
Modelagem física utilizando HTML 5 e Javascript

Jelther Oliveira Gonçalves – RA 097254
IMECC – UNICAMP

e-mail : jetolgon@gmail.com

Orientador : Prof. Dr. Alberto Saa

Introdução

Sabemos da existência pela internet de “applets” que simulam situações físicas como o sistema massa-mola, pêndulo simples, etc.

Estes “applets” são construídos com base em plugins em Flash ou em Java. Com a difusão de dispositivos móveis, estes plugins estão perdendo espaço para tecnologias de desenvolvimento mais novas como o HTML5 aliado ao velho Javascript.

Este trabalho apresenta de forma sucinta e simplória como estas novas tecnologias de desenvolvimento podem substituir modelagens que até hoje eram realizadas nestes plugins.

Sumário

Introdução	2
O HTML.....	5
A versão 5 do HTML.....	6
A tag <canvas>	7
O Javascript	8
Testando os códigos com o JSFiddle	9
O Oscilador Harmônico Simples.....	11
Criando o modelo no <canvas>.....	15
Inserindo o código CSS.....	17
O código Javascript.....	21
O método setInterval()	21
As variáveis globais e locais	22
A função "redesenhar"	24
Criando o gatilho da animação	27
Testando o código final no JSFiddle	28
O Oscilador Harmônico Amortecido	33
Subamortecimento do oscilador harmônico amortecido.....	35
Amortecimento crítico do oscilador harmônico amortecido	36
Superamortecimento do oscilador harmônico amortecido	37
Aumentando a interatividade do applet	39
O componente input e o tipo “range”	39
Utilizando o mouse para mover o bloco	40
Colocando textos dinâmicos	42
Finalizando o código com os parâmetros	43
O oscilador harmônio amortecido no caso subamortecido com parâmetros	43

O oscilador harmônio amortecido no caso crítico com parâmetros	45
O oscilador harmônio amortecido no caso superamortecido com parâmetros	46
Conclusão	47
Bibliografia.....	48

O HTML

O HTML (criado em 1990) é uma abreviação de **HyperText Markup Language**, ou em português **Linguagem de Marcação de Hipertexto**. Esta linguagem é interpretada pelos navegadores (ou browsers) de internet que utilizamos quase que diariamente.

Esta linguagem faz o uso de “tags”, em português , etiquetas (utilizaremos neste texto o termo em inglês) para criação e formatação dos elementos. As tags são escritas dentro de caracteres de menor e maior (“<” e “>”, conhecidos também como “chevron”) e cada uma possui sua função no documento.

As tags podem ter atributos que modificam suas propriedades originais e valores que explicitam (ou dão conteúdo) as mudanças.

Qualquer documento HTML pode ser editado diretamente a partir de um editor de texto comum, como o Bloco de Notas (WINDOWS) ou o Vim (LINUX/UNIX). Estes softwares não são muito práticos, existindo no mercado alguns que agilizam o trabalho de desenvolvimento.

A seguir apresentaremos algumas tags de exemplo do HTML :

<HTML> - define o início do documento

<head> - define o cabeçalho do documento

<body> - conteúdo principal. É o que o navegador entende que deve ser mostrado.

<script> - define os scripts “Javascript” da página

<style> - aplica os estilos CSS (não abordaremos com profundidade o assunto).

As tags são fechadas com um “/”, por exemplo:

```
<body>Hello World</body>
```

A versão 5 do HTML

A quinta versão do HTML foi lançada em 2008 para suprir uma demanda crescente de desenvolvimento Web. Implementou inúmeras mudanças que antes eram supridas por plugins externos ao desenvolvimento HTML como as tags de vídeo, som, geolocalização, etc.

Neste trabalho utilizaremos principalmente a tag <canvas>, assunto que abordaremos no próximo capítulo.



Figura 1 : O logotipo do HTML5, que pode ser visto em muitos sites recentes da Web.

A tag <canvas>

O <canvas> é um elemento do HTML5 utilizado para renderizar animações utilizando o Javascript incluído no documento HTML.

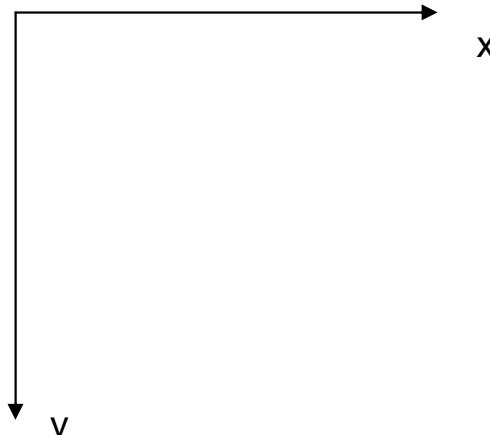
Basicamente, ele depende da resolução do monitor que estiver exibindo e por enquanto só pode desenhar nativamente em 2 dimensões.

O código utilizado para criar o canvas é o seguinte:

```
<canvas id="meuCanvas" style="width : 300px ; height : 300px "> </canvas>
```

No código acima, definimos um “id” para o canvas (utilizado para selecionar este elemento no Javascript) e um estilo para limitar a altura e o comprimento em 300 pixels.

Os eixos coordenados do canvas seguem o sentido :



Como podemos ver, para qualquer tipo de fenômeno nós devemos nos atentar a uma mudança de coordenadas para desenhar no <canvas>.

O Javascript

O Javascript (ou também abreviado como JS) é uma linguagem de programação orientada a objetos implementada em navegadores Web para aprimorar a dinâmica e a interface gráfica dos websites. Foi criada em 1994 e desde então é constantemente aprimorada.

Utilizaremos uma escrita mais simplista neste trabalho, não explorando as capacidades totais da linguagem. Apresentaremos os recursos utilizados a medida que formos usando-os, não aprofundando muito na teoria.

A título de exemplo, abaixo está uma função que dispara uma mensagem :

```
function mensagem() {  
    alert("Olá, isso é um teste!");  
}
```

Já esta outra mostra a soma de 2 números passados via parâmetro para a função:

```
function soma(x,y) {  
    //declara a variável e atribui 0 a ela  
    var resultado = 0;  
  
    //soma os valores passados por parâmetro  
    resultado = x + y;  
  
    //mostra o resultado  
    alert("A soma dos numeros é : " + resultado);  
}
```

Testando os códigos com o JSFiddle

O JSFiddle é uma ferramenta on-line desenvolvida para programadores que desejam ter uma agilidade nos testes de seus códigos.

Podemos acessar o JSFiddle através do endereço <http://jsfiddle.net/>, onde nos é apresentada uma tela com 4 espaços, onde inserimos códigos HTML, Javascript, CSS e vemos o resultado.

Qualquer pessoa pode ao invés de utilizar o JSFiddle, instalar um servidor Apache em seu computador (seja no Windows ou Linux) e rodar os códigos localmente. Preferimos utilizar essa ferramenta on-line pois facilitará que os leitores deste trabalho possam testar os códigos aqui descritos com certa facilidade.

Para quem se interessar, existem soluções prontas como o XAMPP ou o WAMP com executáveis de “one-click-install” para Windows, criando assim um servidor Apache local em seu computador.

Ademais, no JSFiddle é possível utilizar bibliotecas Javascript poderosas como o Jquery, MooTools e Raphael (não aprofundaremos nisso), salvar seus scripts para utilizá-los posteriormente ou compartilhar com outras pessoas.

É recomendável criar uma conta para manter o controle sobre os scripts criados, visto que a cada criação o endereço URL do script não é amigável de ser lembrado.

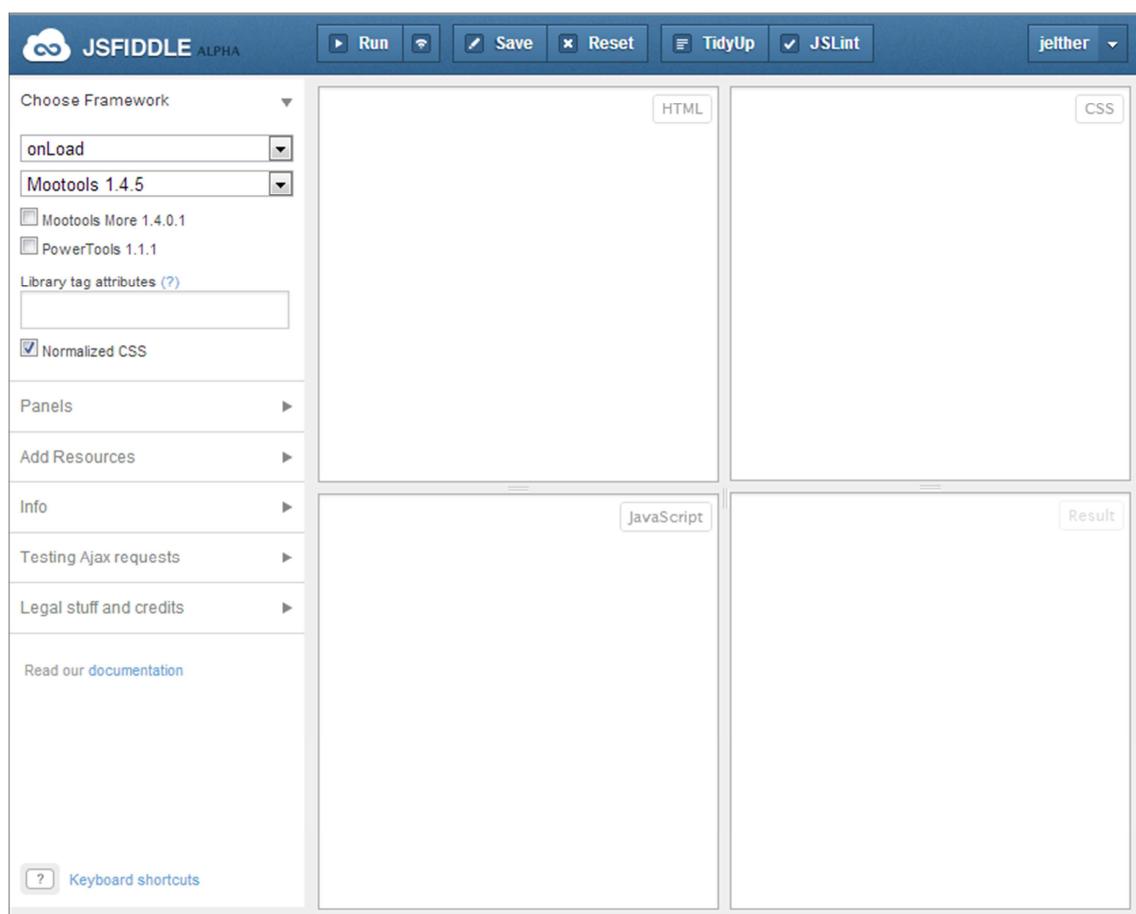


Figura 2 : A interface do JSFiddle

O Oscilador Harmônico Simples

O oscilador harmônico é um sistema que ao ser deslocado de sua posição de equilíbrio, demonstra uma força restauradora proporcional a este descolamento.

Nesta parte utilizaremos um oscilador harmônico simples, isto é, aquele que só apresenta uma força F agindo sobre o sistema.

Esta força F é geralmente explicitada como :

$$F(x) = -kx \quad (1)$$

Onde k é a constante de deformação e x é o deslocamento sofrido pelo bloco.

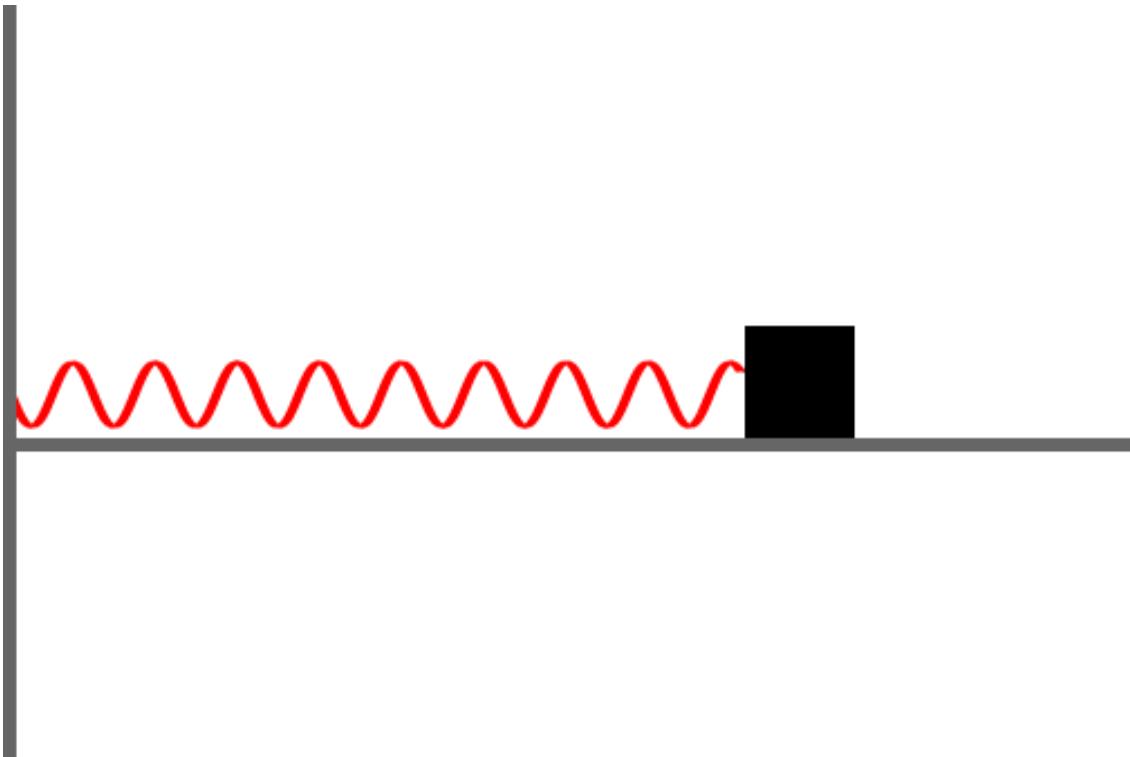


Figura 3 : Um exemplo de sistema massa-mola, onde em preto é o nosso bloco que desliza sobre uma superfície sem atrito (em cinza) ligado a uma parede pela mola vermelha.

Para encontrarmos a equação que descreve o movimento do bloco, vamos utilizar a segunda lei de Newton, onde:

$$F = ma$$

ou

$$F = m\ddot{x}$$

onde os pontos denotam a derivada em relação ao tempo. Utilizando a equação (1) anteriormente, teremos :

$$m\ddot{x} = -kx$$

que resultará em

$$\ddot{x} + \frac{k}{m}x = 0$$

onde chamaremos $\frac{k}{m}$ de ω^2 , ficando com : $\ddot{x} + \omega^2x = 0$.

A solução desta equação diferencial é da forma :

$$x(t) = A \cos(\omega t + \Phi)$$

Onde A é a amplitude do movimento harmônico (em metros), ω é a frequência angular (em rad/s) e Φ é a constante de fase (determinada pelas condições iniciais do movimento e é em radianos).

Vamos considerar uma situação a título de exemplo:

Um bloco de massa de 10 Kg ligado a uma mola onde a constante de deformação possui valor de 100 N/m.

O valor de ω será :

$$\omega^2 = 100 \text{ N/m} / 10 \text{ Kg} \Rightarrow \omega = \sqrt{10} \text{ rad/s}$$

Colocando a amplitude como sendo de $A = 10 \text{ m}$, só nos resta achar o valor da constante de fase Φ , que é calculada a partir das condições iniciais. Neste exemplo faremos com que $x(0) = 0$, isto é, em $t = 0 \text{ s}$ o bloco se encontra na posição inicial. Portanto :

$$10 \cos(\Phi) = 0 \Rightarrow \Phi = \frac{n\pi}{2}, \text{ para } n = 1, 2, 3 \dots$$

Portanto, com $n = 1$, teremos que $\Phi = \frac{\pi}{2}$ e a equação do movimento será :

$$x(t) = 10 \cos(10t + \frac{\pi}{2}) \quad (2)$$

Podemos ver abaixo um gráfico de como essa função se comporta :

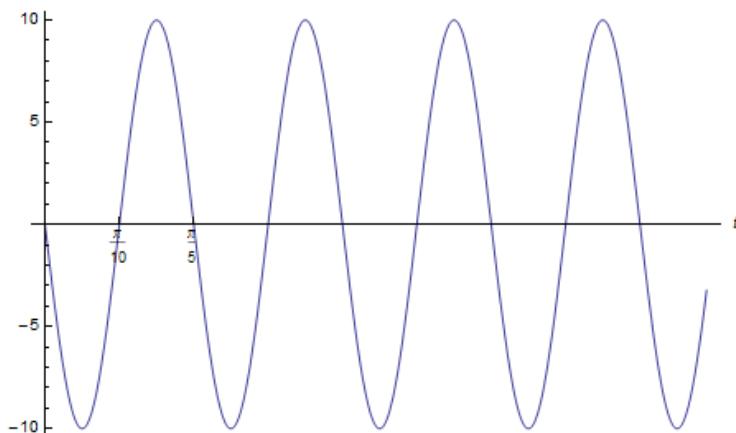


Figura 4 : Gráfico da função (2). Verificamos que existe uma oscilação devido ao cosseno que se repete continuadamente.

Já temos então algumas equações para começar a trabalhar com o desenvolvimento de um código que simule essa situação dentro do HTML5.

Iniciaremos com o desenho somente do movimento em si e posteriormente estudaremos a possibilidade de criar um gráfico que nos mostre a evolução do movimento.

Criando o modelo no <canvas>

Podemos utilizar um bloco de notas ou até mesmo diretamente dentro do JsFiddle para começarmos o nosso trabalho.
Inicialmente escreveremos o código HTML5 necessário para criar o <canvas> :

```
<!--comentários em código html são escritos com essa sintaxe-->
<html>

    <!--começo do cabecalho do documento HTML5-->
    <head>
        <!--define a codificação de caracteres sendo a UTF-8 (UNICODE)-->
        <meta http-equiv="Content-Type" content="text/html" charset="utf-8">

        <!--título do documento-->
        <title>Oscilador Harmônico Simples</title>

    <!--fim do cabecalho do documento HTML5-->
    </head>

    <!--começo do corpo do documento HTML5-->
    <body>

        <!--define o container onde serão armazenados os canvas e cria um canvas com 400px de altura e 800px de largura-->
        <div id="containerCanvas">
            <canvas id="canvasAnimacao" height="400" width="800">Melhor pegar um browser melhor! Use o Chrome ou Firefox!</canvas>
        </div>

        <!--define o container onde ficarão os controles de "Começar" e "Parar"-->
        <div id="containerControles" class="controls">

            <div>
                <input id="start" type="button" name="Start" value="Começar" class="botao"></input>
            </div>

            <div>
                <input id="stop" type="button" name="Start" value="Parar" class="botao"></input>
            </div>

        </div>

    <!--fim do corpo do documento HTML5-->
    </body>
<!--fim do documento HTML5-->
</html>
```

Neste código anterior só definimos o canvas, sem inicializar nenhum código Javascript. Colocando isso no JsFiddle, teremos o resultado abaixo:

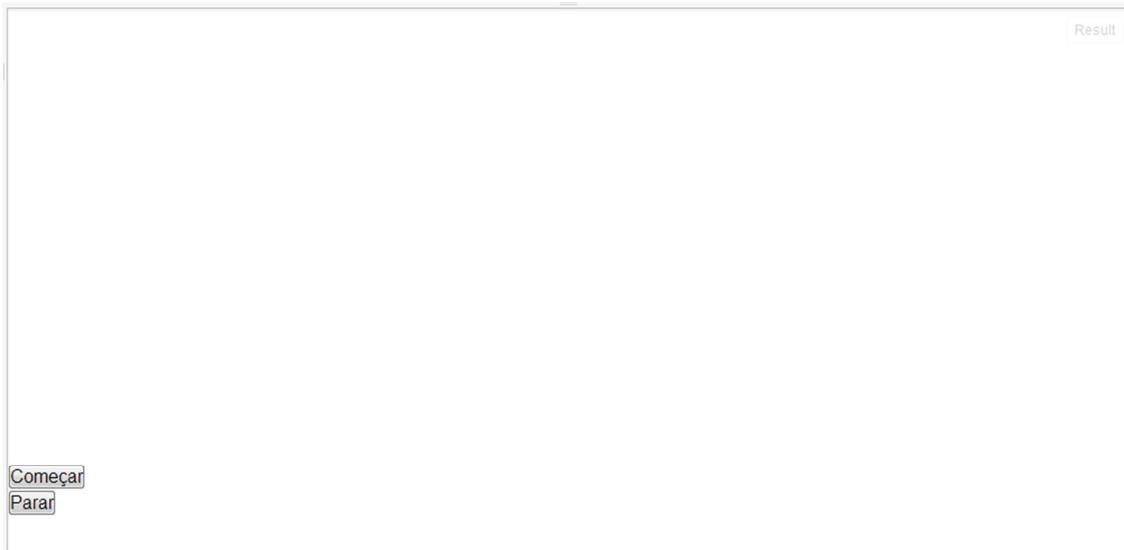


Figura 5 : Tela do resultado do código HTML5.

Veremos, a princípio, que só foram colocados 2 botões : "Começar" e "Parar". Aparentemente não temos nada criado na área branca, mas se inspecionarmos o código através do navegador veremos que o objeto já existe na tela.

Isso se deve ao fato de que precisamos aplicar algumas regras CSS (Cascade Style Sheets) para que possamos ver claramente o canvas.

Inserindo o código CSS

```
canvas {
    display:block;
    margin:10px auto;
    border:1px solid black;
    height:400px;
    width:800px;
}

div {
    float:left;
    margin : 0px auto;
    min-width : 800px;
    padding : 5px;
}

.controls {
    width:400px;
    margin:10px auto;
}

.controls > div {
    border-bottom:1px solid #EEE;
    padding:5px 0;
}

.controls label {
    width:200px;
    display:inline-block;
}

.controls input {
    vertical-align:middle;
}
```

Acima está o código CSS que iremos utilizar. Veja que existe uma sintaxe apropriada para a utilização do mesmo. Adicionamos propriedades aos elementos : canvas, div e as classes nomeadas como : controls e suas heranças (div, label e input).

Com esse código, o resultado fica um pouco mais interessante de se visualizar :

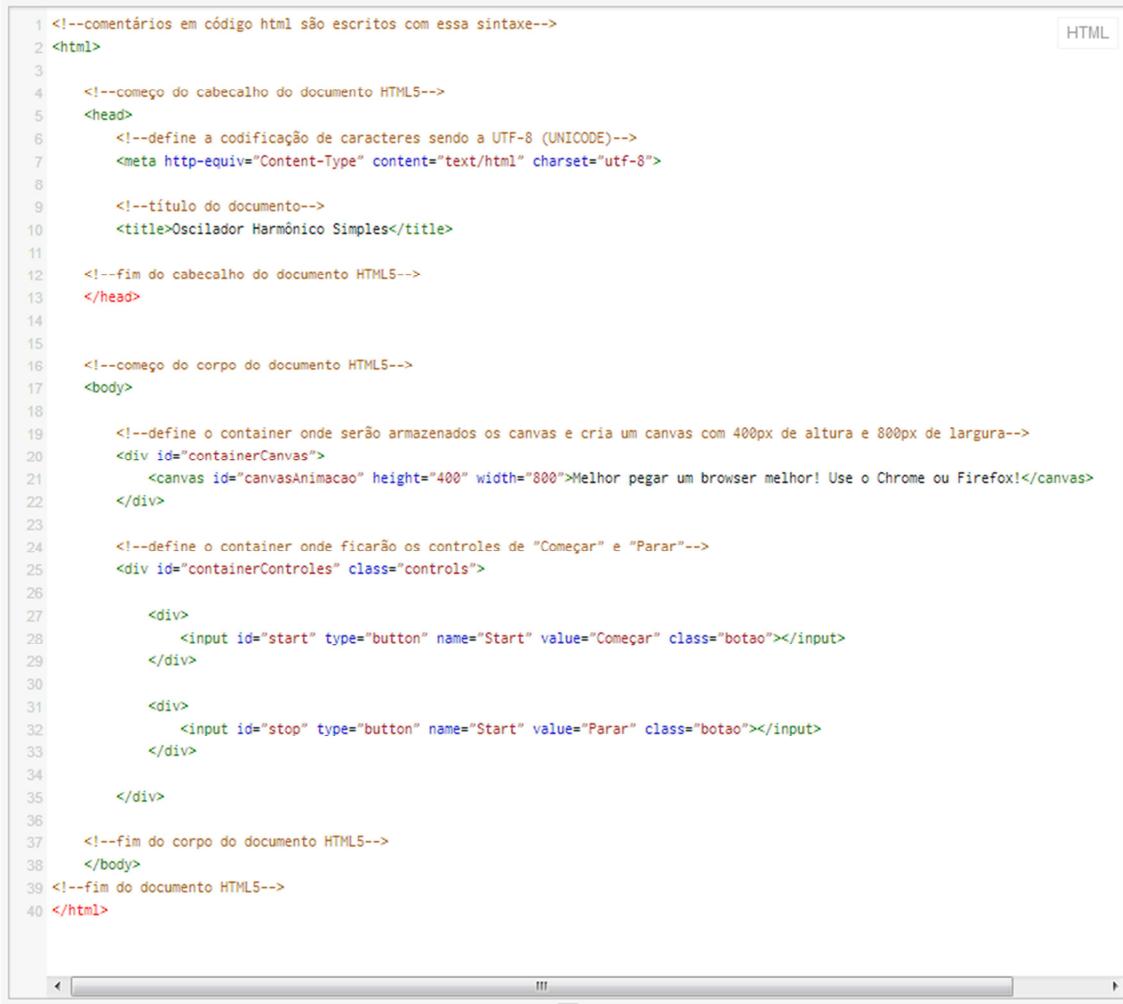


Figura 6 : Tela do resultado do código HTML5 com o CSS.

Podemos ver agora que um quadro com uma borda preta foi realçado do resto dos resultados. Este quadro é o elemento <canvas> com alguns estilos CSS aplicados.

Já estamos prontos para iniciar o desenvolvimento do oscilador, restando basicamente adicionar os códigos em Javascript.

Abaixo algumas telas de como fica cada parte dos quadros dentro do JsFiddle :



```

1 <!--comentários em código html são escritos com essa sintaxe-->
2 <html>
3
4     <!--começo do cabeçalho do documento HTML5-->
5     <head>
6         <!--define a codificação de caracteres sendo a UTF-8 (UNICODE)-->
7         <meta http-equiv="Content-Type" content="text/html" charset="utf-8">
8
9         <!--título do documento-->
10        <title>Oscilador Harmônico Simples</title>
11
12    <!--fim do cabeçalho do documento HTML5-->
13 </head>
14
15
16    <!--começo do corpo do documento HTML5-->
17 <body>
18
19        <!--define o container onde serão armazenados os canvas e cria um canvas com 400px de altura e 800px de largura-->
20        <div id="containerCanvas">
21            <canvas id="canvasAnimacao" height="400" width="800">Melhor pegar um browser melhor! Use o Chrome ou Firefox!</canvas>
22        </div>
23
24        <!--define o container onde ficarão os controles de "Começar" e "Parar"-->
25        <div id="containerControles" class="controls">
26
27            <div>
28                <input id="start" type="button" name="Start" value="Começar" class="botao"></input>
29            </div>
30
31            <div>
32                <input id="stop" type="button" name="Start" value="Parar" class="botao"></input>
33            </div>
34
35        </div>
36
37    <!--fim do corpo do documento HTML5-->
38 </body>
39 <!--fim do documento HTML5-->
40 </html>

```

Figura 7 : O código HTML5 dentro da área específica de HTML no JsFiddle.

```
1 canvas {
2     display:block;
3     margin:10px auto;
4     border:1px solid black;
5     height:400px;
6     width:800px;
7 }
8
9
10 div {
11     float:left;
12     margin : 0px auto;
13     min-width : 800px;
14     padding : 5px;
15 }
16
17 .controls {
18     width:400px;
19     margin:10px auto;
20 }
21
22 .controls > div {
23     border-bottom:1px solid #EEE;
24     padding:5px 0;
25 }
26
27 .controls label {
28     width:200px;
29     display:inline-block;
30 }
31
32 .controls input {
33     vertical-align:middle;
34 }
35 }
```

Figura 8 : O código CSS dentro da área específica de CSS no JsFiddle.

O código Javascript

É dentro do código Javascript que iremos definir o movimento do bloco e desenhá-lo no <canvas> e o primeiro passo é entender o funcionamento do método **setInterval()**.

O método setInterval()

Este método aceita 2 parâmetros, sendo o primeiro alguma função que declaramos e a outra um valor de tempo em milissegundos.

Portanto, poderemos realizar uma chamada do tipo :
setInterval(redesenhar,1000). Estamos basicamente criando um loop que chamará a função “redesenhar” a cada 1000 milissegundos (ou 1 segundo).

Para definir o intervalo de tempo que deveremos chamar a função “desenhar” será a seguinte conta : se quisermos 1 quadro por segundo (ou 1 FPS), deveremos passar o valor de 1000. Se quisermos 10 quadros por segundo, passaremos o valor de 100 (ou seja, $1000/10 = 100$) e assim por diante.

É sabido que para uma visualização fluente, deveremos mostrar 60 quadros por segundo, o que nos leva a $1000/60 = 16,6667$ milissegundos.

Este valor pode variar conforme a necessidade do programador, mas é recomendável não baixar a partir de 30 quadros por segundo, pois o desenho não será tão harmonioso.

Então, o que deve ser feito é chamar uma função que “escreva” em nosso <canvas> em um intervalo de tempo harmonioso.

Então, até agora nosso código será algo do tipo :

```
function comecarDesenho() {  
    setInterval(redesenhar,1000/60);  
}
```

Assim que a função “comecarDesenho” for chamada (por um botão, por exemplo) nosso bloco começará a se movimentar.

As variáveis globais e locais

Para que possamos calcular a posição do bloco, deveremos ter o tempo transcorrido a partir do início do movimento do bloco. Para obtermos isto, criaremos algumas variáveis globais (isto é, podendo ser acessadas de qualquer função) para calcular o tempo transcorrido e outros valores.

Já as variáveis locais podem ser acessadas somente dentro da função criada, por isso o seu nome.

Colocamos o texto "var" em uma variável fora de todas as funções para declarar uma variável global (este texto sendo opcional nesse caso, mas recomendamos para uma melhor visualização do código).

Em variáveis locais, colocamos o texto dentro de alguma função e necessariamente precisamos escrever "var" caso queiramos que ela seja local, do contrário será considerada global.

Utilizaremos outras variáveis globais mais a frente.

```
//=====
//variaveis de controle do tempo
//=====
//tempo transcorrido
var t = 0;
//intervalo de tempo entre os quadros
var intervaloTempo = 0;
//momento inicial do movimento
var primeiroTempo = 0;
//momento atualizado da funcao
var ultimoTempo = 0;
//quantidade de frames transcorridos
var frame = 0;
//armazena o retorno do metodo setInterval (utilizado na parada da execucao do loop
var controleDesenho;

function comecarDesenho() {

    //pega o momento agora
    var date = new Date();

    //atribui para agora o momento inicial do bloco
    primeiroTempo = date.getTime();

    //tambem coloca para o tempo do ultimo frame, somente para iniciar o movimento
    ultimoTempo = primeiroTempo;

    //chama o loop do setInterval e armazena em uma variavel para uma possivel parada
    controleDesenho = setInterval(redesenhar,1000/60);

}
```

Com o código acima, podemos medir o tempo total e a diferença entre os momentos que os quadros são desenhados.

Temos aqui a introdução de uma variável "controleDesenho", utilizada para armazenar o **setInterval()**. Este valor será usado para interrompermos o movimento posteriormente.

De posse desses trechos de código, podemos começar a escrever a função "redesenhar".

A função "redesenhar"

Para desenharmos algo no canvas, utilizaremos a função **document.getElementById("id-do-elemento")**.

Esta função irá retornar o elemento especificado pelo ID (no nosso caso, olhando no código HTML ele será "canvasAnimacao").

Atribuindo esse valor a uma variável, por exemplo "canvas", deveremos chamar o método **getContext("2d")** e armazenar isto em outra variável, que chamaremos de "context".

Este método getContext retorna para a variável "context" os métodos e propriedades que utilizaremos para desenhar no <canvas>.

Portanto, nosso código será :

```
// =====  
// pega o canvas da animacao  
// =====  
canvas = document.getElementById("canvasAnimacao");  
context = canvas.getContext("2d");
```

Estas 2 variáveis (context e canvas) serão declaradas globalmente no script total.

Tendo a variável "context", podemos iniciar o desenho de fato. O método **fillRect()** nos deixa desenhar um retângulo de altura e largura passados por parâmetro da seguinte forma :

```
context.fillRect(xinicial,yinicial,largura,altura);
```

Ou seja, devemos passar uma coordenada (x,y) inicial onde se iniciará o desenho de um retângulo de largura e altura variáveis.

Portanto, para colocar o bloco em "movimento", a cada iteração da função "redesenhar" o valor "xinicial" passado para o método fillRect deverá mudar baseado no tempo.

Contudo, se chamarmos essa função fillRect em toda iteração, o <canvas> gravará por cima do que já foi feito anteriormente.

Para resolver isso, utilizaremos outro método chamado **clearRect()**. Este método possui os seguintes parâmetros :

```
context.clearRect(xinicial,yinicial,largura,altura);
```

Entretanto, ao invés de gravar algum retângulo sólido, ele limpará todo conteúdo existente com base nos parâmetros passados.

Com isso, nossa função "redesenhar" terá a seguinte forma até este ponto :

```
function redesenhar() {  
    // ======  
    //rotinas de ajuste de tempo  
    // ======  
    // incrementa o número de frames decorridos desde o início  
    frame++;  
  
    //pega o tempo no começo de cada iteração  
    var date = new Date();  
    var agoraTempo = date.getTime();  
  
    //calcula a diferença de tempo em relação a última iteração  
    intervaloTempo = agoraTempo - ultimoTempo;  
  
    //incrementa o tempo total com o intervalo decorrido  
    t += intervaloTempo;  
  
    //coloca o tempo atual da iteração como sendo a da última iteração  
    ultimoTempo = agoraTempo;  
    // ======  
  
    // ======  
    // trabalha com o canvas e as variáveis  
    // ======  
    //joga o canvas nas variáveis declaradas globalmente  
    canvas = document.getElementById("canvasAnimacao");  
    context = canvas.getContext("2d");  
  
    //retorna a altura e a largura do nosso elemento <canvas>
```

```
var heightCanvasAnimacao = canvas.height;
var widthCanvasAnimacao = canvas.width;
// =====

// movimento harmonico
// x = A*cos(omega*t + fase)
// neste caso a fase é pi/2 para o bloco oscilar em torno da origem!
bloco.x = bloco.amplitude*Math.cos(bloco.omega*t + (Math.PI/2));

//limpa o desenho com o método clearRect()
context.clearRect(0,0,widthCanvasAnimacao,heightCanvasAnimacao);

//desenha o bloco através do fillRect()
context.fillRect(bloco.x + widthCanvasAnimacao/2, heightCanvasAnimacao -
(heightCanvasAnimacao / 2 + 25), 50, 50);

}
```

A variável "bloco" é um vetor enumerado, que vamos declarar globalmente juntamente com uma variável "mola":

```
//velocidade, posicao em x e massa do bloco
var bloco = {velocidade : 0 , x : 0 , xanterior : 0 , massa : 0 , amplitude : 0 , omega : 0 };

//constante da mola
var mola = {k : 0};
```

Portanto, temos dentro da variável "bloco" e "mola" os valores que vamos usar para calcular a posição final do bloco na função "redesenhar".

As variáveis "canvas", "context" serão declaradas globalmente para que possamos utilizá-las em várias funções ou métodos.

Assim, temos o Javascript funcional, faltando apenas criarmos o disparo da animação através do botão "Começar".

Criando o gatilho da animação

Para deixarmos os 2 botões que criamos no HTML funcionais, utilizaremos uma biblioteca Javascript bastante difundida entre os programadores Web. Esta biblioteca se chama Jquery e facilita muito o desenvolvimento de interfaces interativas.

Não aprofundaremos neste assunto mas nos basta saber que dentro do JSFiddle ela pode ser utilizada diretamente, não sendo necessário criar uma referência externa a algum arquivo.

Para nossos propósitos, utilizaremos o método `$(document).ready()`. Este método garante que a função dentro dele será executada somente quando a página estiver carregada.

Utilizaremos também o método `$("#id-do-elemento").click()`, onde este coloca uma função para ser executada ao clicarmos no elemento especificado entre as aspas.

Portanto, nosso código ficará assim :

```
//quando a página inteira carregar
$(document).ready( function () {

    //associa o começo do desenho ao clique do botão "Start"
    $("#start").click(function (){
        comecarDesenho();
    });

    //para o movimento ao clicar no botão "Parar"
    $("#stop").click(function (){
        clearInterval(controleDesenho);
    });

});
```

Vemos que aqui estamos utilizando o método `clearInterval()` com a variável "controleDesenho" passada via parâmetro. Este método para a função que está sendo executada pelo `setInterval()`.

Portanto, temos todo o código Javascript feito para que o oscilador harmônico seja visível no JSFiddle.

Testando o código final no JSFiddle

Vamos colocar todo o nosso código dentro do JSFiddle para vermos o movimento do oscilador em ação.

O último ajuste necessário é colocar um valor para a massa do bloco, a constante de mola , amplitude do movimento e o ω .

Na função **comecarDesenho()** vamos atribuir os valores da seguinte forma :

```
bloco.massa = 100;
mola.k = 0.01;
bloco.amplitude = 100;
bloco.omega = Math.sqrt(mola.k/bloco.massa);
```

Atentando que todos os valores devem estar no sistema internacional de unidades.

Assim, o código Javascript será :

```
=====
//variaveis de controle do tempo
=====
//tempo transcorrido
var t = 0;
//Intervalo de tempo entre os quadros
var intervaloTempo = 0;
//momento inicial do movimento
var primeiroTempo = 0;
//momento atualizado da funcao
var ultimoTempo = 0;
//quantidade de frames transcorridos
var frame = 0;
//armazena o retorno do metodo setInterval (utilizado na parada da execucao do loop
var controleDesenho;

//velocidade, posicao em x e massa do bloco
var bloco = {velocidade : 0 , x : 0 , xanterior : 0 , massa : 0 , amplitude : 0 , omega : 0 };
//constante da mola
var mola = {k : 0};

function redesenhar() {

=====
//rotinas de ajuste de tempo
=====
// incrementa o numero de frames decorridos desde o inicio
frame++;

//pega o tempo no começo de cada iteração
var date = new Date();
var agoraTempo = date.getTime();

//calcula a diferença de tempo em relação a última iteração
intervaloTempo = agoraTempo - ultimoTempo;

//incrementa o tempo total com o intervalo decorrido
t += intervaloTempo;

//coloca o tempo atual da iteração como sendo a da última iteração
ultimoTempo = agoraTempo;
=====

// trabalha com o canvas e as variáveis
=====

//joga o canvas nas variáveis declaradas globalmente
```

```

canvas = document.getElementById("canvasAnimacao");
context = canvas.getContext("2d");

//retorna a altura e a largura do nosso elemento <canvas>
var heightCanvasAnimacao = canvas.height;
var widthCanvasAnimacao = canvas.width;
=====

// movimento harmonico
// x = A*cos(omega*t + fase)
// neste caso a fase é pi/2 para o bloco oscilar em torno da origem!
bloco.x = bloco.amplitude*Math.cos(bloco.omega*t + (Math.PI/2));

//limpa o desenho com o método clearRect()
context.clearRect(0,0,widthCanvasAnimacao,heightCanvasAnimacao);

//desenha o bloco através do fillRect()
context.fillRect(bloco.x + widthCanvasAnimacao/2, heightCanvasAnimacao - (heightCanvasAnimacao / 2 + 25), 50, 50);

}

function comecarDesenho() {

//pega o momento agora
var date = new Date();

//atribui para agora o momento inicial do bloco
primeiroTempo = date.getTime();

//tambem coloca para o tempo do ultimo frame, somente para iniciar o movimento
ultimoTempo = primeiroTempo;

bloco.massa = 100;
mola.k = 0.01;
bloco.amplitude = 100;
bloco.omega = Math.sqrt(mola.k/bloco.massa);

//chama o loop do setInterval e armazena em uma variável para uma possível parada
controleDesenho = setInterval(redesenhar,1000/60);

}

//quando a página inteira carregar
$(document).ready(function () {

//associa o começo do desenho ao clique do botão "Start"
$("#start").click(function (){
    comecarDesenho();
});

//para o movimento ao clicar no botão "Parar"
$("#stop").click(function (){
    clearInterval(controleDesenho);
});

});
}

```

Devemos usar esse código dentro da região do Javascript no JSFiddle e também selecionar o JQuery nas opções a esquerda.

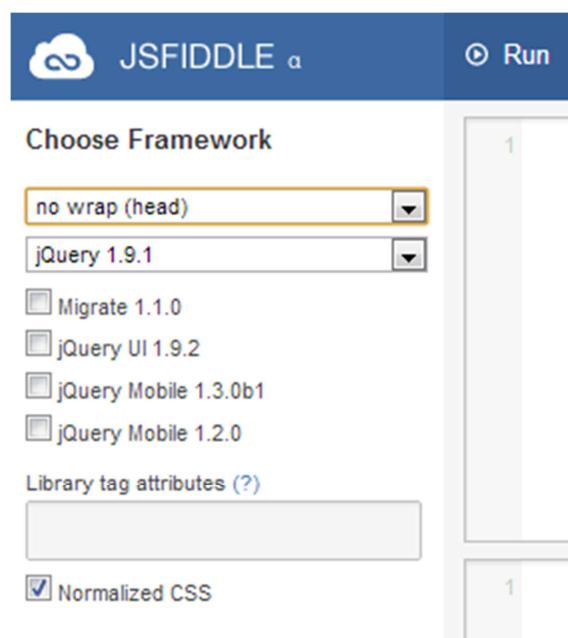
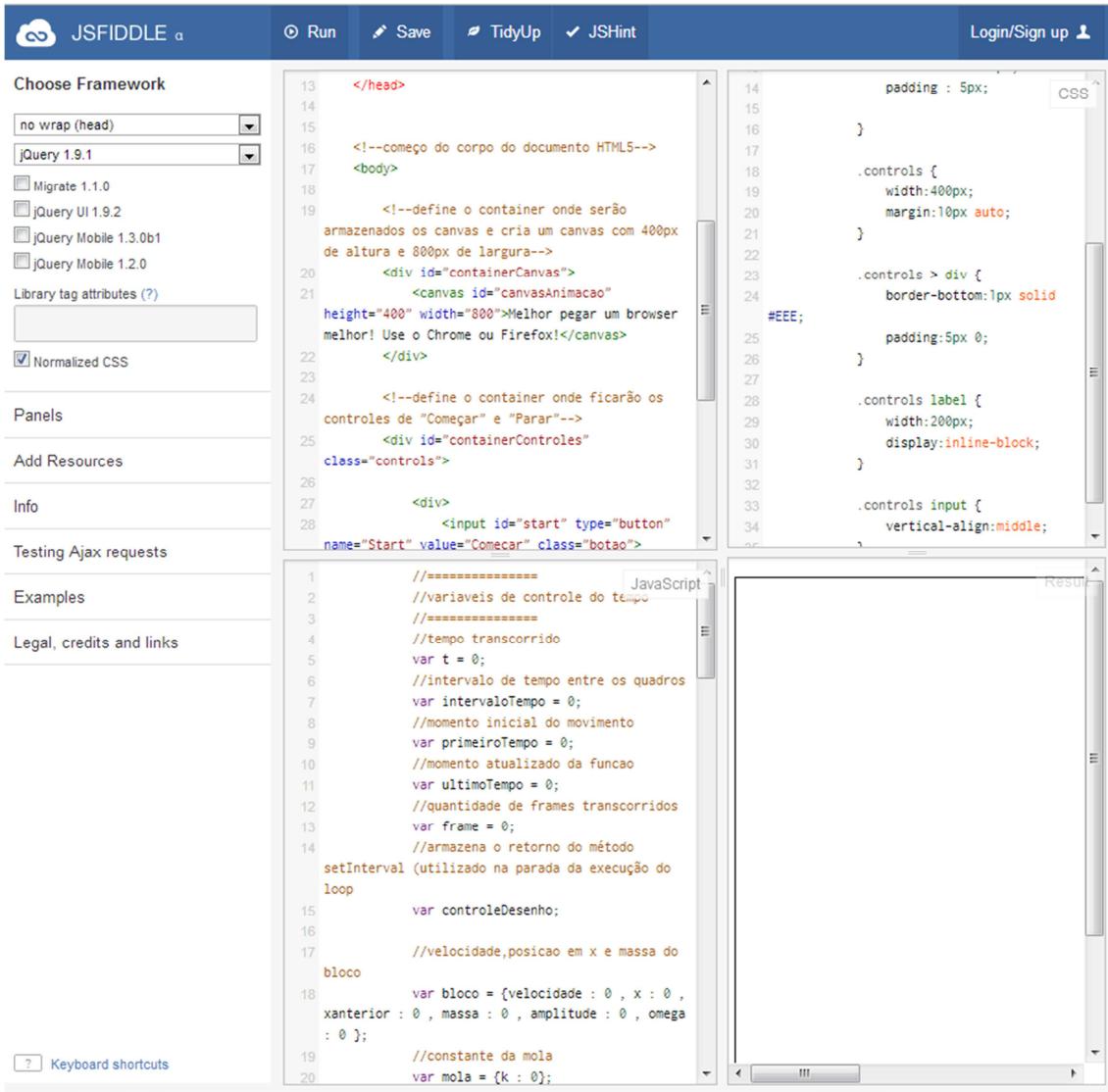


Figura 9 : Escolha pela biblioteca Jquery nas opções do JSFiddle.

Com isso, o código Javascript inserido no JSFiddle fica :



The screenshot shows the JSFiddle interface with the following components:

- Header:** JSFIDDLE, Run, Save, TidyUp, JSHint, Login/Sign up.
- Left Sidebar (Choose Framework):**
 - no wrap (head)
 - jQuery 1.9.1
 - Migrate 1.1.0
 - jQuery UI 1.9.2
 - jQuery Mobile 1.3.0b1
 - jQuery Mobile 1.2.0
 - Library tag attributes (?)
 - Normalized CSS (checked)
- Panels:** Panels, Add Resources, Info, Testing Ajax requests, Examples, Legal, credits and links.
- Code Area:**
 - HTML:**

```
</head>
<!--começo do corpo do documento HTML-->
<body>
    <!--define o container onde serão
        armazenados os canvas e cria um canvas com 400px
        de altura e 800px de largura-->
    <div id="containerCanvas">
        <canvas id="canvasAnimacao"
            height="400" width="800">Melhor pegar um browser
            melhor! Use o Chrome ou Firefox!</canvas>
    </div>
    <!--define o container onde ficarão os
        controles de "Começar" e "Parar"-->
    <div id="containerControles"
        class="controls">
        <div>
            <input id="start" type="button"
                name="Start" value="Comecar" class="botao">
        </div>
    </div>

```
 - CSS:**

```
padding : 5px;
}
.controls {
    width:400px;
    margin:10px auto;
}
.controls > div {
    border-bottom:1px solid #EEE;
    padding:5px 0;
}
.controls label {
    width:200px;
    display:inline-block;
}
.controls input {
    vertical-align:middle;
}
```
 - JavaScript:**

```
=====
//variaveis de controle do tempo
=====
//tempo transcorrido
var t = 0;
//intervalo de tempo entre os quadros
var intervaloTempo = 0;
//momento inicial do movimento
var primeiroTempo = 0;
//momento atualizado da funcao
var ultimoTempo = 0;
//quantidade de frames transcorridos
var frame = 0;
//armazena o retorno do metodo
setInterval (utilizado na parada da execucao do
loop
    var controleDesenho;
    //velocidade, posicao em x e massa do
    bloco
    var bloco = {velocidade : 0 , x : 0 ,
    xanterior : 0 , massa : 0 , amplitude : 0 , omega
    : 0 };
    //constante da mola
    var mola = {k : 0};
```
- Result Area:** Shows a blank canvas area where the animation would be displayed.

Figura 10 : Todos os códigos inseridos no JSFiddle (tela com dimensão reduzida propositalmente).

Com isso, basta utilizarmos a tela no canto inferior direito para iniciar a animação. Pressionando o botão "Começar" um bloco de cor preta vai se movimentar dentro do <canvas> como um oscilador harmônico simples. Abaixo a tela do movimento:

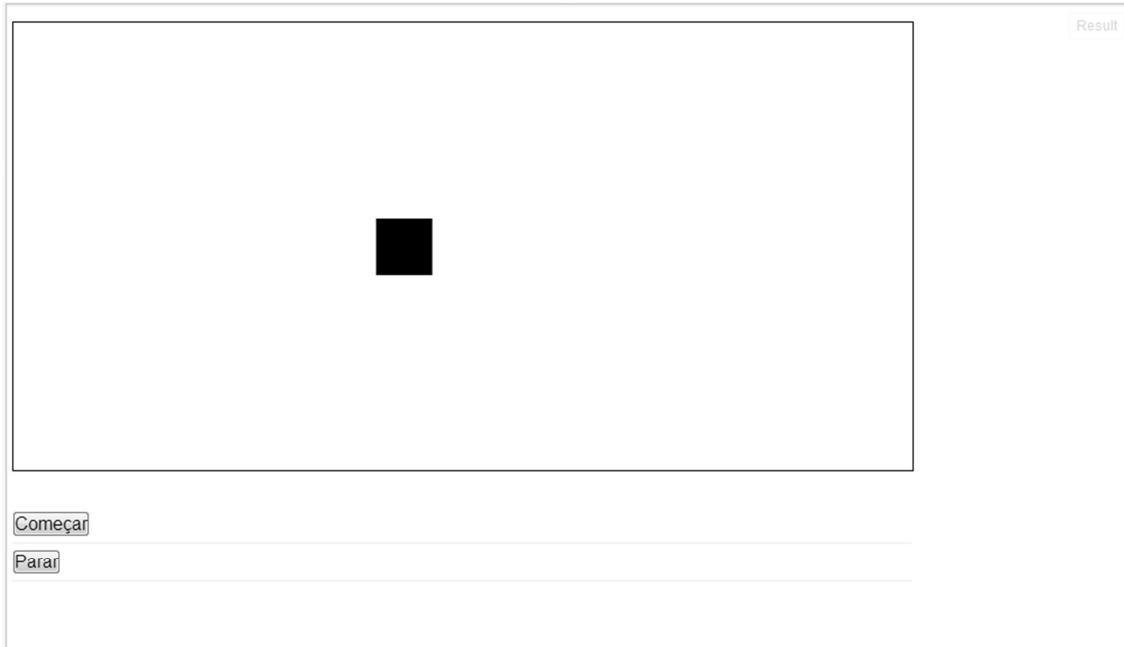


Figura 11 : O bloco preto oscilando dentro do canvas.

Podemos alterar as condições iniciais e vermos claramente como um oscilador se comportaria com tais condições.

Finalmente, temos uma animação possível de ser executada em qualquer navegador com tecnologia HTML5 sem a necessidade de plugins adicionais.

O Oscilador Harmônico Amortecido

Já vimos o caso em que o oscilador harmônico não apresenta qualquer tipo de atrito com a superfície que o bloco está apoiado. No entanto, em casos reais isso não se aplica pois sempre haverá um atrito que vai se opor as oscilações.

Damos o nome de amortecimento (ou "damping") a isso e em nosso oscilador a amplitude máxima vai decrescendo a medida que o movimento é executado.

Chamaremos de força de amortecimento a força que se contrapõe ao movimento do bloco, denotando :

$$F_a = -bv = -b\dot{x}$$

Colocando esta equação na segunda lei de newton (de forma análoga ao oscilador simples), teremos :

$$\begin{aligned} m\ddot{x} &= -kx - b\dot{x} \Rightarrow \\ &\Rightarrow m\ddot{x} + b\dot{x} + kx = 0 \Rightarrow \\ &\Rightarrow \ddot{x} + 2\beta\dot{x} + \omega_0^2 x = 0 \quad (1) \end{aligned}$$

Onde $2\beta = \frac{b}{m} \Rightarrow \beta = \frac{b}{2m}$, chamada de parâmetro de amortecimento e $\omega_0 = \sqrt{\frac{k}{m}}$ é a frequência angular na ausência do amortecimento (igual ao caso do oscilador simples).

A equação (1) é uma equação diferencial ordinária de segunda ordem. Não entraremos em detalhe em como resolvê-la, portanto, verifique em alguma bibliografia específica os termos que utilizaremos daqui em diante.

As raízes da equação indicial serão:

$$r_1 = -\beta + \sqrt{\beta^2 - \omega_0^2}$$

$$r_2 = -\beta - \sqrt{\beta^2 - \omega_0^2}$$

E a solução da equação (1) será (as constantes dependem das condições iniciais do movimento) :

$$x(t) = e^{-\beta t} \left[A_1 \exp\left(\sqrt{\beta^2 - \omega_0^2} * t\right) + A_2 \exp(-\sqrt{\beta^2 - \omega_0^2} * t) \right] \quad (2)$$

Existem 3 tipos de casos diferentes para o oscilador harmônico amortecido : superamortecido, subamortecido e amortecimento crítico.

Cada um deles dependem dos parâmetros β e ω_0 conforme abaixo:

- Subamortecimento: $\omega_0^2 > \beta^2$
- Amortecimento crítico : $\omega_0^2 = \beta^2$
- Superamortecido: $\omega_0^2 < \beta^2$

Veremos cada caso separadamente e o que deve ser mudado em cada script.

Subamortecimento do oscilador harmônico amortecido

Neste caso, é conveniente definir uma nova variável:

$$\omega_1^2 = \omega_0^2 - \beta^2$$

E ficamos com a solução $x(t)$ sendo:

$$x(t) = e^{-\beta t} [A_1 e^{i\omega_1 t} + A_2 e^{-i\omega_1 t}]$$

Utilizando a fórmula de Euler $e^{ix} = \cos(x) + i \sin(x)$:

$$x(t) = A e^{-\beta t} \cos(\omega_1 t - \varphi)$$

Onde A é a amplitude do movimento e φ depende das condições iniciais (posição inicial do bloco).

Para criar este movimento, basta que modifiquemos algumas variáveis e a equação que calcula a posição do bloco.

Variáveis adicionais:

```
//velocidade, posicao em x, massa do bloco, parametros de amortecimento
var bloco = {velocidade : 0 , x : 0 , xanterior : 0 , massa : 0 , amplitude : 0 , omega_0 : 0
, omega_1 : 0 , beta : 0 , b : 0};
```

Inicialização das variáveis na função **comecarDesenho()** (valores de exemplo):

```
bloco.b = 0.1;
bloco.massa = 100;
mola.k = 0.01;
bloco.amplitude = 100;
bloco.omega_0 = Math.sqrt(mola.k/bloco.massa);
bloco.beta = (bloco.b / (2*bloco.massa));
bloco.omega_1 = Math.sqrt(Math.pow(bloco.omega_0,2) - Math.pow(bloco.beta,2));
```

Equação do movimento:

```
bloco.x = bloco.amplitude*Math.pow(Math.E,-bloco.beta*t)*Math.cos(bloco.omega_1*t );
```

Nesta incluímos o método `Math.pow(x,y)`, que eleva o valor x a uma potência y e a constante de Euler `Math.E`.

Com esses códigos alterados na parte Javascript dentro do JSFiddle é possível obter o movimento amortecido desejado.

Amortecimento crítico do oscilador harmônico amortecido

Se $\omega_0^2 = \beta^2$ a equação do movimento fica um pouco mais simples (anulando-se os termos e), ficando:

$$x(t) = (A + Bt)e^{-\beta t}$$

Onde A e B são constantes que dependem das condições iniciais (velocidade e posição).

Este amortecimento pode ser comparado a amortecedores de veículos. Caracteriza-se por um rápido decrescimento do movimento do bloco.

Se utilizarmos as condições $x(0) = 0$ e $\dot{x}(0) = 0$ as constantes vão se anular e o movimento não vai existir. Portanto, ao definirmos um valor inicial para a posição (por exemplo C), teremos:

$$x(t) = (C + C\beta t)e^{-\beta t}$$

Isso nos leva a seguinte conclusão: o movimento necessita de algum valor diferente de 0 para sua posição e opcionalmente de uma velocidade inicial.

Portanto as variáveis ficam sendo:

```
//velocidade, posicao em x, massa do bloco, parametros de amortecimento
var bloco = {velocidade : 0 , x : 0 , xanterior : 0 , massa : 0 , amplitude : 0 , omega_0 : 0 ,
, omega_1 : 0 , beta : 0 , b : 0, x_inicial : 0};
```

Inicialização das variáveis na função **comecarDesenho()** (valores de exemplo):

```
bloco.b = 0.5;
bloco.massa = 100;
mola.k = 0.01;
bloco.x_inicial = 100;
bloco.beta = (bloco.b / (2*bloco.massa));
```

Equação do movimento:

```
bloco.x = (bloco.x_inicial + (bloco.beta*bloco.x_inicial*t))*Math.pow(Math.E,-bloco.beta*t);
```

Falta apenas definir agora o movimento superamortecido.

Superamortecimento do oscilador harmônico amortecido

Se $\beta^2 > \omega_0^2$, os coeficientes da equação diferencial do oscilador se tornam reais. É conveniente definir $\omega_2^2 = \beta^2 - \omega_0^2$ e ficamos com :

$$x(t) = e^{-\beta t} [A_1 e^{\omega_2 t} + A_2 e^{-\omega_2 t}]$$

Vale notar que ω_2 não representa a frequência angular, já que o movimento não é periódico.

Vemos que assim como o amortecimento crítico, se definirmos $x(0) = 0$ e $\dot{x}(0) = 0$ o movimento não vai acontecer. Se, ao invés disso, colocarmos $x(0) = C$ e $\dot{x}(0) = 0$, teremos:

$$\dot{x}(t) = e^{-t(\beta+\omega_2)} [A_1(\omega_2 - \beta)e^{2t\omega_2} - A_2(\beta + \omega_2)]$$

$$\dot{x}(0) = 0 \Rightarrow A_1(\omega_2 - \beta) - A_2(\beta + \omega_2) = 0 \Rightarrow A_2 = A_1 \frac{(\omega_2 - \beta)}{(\omega_2 + \beta)}$$

Colocando isto em $x(0) = C$, teremos:

$$x(0) = C \Rightarrow$$

$$\Rightarrow A_1 + A_2 = C \Rightarrow$$

$$\Rightarrow A_1 \left(1 + \frac{\omega_2 - \beta}{\omega_2 + \beta} \right) = C \Rightarrow$$

$$A_1 = \left(\frac{\omega_2 + \beta}{\omega_2} \right) C$$

Resolvendo para A_2 :

$$A_2 = \left(\frac{\omega_2 + \beta}{\omega_2} \right) \left(\frac{\omega_2 - \beta}{\omega_2 + \beta} \right) C \Rightarrow A_2 = \left(\frac{\omega_2 - \beta}{\omega_2} \right) C$$

Portanto, a equação será:

$$x(t) = e^{-\beta t} \left\{ \left[\left(\frac{\omega_2 + \beta}{\omega_2} \right) C \right] e^{\omega_2 t} + \left[\left(\frac{\omega_2 - \beta}{\omega_2} \right) C \right] e^{-\omega_2 t} \right\}$$

Onde C será a nossa posição inicial.

Então, nossas variáveis ficarão:

```
//variaveis do bloco
var bloco = { A1 : 0, A2 : 0, x: 0, x_inicial : 0, massa: 0, omega_0: 0,omega_2: 0,beta: 0,
b: 0 };
```

A inicialização destas:

```
bloco.x_inicial = 100;
bloco.b = 5;
bloco.massa = 100;
mola.k = 0.01;

bloco.omega_0 = Math.sqrt(mola.k / bloco.massa);
bloco.beta = (bloco.b / (2 * bloco.massa));

if (Math.pow(bloco.beta, 2) < Math.pow(bloco.omega_0, 2)) alert("ERRO! Beta menor
que Omega_0.");

bloco.omega_2 = Math.sqrt(Math.pow(bloco.beta, 2) - Math.pow(bloco.omega_0, 2));

bloco.A1 = ((bloco.omega_2 + bloco.beta) / bloco.omega_2)*bloco.x_inicial ;
bloco.A2 = ((bloco.omega_2 - bloco.beta) / bloco.omega_2)*bloco.x_inicial ;
```

Aqui atentamos ao fato de que usamos uma estrutura de decisão para disparar um erro caso o $\beta^2 < \omega_0^2$.

Tomamos o cuidado também de calcular as constantes A_1 e A_2 uma só vez.

Finalizando, o trecho que realizar o cálculo final é :

```
bloco.x = Math.pow(Math.E, -bloco.beta * t) * (bloco.A1*(Math.pow(Math.E, bloco.omega_2 * t))
+ bloco.A2*(Math.pow(Math.E, -bloco.omega_2 * t)))
```

Então, obtemos o movimento de um bloco com superamortecimento.

Aumentando a interatividade do applet

Até agora utilizamos valores fixos para as demonstrações de ambos os casos de osciladores (amortecido e não-amortecido).

Seria interessante que pudéssemos modificar os valores como velocidade inicial, massa, amortecimento, etc ao diretamente na tela do browser.

Para isto, utilizaremos alguns elementos em HTML5 e pouco de alguns códigos em Javascript.

O componente input e o tipo “range”

Foi introduzida a esta mais nova versão do HTML5 um novo tipo para o componente <input>, que se chama “range”.

De certa forma, este novo tipo mostra ao usuário um ajuste manual de valores com um valor mínimo e máximo definidos e também de quanto em quanto podemos variar estes valores. Sua utilização é bem simples:

```
<input type="range" min="0" max="10" value="0" step="0.1" name="b" id="b" class="valores">
```

Colocando este código no HTML do applet obteremos uma barra da forma:

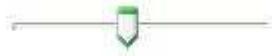


Figura 12 : O componente input com seu tipo definido como “range”.

Podemos então utilizar este componente de forma tranquila com relação a validação dos valores que serão passados ao código Javascript pois é possível definir a faixa numérica que ele pode ser alterado. Vamos utilizar esse componente para definir a velocidade inicial do movimento.

Utilizando o mouse para mover o bloco

É interessante que possamos usar o mouse para mover o bloco e com isso visualizarmos melhor como é o comportamento de um sistema físico real.

Para isso, vamos utilizar alguns eventos no Javascript. Podemos definir a seguinte variável globalmente em nosso script:

```
var mouse = {x : 0 , clicado : false};
```

Esta se encarregará de armazenar a posição do mouse dentro do <canvas> e uma variável booleana para validar se está havendo um clique ou não.

A seguir, definimos a função **getMousePosition()**:

```
var getMousePosition = function(e) {
    mouse.x = e.pageX - canvas.offsetLeft;
    if (mouse.clicado) {
        bloco.x = mouse.x - (widthCanvasAnimacao / 2) - 25;
        bloco.x_inicial = mouse.x - (widthCanvasAnimacao / 2) - 25;
        t = 0;
    }
};
```

Onde só alteramos o valor x do bloco quando o mouse estiver clicado e ao final zeramos o tempo.

Trocamos também dentro da função **redesenhar()** o instante que o valor final da posição do bloco é calculado pelo seguinte trecho:

```
if (!mouse.clicado) {
    bloco.x = bloco.x_inicial*Math.pow(Math.E,-bloco.beta*t)*Math.cos(bloco.omega_1*t );
```

Isto é, o cálculo só irá ocorrer quando o mouse não estiver com seu botão esquerdo sendo pressionado.

Finalmente, dentro da função **comecarDesenho()** (a primeira a ser chamada) definiremos adicionalmente ao código já existente o seguinte:

```
//funções de movimento do mouse
canvas.onmousemove = get.mousePosition;

canvas.onmousedown = function(e) {
    if (e.which == 1) {
        get.mousePosition(e);
        mouse.clicado = true;
    }
};

canvas.onmouseup = function(e) {
    if (e.which == 1) {
        mouse.clicado = false;
    }
};
```

Ou seja, para as funções já inerentes ao próprio <canvas> “onmousemove” (quando o ponteiro se mexer dentro da área do canvas), “onmousedown” (ao pressionarmos o botão esquerdo) e “onmouseup” (ao soltarmos o botão esquerdo) definimos novas funções para alterarmos os valores da variável **mouse** e/ou invocarmos a função **get.mousePosition()**.

De posse destas alterações, é perfeitamente possível obter em qualquer um dos applets acima algumas interações interessantes.

Colocando textos dinâmicos

Conforme o movimento desejado ocorre, é interessante que possamos ver como os valores mudam sem para que seja necessário depurar o código.

Para realizar isto, podemos utilizar inúmeros componentes do HTML5 em conjunto com o framework já citado para Javascript chamado Jquery. Neste trabalho não iremos aprofundar nisto, por isso utilizaremos o componente ``. Para defini-lo, basta escrevermos:

```
<span id="posicao"></span>
```

Onde colocamos um id para este componente e nenhum texto interno. O deixamos vazio propositalmente para que durante a execução do código Javascript este seja atualizado com novos valores.

Já no Javascript, basta chamarmos a seguinte linha:

```
$("#posicao").html(bloco.x);
```

Ou seja, dizemos para preencher o componente com id **posicao** o valor **bloco.x**. Vale notar que o comando `.html()` sempre irá trocar o conteúdo previamente inserido.

Finalizando o código com os parâmetros

O oscilador harmônico amortecido no caso subamortecido com parâmetros

Podemos definir a posição inicial e a velocidade inicial do oscilador harmônico amortecido fazendo:

$$x(0) = S_1 \text{ e } \dot{x}(0) = V_1$$

Assim as equações do oscilador harmônico amortecido no caso subamortecido ficam:

$$A \cos \varphi = S_1 \text{ e } \omega_1 A \sin \varphi = V_1$$

Portanto:

$$\tan \varphi = \frac{V_1}{\omega_1 S_1} \text{ e } A = \frac{S_1}{\cos \varphi}$$

E a equação fica :

$$x(t) = \frac{S_1}{\cos \varphi} e^{-\beta t} \cos \left(\omega_1 t - \tan^{-1} \left(\frac{V_1}{\omega_1 S_1} \right) \right)$$

Se $S_1 = 0$ ou $V_1 = 0$, ficamos com:

$$x(t) = S_1 e^{-\beta t} \cos(\omega_1 t)$$

Que é a equação inicial que apresentamos.

Assim, basta criarmos “ranges” com os parâmetros desejados e a interação via mouse para que o applet seja totalmente parametrizado.

Devemos alterar as variáveis do bloco, incluir o código dos “ranges” no HTML e adicionar os controles no Javascript. Deixaremos o código de forma completa abaixo em um link, onde é possível utilizá-lo em algum computador com o servidor Web instalado ou coloca-lo diretamente no JSFiddle (tomando o cuidado de separar a parte HTML, Javascript e CSS no software).

Porém, antes vamos nos atentar ao seguinte : se o valor de amortecimento $b = 0$, a variável $\beta \rightarrow 0$ pois $\beta = \frac{b}{2m}$. Portanto, a equação do movimento para o caso subamortecido fica:

$$x(t) = \frac{S_1}{\cos \varphi} \cos \left(\omega_1 t - \tan^{-1} \left(\frac{V_1}{\omega_1 S_1} \right) \right)$$

Que é exatamente a equação para o caso não amortecido. Assim, omitiremos o caso do oscilador harmônio simples com parâmetros e basta colocar 0 no “range” do parâmetro b para visualizar o caso do oscilador simples.

O link para o código funcional e completo será:

<http://vigo.ime.unicamp.br/osciladores/oha-subamortecido.html>

O oscilador harmônio amortecido no caso crítico com parâmetros

Assim como antes, faremos:

$$x(0) = S_1 \text{ e } \dot{x}(0) = V_1$$

De onde na equação do oscilador harmônico amortecido no caso crítico fica:

$$A = S_1 \text{ e } B = V_1 + S_1\beta$$

E a equação final fica:

$$x(t) = e^{-\beta t} [S_1 + (V_1 + S_1\beta)t]$$

Podemos ver que se $V_1 \rightarrow 0$ a equação toma a forma:

$$x(t) = e^{-\beta t} [S_1 + S_1\beta t]$$

Que é a mesma forma proposta anteriormente. Vale notar que se não houver $\beta \neq 0$, ficaremos com $x(t) = S_1$, onde o bloco ficará imóvel na posição inicial definida.

O link para o código funcional e completo será:

<http://vigo.ime.unicamp.br/osciladores/oha-critico.html>

O oscilador harmônio amortecido no caso superamortecido com parâmetros

Neste caso as equações ficam um pouco complicadas, mas basta um pouco de manipulação algébrica para simplificá-las, portanto, fazendo:

$$x(0) = S_1 \text{ e } \dot{x}(0) = V_1$$

E colocando na equação

$$x(t) = e^{-\beta t} [A_1 e^{\omega_2 t} + A_2 e^{-\omega_2 t}]$$

ficamos com:

$$A_1 + A_2 = S_1 (*)$$

e

$$A_1(\omega_2 - \beta) - A_2(\omega_2 + \beta) = V_1 (**)$$

Utilizando (*) da forma $A_1 = S_1 - A_2$ em (**) obtemos para A_2 a seguinte expressão:

$$A_2 = \frac{S_1(\omega_2 - \beta) - V_1}{2\omega_2}$$

E usando na primeira expressão de novo obtemos:

$$A_1 = \frac{2S_1\omega_2 - S_1(\omega_2 - \beta) + V_1}{2\omega_2}$$

De forma que a equação do movimento no script deve constar com estes cálculos das constantes para que o movimento seja superamortecido.

Enfim, o link para o código completo e funcional será:

<http://vigo.ime.unicamp.br/osciladores/oha-superamortecimento.html>

Finalizamos, então, a parametrização dos valores de velocidade inicial e posição inicial para o movimento do bloco.

Conclusão

De fato, mostramos que é possível a criação de um “applet” com a tecnologia HTML5 e Javascript. Conforme dito anteriormente, a grande vantagem de utilizar estas novas tecnologias é de que não existirá a necessidade de instalarmos plug-ins e derivados para que possamos utilizá-los.

O conceito do oscilador harmônico amortecido utilizado neste trabalho foi relativamente simples, pois não houve a necessidade de utilizar algoritmos complicados para obtermos os valores de posição do bloco.

De posse de alguma criatividade e interesse, é possível utilizar o JSFiddle e/ ou servidores locais para criar applets que representem situações reais somente utilizando algum browser mais recente para visualizar o resultado.

Bibliografia

- [1] - S. T. Thornton e J. B. Marion, Classical Dynamics, 5^a ed., Thomson Brooks/Cole, 2004.
- [2] - Halliday e Resnick, Fundamentos da Física, Volume II, Gravitação, Ondas e Calor, 8a Edição, 2010.
- [3] – W3Schools – The world largest web development database , em <http://www.w3schools.com/>.
- [4]- JQuery – Javascript Library – em <http://jquery.com/>.
- [5] – The Concord Consortium - HTML5-based scientific models, visualizations, graphing, and probeware, em <http://lab.dev.concord.org>.
- [6] – HTML5 - Canvas Cookbook em <http://www.html5canvastutorials.com>