

Programação Web para Jogos



Cruzeiro do Sul Virtual
Educação a distância

Material Teórico



Canvas, Tratamento de Eventos, Multimídia e Persistência

Responsável pelo Conteúdo:

Prof. Me. Alcides Teixeira Barboza Junior

Revisão Textual:

Prof.^a Dr.^a Selma Aparecida Cesarin

UNIDADE

Canvas, Tratamento de Eventos, Multimídia e Persistência



- Introdução;
- Conceitos Básicos do Elemento *Canvas*;
- Manipulação do Elemento *Canvas* em *Javascript*;
- Tratando Eventos no *Canvas* com *Javascript*;
- Manipulação de Áudio e Persistência de Dados no Jogo;
- Criando um Jogo com *Javascript* e o Elemento *Canvas*.



OBJETIVO DE APRENDIZADO

- Apresentar o desenvolvimento de jogos com o elemento *Canvas*;
- Apresentar o tratamento de ações (eventos) do usuário no *Canvas*;
- Demonstrar o uso de áudio conforme ações no jogo;
- Apresentar como persistir dados no jogo;
- Desenvolver um jogo com as Tecnologias Web utilizando *Canvas*.

Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:

Determine um horário fixo para estudar.

Mantenha o foco! Evite se distrair com as redes sociais.

Procure manter contato com seus colegas e tutores para trocar ideias! Isso amplia a aprendizagem.

Seja original! Nunca plágie trabalhos.

Aproveite as indicações de Material Complementar.

Conserve seu material e local de estudos sempre organizados.

Não se esqueça de se alimentar e de se manter hidratado.

Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

Contextualização

Você desenvolveu um Jogo na Unidade 2 trabalhando exclusivamente com *tagsdiv* e imagens, um pouco de *CSS* e *JavaScript*.

A ideia foi apresentar as três Tecnologias do lado cliente para você conhecer e desenvolver algo real.

Como foi sua experiência com esse primeiro Jogo? Lembre-se que a prática leva à perfeição e, nesse caso, à criatividade para criar outros jogos.

Embora seja possível criar jogos para a *Web* no formato visto até agora, existem outras possibilidades, como, por exemplo, usando a *tag CANVAS*. Essa *tag* foi introduzida no *HTML5* e permite criar desenhos, animações e jogos por meio do *JavaScript*.

Nesta unidade, iremos conhecer como utilizar a *tagCanvas* e como tratar eventos do usuário nesse elemento do *HTML* com o uso do *JavaScript*. O objetivo principal é desenvolver um jogo utilizando o *canvas* e com recursos adicionais; veremos como aplicar áudio e gravar informações do jogador, como, por exemplo, o *high score* no jogo.

Vamos começar!

Introdução

Nesta unidade, iremos aprender como utilizar a *tag*<canvas>. Essa *tag*, juntamente com outros recursos das tecnologias Web (CSS e JavaScript), possibilita o desenvolvimento de jogos em navegadores que utilizam técnicas de renderização de imagens e desenhos por meio de programação.

O conteúdo da *tag*<canvas> é renderizado com a Linguagem *JavaScript* e, para fazer uso desta *tag*, é necessário inseri-la dentro da *tag*<body> do documento HTML e acessá-la com o *JavaScript*.

Vamos entender um pouco, também, sobre como funciona o tratamento de eventos em *JavaScript*. Esses eventos são um conjunto de ações processadas num elemento inserido no HTML; nesse caso, o elemento HTML que sofrerá a ação será a *tag*<canvas>.



Importante!

Para você seguir com os passos apresentados nesta Unidade, terá de fazer o *download* do arquivo extra disponível no ambiente *Blackboard*.

Conceitos Básicos do Elemento Canvas

O elemento ou *tag* canvas foi introduzido no HTML5 e sua sintaxe possui a *tag* de abertura e de fechamento. Possui, também, somente dois atributos, o *width* e o *height*. Esses atributos são opcionais e podem ser configurados por meio do CSS.

A sintaxe da *tag* é:

```
<canvas id="algum_nome" width="150" height="150"></canvas>
```

Como é um recurso novo, navegadores antigos não o reconhecem. Nesse caso, podemos inserir um texto ou imagem dentro do bloco da *tag*. Assim, se o navegador não reconhecer o comando, exibirá a mensagem que adicionamos.

Para inserir o texto ou imagem, devemos seguir a sintaxe:

```
<canvas id="algum_nome" width="150" height="150">
```

Inserimos aqui nosso texto ou imagem com a *tag*

```
</canvas>
```

Crie um arquivo novo do tipo *HTML* no *Brackets* e insira o código do exemplo da Figura 1.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Run</title>
6      <meta charset="UTF-8">
7  <style>
8      #exemplo{
9          background-color: red;
10         border: 2px solid #00f;
11     }
12  </style>
13 </head>
14 <body>
15     <canvas id="exemplo" width="150" height="150">
16     </canvas>
17 </body>
18 </html>
```

Figura 1 – Tag Canvas com CSS

Após salvar seu arquivo, abra-o no navegador.

O que você visualizou?

Deve ser algo simples, como um quadrado com fundo vermelho e borda azul, certo?

Certo! Isso é o que o *canvas* é em *HTML*; porém, a partir do momento que o inserimos na página, temos uma área gráfica que poderá ser manipulada de diferentes formas com o uso de *JavaScript*.

Antes de avançarmos, precisamos entender o Sistema de coordenadas utilizado na Programação de Jogos usando *HTML* e *JavaScript*. Em algumas tecnologias, como *engines* famosas do Mercado, o Sistema de Coordenadas é igual ao da Matemática, isto é, temos os três eixos X, Y e Z nos mesmos sentidos.

No nosso contexto, que é o elemento *canvas* do *HTML*, o sistema de coordenadas muda de sentido no eixo Y. Nesse caso, os valores positivos aumentam para baixo e os negativos para cima, ambos em relação à origem.

A Figura 2 demonstra o sistema de coordenadas 2D com base no exemplo do *canvas* criados anteriormente.

Os elementos inseridos no *Canvas* serão posicionados a partir da origem (0,0), sendo que os valores de X negativos são deslocados para a esquerda da tela, fazendo com que eles fiquem ocultos no navegador.

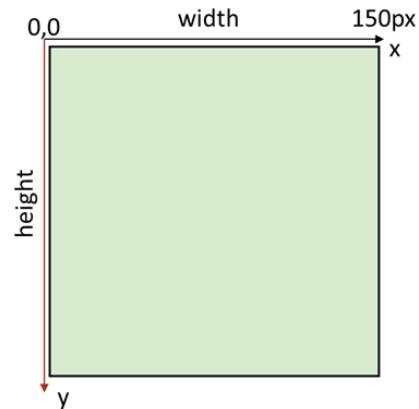


Figura 2 – Sistema de coordenadas 2D, *Canvas*

Já valores positivos deslocam os elementos para a direita, caso o valor seja maior que o tamanho do *canvas*; geralmente, os elementos desaparecem do lado direito.

Para o eixo Y, o princípio é semelhante: para valores negativos, os elementos sobem, podendo desaparecer na parte superior, e para valores positivos, os elementos descem e também podem desaparecer caso o valor estipulado seja maior que o *height* do *canvas*.

Como você notou, usar o *canvas* em HTML somente não tem muita utilidade; então, vamos começar a manipular essa *tag* em *JavaScript*.

Manipulação do Elemento *Canvas* em *JavaScript*

O elemento *Canvas* do *HTML* fornece um mecanismo de renderização 2D ou 3D para o *JavaScript*. Assim, manipularmos o *canvas* no *JavaScript*.

Primeiro, devemos capturar a referência no código; em seguida, devemos escolher qual o contexto em que desejamos trabalhar. O contexto pode ser 2d ou *webgl* para se trabalhar com 3D.

Na Figura 3, exibimos um pequeno exemplo. Crie um arquivo *HTML* com o código apresentado.

```

1  <!DOCTYPE html>
2 ▼ <html>
3 ▼ <head>
4      <meta charset="UTF-8">
5      <title>Run</title>
6      <meta charset="UTF-8">
7 ▼     <style>
8 ▼         #exemplo{
9             border: 2px solid #f00;
10        }
11    </style>
12  </head>
13 ▼ <body>
14      <canvas id="exemplo" width="150" height="150"></canvas>
15
16 ▼     <script>
17     var canvas = document.getElementById("exemplo");
18     var ctx = canvas.getContext("2d");
19
20     ctx.beginPath();
21     ctx.arc(130, 50, 20, 0, 2 * Math.PI);
22     ctx.stroke();
23
24     ctx.font = "20px Arial";
25     ctx.fillText("Olá, tudo bem?", 0, 130);
26
27     ctx.fillStyle = "blue";
28     ctx.fillRect(10, 10, 100, 100);
29   </script>
30 </body>
31 </html>
```

Figura 3 – Manipulando o *canvas* em *JavaScript*

Criamos um elemento *canvas* na linha 14 com o id “exemplo”. Precisamos, então, capturar esse id em *JavaScript*.

A linha 17 apresenta uma das formas possíveis de se capturar o elemento por meio do seu id. Nessa linha, criamos a variável chamada *canvas*, que terá a referência da *tag*, ou seja, tudo que for feito com essa variável no sentido gráfico, será exibido no *canvas*.

Como mencionado, precisamos escolher em qual o contexto do *canvas* iremos trabalhar. Assim, na linha 18, escolhemos o contexto 2d e associamos essa escolha a uma variável chamada *ctx* (abreviação para contexto).

A partir da linha 20, estamos utilizando as funções associadas ao contexto *e*, consequentemente, ao *canvas*, que nos permitem desenhar e escrever no *canvas*.

Por exemplo, a linha 21 permite configurar um círculo que será pintado posteriormente pelo comando *stroke* (somente contorno). Esta função solicita os parâmetros *arc* (*x*, *y*, *raio*, ângulo inicial, ângulo final).

Já na linha 25, estamos utilizando a função que irá escrever um texto dentro do *canvas*, na qual passamos o texto em que será escrita a posição X e Y. Por fim, na linha 28, estamos desenhando um quadrado com a cor azul de preenchimento.

Esses comandos foram somente para demonstrar como usar a *tag canvas* no *JavaScript*. Tenha em mente que existem outros comandos. Procure acessar os *links* dos materiais complementares ou as nossas indicações do “saiba mais”, para conhecer outras possibilidades.

Outro conceito importante que precisamos conhecer está relacionado à animação.

Como podemos fazer uma animação? Afinal, os jogos estão repletos de movimentos, correto?

Existem alguns passos básicos para fazer isso e algumas funções que chamamos de temporizadores.

Primeiro, precisamos analisar os passos que são necessários para fazer uma animação.

Os passos são:

- Limpar o *canvas*;
- Salvar o estado do *canvas* atual (opcional em alguns casos);
- Desenhar algo;
- Restaurar o estado do *canvas* (opcional em alguns casos).

Para controlarmos uma animação, podemos acionar funções com base nas ações do usuário ou por meio de temporizadores. As funções que podem ser utilizadas são:

- *setInterval* (função que será chamada tempo da chamada):
 - » Faz a chamada a uma função a cada milissegundo repetidamente;

- setTimeout (função que será chamada tempo da chamada):
 - » Faz a chamada a uma função depois que o tempo em milissegundos passar; essa chamada é executada somente 1 vez;
- requestAnimationFrame():
 - » um método mais moderno para fazer animações; possui uma taxa de atualização de 60 quadros por segundo; contudo, esse comando não elimina os demais. Tudo depende do que precisamos fazer.

Vamos fazer um exemplo simples para entendermos como usar o *canvas* e ao mesmo tempo fazer uma animação.

Crie um arquivo HTML e insira o código apresentado na Figura 4:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Run</title>
6      <meta charset="UTF-8">
7  <style>
8      #exemplo{
9          border: 2px solid #f00;
10     }
11  </style>
12 </head>
13 <body>
14     <canvas id="exemplo" width="800" height="200"></canvas>
15
16 <script>
17 var canvas = document.getElementById("exemplo");
18 var ctx = canvas.getContext("2d");
19 var posY=-10;
20 var posX=0;
21
22 function desenhaObjeto(){
23     ctx.clearRect(0,0,canvas.width,canvas.height);
24     ctx.fillRect(posX, posY, 20, 20);
25     if (posY >= canvas.height){
26         posX=Math.round(Math.random()*750);
27         posY=-10;
28     }
29     posY++;
30 }
31
32 setInterval(desenhaObjeto,10)
33 </script>
34 </body>
35 </html>

```

Figura 4 – Exemplo de animação

Abra o arquivo criado no navegador e veja o resultado. Como já comentamos, primeiro precisamos capturar a referência do *canvas* no *JavaScript* (linha 17); em seguida, escolhemos o contexto 2d (linha 18).

Nossa animação é bem simples: fazer um quadrado de dimensões 20x20 (linha 24) cair a partir da parte superior do *canvas* (linhas 19 e 27, variável *posY*) e desaparecer na parte inferior.

Sempre que o objeto desaparecer na parte inferior (linha 25), devemos criar um novo numa posição X aleatória (linha 26) e fazer novamente sua animação de cair.

O processo de repetir o código da função desenha Objeto() é feito pelo código da linha 32, que faz a chamada à função a cada 10 milissegundos.

Faça um teste no seu código: comente a linha 23 com // e veja o que acontece.

Você entendeu o que ocorreu?

Basicamente, não estamos limpando a tela e sempre que a função é chamada, desenha um novo objeto deslocando a posição Y. Com isso, criamos um efeito de arrasto, com a linha 23 e garantimos sempre que o *canvas* será limpo para que o próximo objeto seja desenhado.

Agora que você já sabe manipular o *canvas* em *JavaScript*, tente alterar seu código, substituindo a função setInterval pelas demais que comentamos.

Talvez seja necessário fazer pequenas mudanças no código, mas o desafio pode aprimorar seu raciocínio lógico.



Referências do elemento *canvas* da *W3Schools*. Acesse: <https://goo.gl/BYsVMS>

Tutorial da MDN *web docs* sobre *Canvas*. Acesse: <https://goo.gl/xG4YCZ>

Veja exemplo do uso do *requestAnimationFrame* em <https://goo.gl/6dxGae>

Exemplo de animação da *W3Schools*. Acesse: <https://goo.gl/hVL23u>

Exemplo de animação com o *requestAnimationFrame*, da *MDN Web docs*.

Acesse: <https://goo.gl/tdpM4k>

Veja outros exemplos bem interessantes no *site MDN web docs*. Procure refazer alguns dos exemplos desse *site* na sua máquina e rode para ver o resultado. Acesse <https://goo.gl/8PKW93>

Tratando Eventos no *Canvas* com *Javascript*

Num jogo, é comum realizarmos modificações conforme as ações do jogador, como, por exemplo, fazer o personagem pular quando se aperta o espaço, disparar um tiro ou fazer o personagem andar conforme as teclas direcionais são pressionadas.

O tratamento de eventos é o mesmo que já foi visto. A diferença é que devemos associar o evento ao *Canvas*. O tratamento de eventos no *JavaScript* pode ser adicionado aos elementos do *HTML*, ao documento, à janela ou ao formulário. Geralmente, estão associados às ações do usuário como cliques e pressionamento de teclas, mas existem casos em que são acionados por eventos próprios do documento.

Para adicionarmos um evento a um elemento, utilizamos a seguinte sintaxe:

```
elemento.addEventListener(evento, função);
```

Nessa sintaxe, o elemento é a referência por meio do ID feita no *JavaScript*, como já vimos antes. O parâmetro evento diz respeito aos eventos que existem e que podemos tratar, e a função é o que faremos quando o evento definido for disparado.

Os eventos associados ao mouse são:

- *click*
botão esquerdo do *mouse* ou tecla *enter*;
- *dblclick*
quando pressionamos duas vezes o botão do *mouse*;
- *mousedown*
quando pressionamos qualquer um dos botões do *mouse*;
- *mouseup*
quando liberamos o botão pressionado anteriormente;
- *mouseover*
quando colocamos o *mouse* sobre um elemento;
- *mouseout*
quando tiramos o *mouse* de um elemento;
- *mousemove*
quando movemos o ponteiro do *mouse*.

Alguns eventos que não dependem do usuário são:

- *load*
carregamento completo do conteúdo;
- *unload*
fechamento de um documento;
- *focus*
ocorre quando a janela ou algum elemento HTML recebe o foco.
- E, por fim, os elementos disparados por meio do teclado:
 - *keydown*
quando pressionarmos uma tecla;
 - *keypress*
quando pressionarmos uma tecla que resulte em um caractere;
 - *keyup*
quando soltarmos a tecla pressionada.

Crie um arquivo HTML novo na ferramenta. Insira o código apresentado na Figura 5.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Run</title>
6      <meta charset="UTF-8">
7  <style>
8      #exemplo{
9          border: 2px solid #f00;
10     }
11  </style>
12 </head>
13 <body>
14     <canvas id="exemplo" width="500" height="500"></canvas>
15
16 <script>
17 var canvas = document.getElementById("exemplo");
18 var ctx = canvas.getContext("2d");
19 var x = 0;
20 var y = 300;
21
22 canvas.addEventListener("click",trataClique);
23 document.addEventListener("keydown",trataTeclas);
24
25 function trataClique(e){
26     ctx.beginPath();
27     ctx.arc(e.clientX,e.clientY,20,0,Math.PI*2);
28     ctx.fill();
29 }
30
31 function trataTeclas(e){
32     var cod_tecla = e.keyCode;
33     //37 = esq / 38 = cima / 39 = dir / 40 = baixo
34     if (cod_tecla == 37){
35         x = x - 10;
36     } else if (cod_tecla == 39){
37         x = x + 10;
38     } else if (cod_tecla == 38){
39         y = y - 10;
40     } else if (cod_tecla == 40){
41         y = y + 10;
42     }
43     desenha(x,y);
44 }
45
46 function desenha(posx,posy){
47     ctx.clearRect(0,0,canvas.width,canvas.height);
48     ctx.fillRect(posx,posy,30,30);
49 }
50
51 desenha(x,y);
52 </script>
53 </body>
54 </html>

```

Figura 5 – Exemplo de eventos com *canvas*

Execute no navegador esse exemplo e veja o que acontece quando clica em algum ponto dentro do *canvas* e quando são pressionadas as teclas direcionais.

Percebeu que se trata de dois eventos distintos? O clique do *mouse* no *canvas* (linha 22) e o pressionamento das setas no documento (linha 23).

Quando você clica no *canvas*, o código chama a função *trataClique*, e quando pressiona alguma tecla, chama a função *trataTeclas*.

Também podemos, dentro da função que trata o evento, criar diferentes lógicas para executarmos diversas ações. Outro ponto importante é notarmos a função

chamada quando o evento é acionado (linha 25 e 31). Quando trabalhamos com eventos, é passado para a função chamada um parâmetro que corresponde ao objeto *Event* do *JavaScript*. Existem diversas propriedades nesse objeto. Iremos usar algumas durante nossa Disciplina.

No exemplo da Figura 5, usamos o objeto *Event* passado como parâmetro (variável *e*) para obtermos a posição do *mouse* no eixo X e eixo Y (linha 27) e para verificarmos o código da tecla pressionada (linha 32).

Embora seja possível adicionar mais de um evento no mesmo elemento, fique atento(a) à sua real necessidade para descobrir o que e quando usar.

Manipulação de Áudio e Persistência de Dados no Jogo

Estamos quase finalizando nossos conceitos básicos para que, na próxima seção, seja possível criar um jogo com tudo isso junto.

Agora, precisamos aprender como reproduzir um arquivo de áudio e como gravar alguns dados no navegador, tudo por meio do *JavaScript*.

Para reproduzir áudio com programação por meio do *JavaScript*. Inicialmente, podemos carregar o arquivo de áudio com a tag<áudio> ou, então, manipular o objeto *Audio* do próprio *JavaScript*.

Nos materiais desta Unidade que você baixou, existem alguns arquivos de áudio. Para facilitar nosso exemplo, crie um arquivo HTML dentro da pasta que possui esses arquivos de áudio e insira o código apresentado na Figura 6 neste arquivo.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Exemplo</title>
6      <meta charset="UTF-8">
7  </head>
8  <body>
9
10 <script>
11     var audio1 = new Audio();
12     audio1.src="bump.mp3";
13
14     document.addEventListener("click",tocarSom);
15
16     function tocarSom(){
17         audio1.play();
18     }
19
20 </script>
21 </body>
22 </html>
```

Figura 6 – Exemplo básico de áudio em *JavaScript*

Execute o exemplo verificando que, quando clicamos no documento (linha 14), é reproduzido um som (linha 17), que foi previamente carregado por meio do objeto Audio (linhas 11 e 12).

Ressaltamos que esse tipo de reprodução de áudio é o mais básico. Acesse as indicações de materiais no final desta Seção para verificar exemplos mais sofisticados e o uso da API Web Audio.

Outro conceito importante para nosso Jogo da próxima seção é o recurso que permite gravar dados no navegador.

Existem diferentes possibilidades. Neste material, iremos abordar o formato por meio do objeto *Storage*, que permite gravar dados com a propriedade *localStorage* ou *sessionStorage*. A diferença é que no primeiro caso (*localStorage*), os dados ficam armazenados até que sejam apagados pela aplicação ou pelo usuário. Já com o *sessionStorage*, os dados são apagados quando a janela ou a aba do navegador é fechada.

Para as duas propriedades, existem duas funções principais:

- *setItem("nome_chave", "valor_chave")*:

 - » criar uma nova chave no *Storage*;
 - » *getItem("nome_chave")*:
 - » retorna o valor de uma chave gravada no *Storage*.

Uma dica importante é saber onde você encontra o *Storage* no seu navegador. No *Chrome*, você pode pressionar a tecla F12 para ativar o painel de desenvolvedor.

No painel de desenvolvedor, localize a aba *Application* (aplicação). Você irá visualizar, no lado esquerdo desse painel, o *Storage* com diferentes opções.

Localize e abra a opção *Local Storage*. Clique na opção “*file*” que corresponde ao arquivo atualmente carregado e, no lado direito, visualize uma tabela com as colunas “key” e “value”.

Os valores que criarmos serão visualizados nesse tela. Veja um exemplo do *Storage* na Figura 7.

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. On the left, the 'Storage' section is expanded, showing 'Local Storage' with a single entry for 'file://'. This entry has a sub-section for 'Service Workers' which contains a key-value pair: 'pontuacaoSnake' with a value of '34'. The 'Console' tab is also visible at the top.

Key	Value
pontuacaoSnake	34

Figura 7 – Exemplo de dados gravados no *LocalStorage*

Vamos fazer um exemplo que irá gravar a quantidade de cliques dados dentro de um *canvas*.

Crie um arquivo HTML e insira o código apresentado na Figura 8.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Exemplo</title>
6      <meta charset="UTF-8">
7  <style>
8      #exemplo{border: 2px solid black;}
9  </style>
10 </head>
11 <body>
12     <canvas id="exemplo" width="100" height="100"></canvas>
13 <script>
14     var canvas = document.getElementById("exemplo");
15     var ctx = canvas.getContext("2d");
16
17     canvas.addEventListener("click",gravarClique);
18     document.addEventListener("keydown",mostrarCliques);
19
20     function gravarClique(e){
21         var cliques = 0;
22         if (localStorage.getItem("cliquecanvas")){
23             cliques = parseInt(localStorage.getItem("cliquecanvas"));
24             cliques = cliques + 1;
25         }else{
26             cliques = 1;
27         }
28         localStorage.setItem("cliquecanvas",cliques);
29     }
30
31     function mostrarCliques(e){
32         ctx.font = "20px Verdana";
33         ctx.clearRect(0,0,canvas.width,canvas.height);
34         if (e.keyCode == 13){
35             ctx.fillText(localStorage.getItem("cliquecanvas"),canvas.width/2,canvas.height/2);
36         }
37     }
38 </script>
39 </body>
40 </html>
```

Figura 8 – Exemplo de uso do *LocalStorage*

Nesse exemplo, sempre que clicarmos no *canvas*, será adicionado 1 clique na chave “cliquecanvas”

Faça um teste: clique algumas vezes, visualize o local *storage* após alguns cliques, feche o navegador e, em seguida, reabra o arquivo do exemplo.

Verifique novamente o local *storage*. Os cliques ainda estão gravados, certo?

Em nosso exemplo, armazenamos a quantidade de cliques no *canvas* (linhas 21 à 28). Perceba que na linha 34 verificamos se foi pressiona a tecla *Enter*; caso seja verdadeiro, é exibido no *canvas* em formato texto essa quantidade de cliques.

A lógica da função da linha 20 é verificar se já existe a chave armazenada no *storage*; se existir, devemos pegar esse valor convertido para inteiro e adicionamos mais um clique; em seguida, gravamos novamente o valor na mesma chave. Caso a chave não exista, então, precisamos atribuir o valor um de início, para então ser gravada posteriormente.

Vamos, agora, juntar tudo o que estudamos até aqui para criarmos um jogo com o uso de *canvas*.



Exemplo do uso do objeto *Audio* da *W3Schools*. Acesse: <https://goo.gl/wZzWcx>

Acesse o site *MDN web docs* para consultar sobre o uso mais avançado de reprodução de áudio em *JavaScript*. O link <https://goo.gl/eHMKTk> apresenta uma referência geral à API.

Basic concepts behind Web Audio API. Acesse: <https://goo.gl/M4mkto>

Introdução ao uso da *API Web Audio (CSS-TRICKS)*. Acesse: <https://goo.gl/RGJQ4G>

Usando o objeto áudio em *JavaScript*. Acesse: <https://goo.gl/NuMQxC>

Acesse o site *MDN web docs* para ler mais sobre áudio para *games*. Trata-se de material bem interessante: <https://goo.gl/y5beWt>

Para consultar alguns exemplos de jogos feitos em *HTML*, *CSS* e *JavaScript*, acesse o site *CodeinComplete*: <https://goo.gl/ZpNWg6>

Para verificar mais funções do objeto *Storage*, acesse o link da *W3Schools*: <https://goo.gl/131H1X> ou o link *MDN web docs*: <https://goo.gl/c16htR>

Acesse o site *MDN web docs* para ler o artigo referente ao uso da *API Web Audio* e testar alguns exemplos bem interessantes. O link está disponível em: <https://goo.gl/PFsDQX>

Criando um Jogo com *Javascript* e o Elemento *Canvas*

Vamos criar o famoso Jogo *Snake* (“Jogo da cobrinha”), um Jogo que sofreu diversas modificações desde 1979 (*Jogo Blockade*) e se tornou mais popular com os celulares da Nokia (1990), pois era um jogo que já vinha com o aparelho (Figura 9).

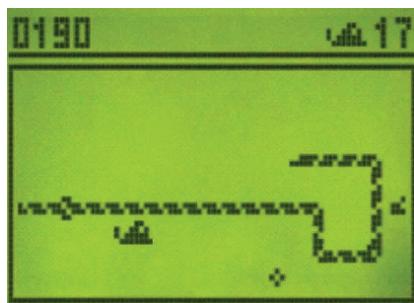


Figura 9 – Jogo *Snake* (NOKIA, 1990)

O Jogo consiste em controlar uma cobra que pode ser direcionada para a direita, para a esquerda, para cima e para baixo. O objetivo é alimentar a cobra fazendo-a colidir com as comidas que aparecem aleatoriamente. A cada comida coletada, o corpo da cobra cresce uma vez; a dificuldade do Jogo aumenta conforme o tamanho da cobra aumenta.

O critério de derrota está associado à colisão da sobra com as paredes do cenário ou com o seu próprio corpo.

Iremos começar criando o cenário para o Jogo. Esse cenário consiste em um *canvas* com dimensões de 600 x 600.

Crie um arquivo HTML com o código apresentado na Figura 10.

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>Snake Game</title>
6    <style>
7      canvas{
8        margin: 0 auto;
9        border: 1px solid #000;
10       display: block;
11     }
12   </style>
13 </head>
14 <body>
15   <canvas id="snake" width="600" height="600"></canvas>
16   <script src="snake.js"></script>
17 </body>
18 </html>
```

Figura 10 – Código HTML do Jogo *Snake*

Verifique que criamos um *canvas* (linha 15) com o id igual a “*snake*”. O *canvas* foi formatado em *CSS* (linhas 7 a 11) com uma borda preta (linha 9) e centralizado horizontalmente na tela (linha 8).

Na linha 16, foi inserida a referência ao arquivo *JS* que possui a lógica do Jogo como um todo; o *HTML* basicamente fornece somente o *canvas* para que possamos programar em *JavaScript*.

Abra o arquivo *snake.js* para começar a desenvolver o código. Lembre-se de que precisamos primeiro armazenar a referência ao *canvas* e escolher o contexto em que iremos trabalhar.

Insira o código apresentado na Figura 11.

```

1  var cvs = document.getElementById("snake");
2  var ctx = cvs.getContext("2d");
```

Figura 11 – Código *JavaScript* do Jogo *Snake* (parte 1)

Precisamos criar algumas variáveis para armazenarmos os dados do Jogo; também iremos criar um vetor para a cobra (Figura 12).

```

4  var box = 20;
5
6  var snake = [];
7
8  snake[0] = {
9    x : 10 * box,
10   y : 10 * box
11 };
12
13 var pontos=0;
14
15 var food = {
16   x : Math.floor(Math.random()*29+1) * box,
17   y : Math.floor(Math.random()*29+1) * box
18 }
19
20 var d;
```

Figura 12 – Código *JavaScript* do Jogo *Snake* (Parte 2)

A linha 4 cria uma variável que representa o tamanho de cada parte do corpo da cobra. Na linha 6, criamos um vetor que apresenta o corpo da cobra.

A ideia é que esse vetor aumente conforme ocorra a colisão com as comidas, aumentando, consequentemente, o tamanho da cobra. Cada parte da cobra será representada por um objeto.

Um objeto é delimitado com “{“ e “}” e pode conter atributos (variáveis) e métodos (funções), isto é, cada parte do corpo da cobra será um objeto composto pelas variáveis X e Y, que representam a posição desse objeto na tela.

Nas linhas 8 até 11, criamos a cabeça da cobra, que deverá estar no início do vetor, ou seja, posição 0. Os valores de x e y representam uma posição qualquer no *canvas* para começar o Jogo.

Analise que usamos a mesma ideia de objeto para criar a comida. A linha 15 define um objeto chamado “*food*”, composto também por X e Y; os valores desses atributos da comida serão sempre aleatórios dentro do *canvas*.

A variável criada na linha 20 servirá para representar a direção da cobra, conforme as teclas pressionadas.

Agora começaremos a lógica: como o jogador precisa interagir com o Jogo, iremos trabalhar com o evento de teclado. Para determinar as teclas pressionadas, insira no arquivo JS o código da Figura 13.

```

22 document.addEventListener("keydown",direcao);
23
24 ▼ function direcao(event){
25     var key = event.keyCode;
26 ▼     if( key == 37 && d != "RIGHT"){
27         d = "LEFT";
28 ▼     }else if(key == 38 && d != "DOWN"){
29         d = "UP";
30 ▼     }else if(key == 39 && d != "LEFT"){
31         d = "RIGHT";
32 ▼     }else if(key == 40 && d != "UP"){
33         d = "DOWN";
34     }
35 }
```

Figura 13 – Código JavaScript do Jogo *Snake* (Parte 3)

Sempre que o jogador pressionar as setas direcionais do teclado, iremos chamar a função “*direcao*”. Essa função verifica o código da tecla e atribui a variável d à direção que a cobra deve ir.

Perceba que em todas as condições da estrutura “if”, avaliamos o código da tecla e a direção atual da cobra; assim, garantimos que o jogador não faça a cobra voltar no sentido contrário da tecla pressionada.

Vamos criar, agora, uma função que irá verificar a colisão da cobra com seu próprio corpo (Figura 14).

```

37 ▼ function collision(head,array){
38 ▼     for(var i = 0; i < array.length; i++){
39 ▼         if(head.x == array[i].x && head.y == array[i].y){
40             return true;
41         }
42     }
43     return false;
44 }
  
```

Figura 14 – Código JavaScript do Jogo Snake (Parte 4)

Como a cobra está representada no código por meio de um vetor, precisamos de uma repetição (estrutura for) para percorrer cada parte do corpo e checar se o valor de X e Y das partes não é igual aos valores de X e Y da cabeça da cobra (linha 39). Se ocorreu a colisão, devolvemos verdadeiro (true); caso contrário falso (false).

A próxima função será o núcleo do Jogo. Iremos chamar a função de *draw*, que possui a tarefa de repintar a cobra na tela a cada movimento e fazer as checagens para verificar se o jogador comeu a comida ou se houve colisão.

Insira o código apresentado na Figura 15.

```

46 ▼ function draw(){
47     ctx.fillStyle = "#dbdbdb";
48     ctx.fillRect(0,20,cvs.width,cvs.height);
49
50 ▼     for( i = 0; i < snake.length ; i++){
51         ctx.fillStyle = ( i == 0 )? "green" : "blue";
52         ctx.fillRect(snake[i].x,snake[i].y,box,box);
53
54         ctx.strokeStyle = "red";
55         ctx.strokeRect(snake[i].x,snake[i].y,box,box);
56     }
57
58     ctx.fillStyle = "black";
59     ctx.fillRect(food.x, food.y, box, box);
60
61     var snakeX = snake[0].x;
62     var snakeY = snake[0].y;
63
64     if( d == "LEFT")  snakeX -= box;
65     if( d == "UP")   snakeY -= box;
66     if( d == "RIGHT") snakeX += box;
67     if( d == "DOWN") snakeY += box;
68
69 ▼     if(snakeX == food.x && snakeY == food.y){
70 ▼         food = {
71             x : Math.floor(Math.random()*29+1) * box,
72             y : Math.floor(Math.random()*29+1) * box
73         }
74         somaPonto();
75 ▼     }else{
76         snake.pop();
77     }
78
79 ▼     var newHead = {
80         x : snakeX,
81         y : snakeY
82     }
83
84     if(snakeX < 0 || snakeX > cvs.width-box || snakeY < 20 ||
85 ▼         snakeY > cvs.height-box || collision(newHead,snake)){
86         clearInterval(game);
87         gravaRecorde();
88     }
89
90     snake.unshift(newHead);
91 }
  
```

Figura 15 – Código JavaScript do Jogo Snake (Parte 5)

O bloco de código das linhas 50 a 56 é responsável por desenhar o corpo inteiro da cobra. Sempre que a cobra por repintada, é verificado se a posição do vetor é a inicial (0), o que indica que é a cabeça e deve ser colorida de verde; caso contrário, o corpo será pintado de vermelho (linha 51).

As linhas 58 e 59 desenham na tela a comida na cor preta.

Como a cobra deve se mover pela tela, devemos armazenar os valores de X e Y da cabeça (linhas 61 e 62) e, posteriormente, aumentar ou diminuir seus valores conforme a direção que o jogador escolheu (linhas 64 a 67).

Precisamos, ainda, verificar se a cobra comeu a comida ou se ocorreu algum tipo de colisão referente ao critério de derrota. O código apresentado nas linhas 69 a 77 verifica se houve a colisão da cobra com a comida. Caso tenha ocorrido, a comida é posicionada em outro ponto (x,y) aleatório e o jogador ganha um ponto (linha 74); caso contrário, é removida a primeira posição do vetor (linha 76), para que seja adicionada a nova posição da cobra, configurada na linha 79 a 82 e atribuída ao final do vetor na linha 90.

O trecho de código entre as linhas 84 e 88 faz a checagem da colisão da cobra com os cantos do *canvas* e com seu próprio corpo. Caso ocorra uma dessas colisões, o Jogo é interrompido (linha 86) e a pontuação alcançada é gravada no *Storage* (linha 87).

Estamos quase finalizando nosso Jogo.

Insira, agora, as linhas de código, conforme a Figura 16.

```

93 ▼ function somaPonto(){
94     pontos=pontos+1;
95     mostrarPontos();
96 }
97
98 ▼ function gravaRecorde(){
99     var record = 0;
100    if (localStorage.getItem("pontuacaoSnake") != null){
101        record = parseInt(localStorage.getItem("pontuacaoSnake"));
102    if (record < pontos){
103        record = pontos;
104    }
105    }else{
106        record = pontos;
107    }
108    localStorage.setItem("pontuacaoSnake",record);
109    mostrarRecorde();
110 }
111
112 ▼ function mostrarRecorde(){
113     ctx.clearRect(cvs.width - 250,0,cvs.width,20);
114     var record = 0;
115     ctx.fillStyle = "#0000ff";
116     ctx.font="20px Arial";
117    if (localStorage.getItem("pontuacaoSnake") != null){
118        record = parseInt(localStorage.getItem("pontuacaoSnake"));
119    }
120    ctx.fillText("Recorde: "+record, cvs.width - 250,16,250);
121 }
122
123 ▼ function mostrarPontos(){
124     ctx.clearRect(0,0,250,20);
125     ctx.fillStyle = "#ff0000";
126     ctx.fillText("Pontos: "+pontos, 1,16,250);
127 }
```

Figura 16 – Código JavaScript do Jogo *Snake* (Parte 6)

O código apresentado na Figura 16 é referente à pontuação e recorde no Jogo. A função somaPonto (linhas 93 a 96) é acionada sempre que a cobra come uma comida, e como os pontos precisam ser repintados no Canvas, assim que os pontos são somados, a função mostrarPontos (linhas 123 a 127) é acionada, para que seja atualizado o valor na tela.

As funções gravaRecorde e mostrarRecorde também trabalham em conjunto. A gravaRecorde tem a tarefa de verificar se existe um recorde já gravado no *storage* e se ele é menor do que o recorde alcançado na tentativa atual (linha 100 a 104); caso seja afirmativo, a função substitui o valor armazenado pelo novo valor (linha 103).

Se não existir nenhum recorde ainda no *storage*, a função armazena a pontuação atual, sem necessidade de verificações (linha 106). A efetivação da gravação no local *storage* ocorre na linha 108, na qual inserimos a chave “pontuacaoSnake”, que armazena no *storage* o recorde alcançado.

A função mostrarRecorde será sempre acionada quando ocorrer o critério de derrota, ou seja, *game over*. Essa função consulta o valor gravado na chave “pontuacaoSnake” (linha 118) e apresenta no *canvas* como um texto (linha 120).

Agora sim, vamos encerrar o Jogo, inserindo as linhas conforme a Figura 17.

```

129  mostrarRecorde();
130  mostrarPontos();
131
132  var game = setInterval(draw,100);

```

Figura 17 – Código JavaScript do Jogo Snake (parte 7)

Sempre que abrirmos o Jogo, devem ser apresentados os pontos, mesmo que zerados (linha 130), o recorde obtido na última jogada (linha 129) e, principalmente, devemos iniciar o temporizador.

O temporizador (linha 132) fará o Jogo funcionar acionando a função “draw” a cada 100 milissegundos. Assim, a cada direção pressionada, o *canvas* é repintado para mostrar a nova posição da cobra. Essa atualização do *canvas* realizada a certa velocidade dá a visualização de uma animação que, nesse caso, seria o movimento da cobra no *canvas*.

Salve seu arquivo e teste no navegador.

Você conseguiu rodar com sucesso o Jogo?

Caso tenha ocorrido algum erro (lembre-se do painel de desenvolvedor F12 e aba console), verifique seu código novamente com os trechos apresentados neste Material.

Parabéns!

Você fez seu primeiro Jogo utilizando *canvas*.

Como desafio, procure reproduzir um som sempre que ocorrer a colisão com a comida e outro quando ocorrer o “*game over*”.

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:



Sites

W3SCHOOLS HTML: The Language for Building Pages

<https://goo.gl/gLZ9v>

Sololearn – Everyone can code

<https://goo.gl/zAeEe8>

Codecademy

<https://goo.gl/OTm7rG>

Tutoriais MDN Mozilla

<https://goo.gl/QKTkc7>

Phaser Framework

<https://goo.gl/ij8P8H>

Referências

BUCHARD, E. **The Web Game Developer's Cookbook:** Using Java Script and HTML5 to Develop Games. USA: Addison-Wesley, 2013.

RAMTAL, Dev; DOBRE, Adrian. **Physics for JavaScript games, animation, and simulations:** with HTML5 canvas. New York: Apress, 2014.

SILVA, Mauricio Samy. **Html 5:** a linguagem de marcação que revolucionou a web. São Paulo: Novatec, 2011.

Sites Visitados

World Wide Web Consortium W3C. Disponível em: <<https://www.w3.org/html/>>. Acesso em: 9 jun. 2018.



Cruzeiro do Sul
Educacional