo podemos ver o Canvas como uma biblioteca JavaScript, enquanto o Flash seria algo totalmente novo para o profissional.

Desnecessário dizer que o Canvas ultrapassa o Flash neste quesito, sendo muito mais fácil treinar um profissional para desenvolver uma aplicação em Canvas do que para desenvolver em Flash.

3.1.4. Portabilidade

O último critério comparativo é a Portabilidade. Ser portável significa funcionar na maior quantidade possível de ambientes que os usuários podem estar utilizando. No universo das linguagens de programação, por exemplo, podemos dizer que o Java é mais portável que o .Net, visto que o primeiro roda em mais sistemas operacionais que o segundo (que é voltado especificamente para ambientes Microsoft e depende de software não nativo para portá-lo).

Para esse quesito iremos quebrar a comparação em duas categorias: portabilidade no presente e portabilidade futura.

Para a portabilidade no presente, temos que considerar o fato de que o Canvas é uma especificação em andamento, logo, não é aconselhável que seja utilizado para fins não educativos.

Para o futuro, no entanto, o cenário se inverte, pois o Canvas, como visto no critério Facilidade de Aprendizado, é análogo a uma biblioteca do JavaScript que por sua vez vem junto com o navegador, ou seja, qualquer navegador que suporte as especificações do JavaScript e do Canvas (que são elaboradas pela mesma instituição) executarão as aplicações desenvolvidas. Outro ponto importante é a eminente descontinuação do Flash para dispositivos móveis.

3.2. PONTOS DE EXTENSÃO

Este trabalho não contemplou todos os aspectos possíveis de aplicações dinâmicas. Existem vários outros conceitos não explorados que poderiam ser detalhados, enriquecendo a comparação entre as ferramentas.

Um deles é a utilização de áudio nas aplicações. Jogos e players de vídeo são os principais candidatos a utilização dessa característica.

Outro é a análise dos frameworks emergentes para HTML5, como o KineticJS. Esse framework facilita o desenvolvimento de aplicações interativas, poupando o desenvolvedor do trabalho de identificar em que objeto o ponteiro do mouse estava quando o usuário efetuou um clique com o botão esquerdo, por exemplo.

Por último, seria de grande valia a exploração de desenvolvimentos de jogos com o contexto *webgl*, que busca trazer para o navegador o poder dos processadores gráficos da máquina do usuário. Esse contexto poderia viabilizar um mercado para o Canvas até então não explorado na Internet.

3.3. VIABILIDADE DO CANVAS

O Canvas é uma alternativa viável as soluções atuais para desenvolvimento de aplicações dinâmicas e interativas para Internet? Sim, o Canvas é uma alternativa viável. Indo além, em um futuro próximo pode ser que o Canvas seja a única opção a se considerar, quando os desenvolvedores forem obrigados a dar suporte para a crescente fatia de dispositivos móveis dentre os possíveis utilizadores dos nossos sistemas.

4. REFERÊNCIAS

CHUN, R. **Adobe Flash Professional CS5:** Classroom in a Book. Berkeley: Adobe Press, 2010.

FULTON, S.; FULTON, J. **HTML5 Canvas**. Sebastopol: O'Reilly Media, Inc., 2011.

ROWELL, E. **HTML5 Canvas Cookbook**. Birmingham: Packt Publishing Ltd., 2011.

WINOKUR, D. Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5. **Adobe Featured Blogs**, 2011. Disponível em: <http://blogs.adobe.com/conversations/2011/11/flash-focus.html>. Acesso em: 3 nov. 2012.

5. APÊNDICE A – Comparação de Desempenho

Além dos critérios comparativos apresentados no trabalho, o quesito desempenho pode ser explorado, a fim de aumentar a qualidade da comparação. Para tanto, este apêndice apresenta um *benchmark* de desempenho entre as tecnologias comparadas no trabalho, ou seja, entre o Flash e o Canvas.

Antes da apresentação dos valores, é importante observar que, devido à natureza distinta entre as ferramentas (em suas arquiteturas, o Flash utiliza o conceito *retained* enquanto o Canvas utiliza o *immediate*), os resultados do *benchmark* não são tão representativos quanto seriam se as ferramentas tivessem as mesmas origens.

O *benchmark* elaborado consiste em:

- Aplicação com um temporizador configurado para rodar infinitamente;
- A cada execução, o código do temporizador incrementa uma variável numérica e mostra, no centro do quadro, o tempo de execução em milissegundos;
- Assim que a variável numérica atinge o valor de 2^8, a aplicação para de atualizar o tempo de execução.

O teste foi executado em um ambiente com as seguintes características:

- Sistema operacional Windows 7 (64 bits);
- 8 GB de memória RAM;
- Processador Intel i7 2.40GHz;
- Navegador Google Chrome versão 23.

Os resultados (em milissegundos) obtidos estão listados na Tabela 3.

Execução nº	Flash	Canvas
1	4260	1408
2	4267	1422

3	4286	1420
4	4253	1420
5	4284	1411
6	4261	1411
7	4252	1413
8	4260	1423
9	4267	1415
10	4275	1426
Média	**4266,55**	**1416,9**

Tabela 3 Resultados do Benchmark

Conclui-se que o Canvas é cerca de 3 vezes mais rápido que o Flash, nas condições e ambiente utilizados na comparação.

O código fonte das aplicações utilizadas está no conteúdo anexo a este trabalho.