Curso de

Javascript e HTML: pratique lógica com desenhos, animações e um jogo

Desenhando gráficos com Canvas

001.geral do que veremos

Olá! Me chamo Flávio Almeida, e este treinamento é direcionado àqueles que terminaram a **primeira parte** do curso de Lógica de Programação e desejam praticar mais um pouco, para se aprimorarem nesta técnica.

Neste treinamento, aplicaremos o conceito de lógica de programação para criarmos desenhos em tela, como formas geométricas com diferentes cores, ou ainda, realizar pequenas animações em tela. Podemos também criar um jogo, como tiro ao alvo, onde temos um símbolo de alvo que se move pela tela e nosso objetivo é clicar sobre ele.

A ideia deste treinamento é utilizarmos conceitos de lógica, já aprendidos anteriormente, para aplicações diferentes.

Para fazermos isso, do zero, seria complicado. Sendo assim, as linguagens têm bibliotecas de instruções, em geral, que chamamos de **APIs**. Aprenderemos um pouco da API de gráficos do JavaScript para que possamos desenhar na tela.

Aplicaremos a lógica de programação para criar desenhos e animações, os "pincéis" e as "telas" são informações contidas nas bibliotecas. Vamos começar!

002.O canvas será nossa tela!

Nosso primeiro passo nesta aula será criarmos nosso `programa.html`.

Abriremos nosso editor de texto e salvaremos o arquivo em nossa área de trabalho, com o nome `programa.html`.

Aqui estamos utilizando o Sublime 2, mas você pode utilizar qualquer editor de sua preferência.

Lembrando que, no curso anterior de Lógica de Programação, vimos que quando queremos escrever um código em JavaScript utilizamos as tags `<meta charset=UTF-8">` e a `<script>` para podermos escrever nosso código.

Conforme vimos, tudo que está dentro da tag `<script>` pertence ao mundo JavaScript, e tudo que está fora - seja antes ou depois -, pertence ao mundo HTML.

Aprendemos no mundo JavaScript, a utilizar funções do mundo HTML por meio do `document`, por exemplo o `document.write()` que nos permite imprimir algo:

```
<meta charset="UTF-8">

<script>

document.write("");

</script>
```

Podemos, por exemplo, escrever a mensagem "Oi":

```
<meta charset="UTF-8">

<script>

document.write("<h1>Oi</h1>");

</script>
```

Salvaremos o programa e retornaremos ao navegador. Nele, escolheremos a opção de abrir arquivo, e selecionaremos o `programa.html`, assim, temos a seguinte exibição:

Oi

Entretanto, nosso objetivo não é escrever HTML no mundo HTML, em vez disso, queremos desenhar. Sendo assim, não utilizaremos o `document.write()`.

Para desenhar precisamos: de uma tela e de um pincel.

Vamos acessar a página do Google e digitar "*empty canvas*". A palavra "*empty*" significa "vazio", em Português.

Observando os resultados, vemos que são exibidas imagens de telas em branco. Os mundos HTML e JavaScript utilizam o Inglês, por isso, as tags estão neste idioma, "*canvas*" também é uma tag do mundo HTML.

Se inserirmos em nosso código a tag `<canvas>`, ela é válida do mundo HTML e tem a mesma finalidade de uma tela em branco, ou seja, uma área em que podemos desenhar.

Como vamos trabalhar com desenhos, não iremos utilizar textos, não será necessária a inclusão da tag `<meta charset="UTF-8">`, que serve para resolvermos problemas de acentuação.

Assim, temos o mínimo para podermos começar a programar:

```
<canvas></canvas>

<script>

</script>
```

Recapitulando: O *canvas* é uma área da tela onde podemos desenhar, escrever com um pincel.

Precisamos informar, em nosso programa, quanto o `<canvas>` ocupa de espaço. Para isso, utilizaremos dois atributos, o `width`, que em português é "largura", diremos que é 600, e o `height`, ou "altura", que será de 400.

```
<canvas width="600" height="400"> </canvas>
```

Se salvarmos e recarregarmos a página, nada acontece para nós, visualmente. Isso porque o `<canvas>` é branco, por padrão. Como o fundo do navegador também é, não conseguimos ver o que acabamos de criar.

Para que possamos visualizar, faremos com que ele ganhe cor. É o que veremos adiante.

003.Desenhando com um pincel

Nesta aula, aprenderemos a desenhar utilizando um pincel.

Nas aulas anteriores, aprendemos que o `document` possui uma função que nos permite utilizar algo do mundo HTML dentro do mundo JavaScript.

Para isso, criaremos a variável `tela`, e pediremos ao `document.querySelector()` o parâmetro `"canvas"` - note que está entre aspas:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector("canvas");

</script>
```

COPIAR CÓDIGO

JavaScript pode utilizar tanto as aspas ("), quanto apóstrofo ('), que nos referimos como **aspas simples**. É o que utilizaremos até o final do treinamento, apenas para simplificar a digitação.

Assim, o documento tem o `<canvas>` e, dentro do JS, pedimos para o `document` executar a função `querySelector()` para pegar este `<canvas>` e passá-lo como valor na variável `tela`.

Isso nos dá a área para escrever, mas e o pincel? Criaremos uma variável `pincel`, que receberá como valor a `tela`:

```
<canvas width="600" height="400"></canvas>

<script>

   var tela = document.querySelector("canvas");

   var pincel = tela.

</script>
```

A `tela`, assim como o `document` sabe buscar um elemento da página por meio da tag, e fornecer um retorno para a variável, poderá passar o `pincel`.

No mundo da programação podemos ter gráficos em 2D, 3D, 4D, enfim, portanto, precisamos informar à tela qual o tipo de pincel, ou seja, o contexto no qual escreveremos na tela. Para isso, utilizaremos o `getContext()` e passaremos o `2d` como parâmetro:

```
<canvas width="600" height="400"></canvas>

<script>

   var tela = document.querySelector("canvas");

   var pincel = tela.getContext('2d');

</script>
```

Já temos a `tela` e o `pincel`, portanto podemos escrever na tela.

Se o pincel foi dado pelo `<canvas>`, qualquer operação feita nele atuará sobre a tela.

Nossa primeira operação será desenhar um retângulo de dimensões `600x400`, para preencher todo o `<canvas>`. Para fazer isso, chamaremos o `pincel`, e pediremos para ele, com o operador `.`, preencher o retângulo.

Preencher em Inglês é "*fill*", enquanto "*rect*" é a abreviação de retângulo:

```
<canvas width="600" height="400"></canvas>

<script>

   var tela = document.querySelector("canvas");

   var pincel = tela.getContext('2d');

   pincel.fillRect();

</script>
```

Ou seja, estamos pedindo ao JavaScript para preencher um retângulo. Entretanto, precisamos informar ao programa o ponto da tela onde desejamos começar a preencher.

Assim como um pintor inicia sua pintura em um ponto específico da tela, nós também precisamos posicionar nosso pincel no `<canvas>` para podermos iniciar nosso preenchimento.

Como sabemos, na matemática trabalhamos com a ideia de **planos cartesianos**, onde temos um eixo x, na horizontal, e um eixo y, na vertical.

De acordo com as dimensões que especificamos, começaremos nosso preenchimento no canto superior esquerdo, isso significa que iniciaremos no ponto 0 tanto no eixo x quanto y.

No código, estas coordenadas são inseridas entre os parênteses da instrução `pincel.fillRect()`:

```
<canvas width="600" height="400"></canvas>

<script>

   var tela = document.querySelector("canvas");

   var pincel = tela.getContext('2d');
```

```
    pincel.fillRect(0, 0);

</script>
```

Dessa coordenada, queremos criar uma forma que tenha 600 de largura, e 400 de altura, portanto, inseriremos isto em nosso código:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector("canvas");

    var pincel = tela.getContext('2d');

    pincel.fillRect(0, 0, 600, 400);

</script>
```

Salvaremos o programa e recarregaremos a página, e vemos na tela um retângulo preto, com as dimensões 600 (largura) x 400 (altura), posicionando o pincel a partir do ponto mais extremo no canto superior esquerdo.

Como estamos utilizando o `fill`, ou "preencher", ele completa o restante da área com preto.

Só que, a cor preta é muito forte, colocaremos uma cor mais suave. Para indicar qual a "tinta", temos de indicar o `fillStyle`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector("canvas");

    var pincel = tela.getContext('2d');
```

```
    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle

</script>
```

Diferentemente de `fillRect()`, de `getContext()`, ou de `querySelector()`, o `fillStyle` **não é uma função**, e sim o que chamamos de **propriedade**. É equivalente a uma variável, por receber um valor, que é a cor. No caso, utilizaremos o 'lightgrey':

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector("canvas");

    var pincel = tela.getContext('2d');

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'lightgrey';

</script>
```

Salvaremos e recarregaremos a página. Entretanto o retângulo continua preto, por quê?

Pensando em uma situação real, se pedimos para alguém pintar algo, a pessoa precisará da tinta. No caso, estamos passando as informações de cor após termos pintado, sem termos fornecido a tinta.

Sendo assim, o `fillStyle` deve vir antes do `fillRect`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector("canvas");
```

```
    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

</script>
```

Salvaremos e recarregaremos. Deu certo! Temos nosso `<canvas>` em cinza claro. Qualquer desenho que fizermos respeitará a área que delimitamos nas dimensões.

Aumentaremos as dimensões, para `800` de largura e `600` de altura:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector("canvas");

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 800, 600);

</script>
```

Salvaremos e recarregaremos a página. Não aconteceu nada. Isso porque, por mais que tenhamos informado novas dimensões, o máximo que caberá em nosso `<canvas>` é `600x400`, portanto, respeitaremos isso.

004.Nossa primeira obra de arte. Será?

Nesta aula, daremos continuidade ao projeto da aula anterior, onde fizemos nosso primeiro desenho.

Nossa visualização será, no lado esquerdo, do `<canvas>` em sua totalidade, e, no lado direito, veremos nosso código.

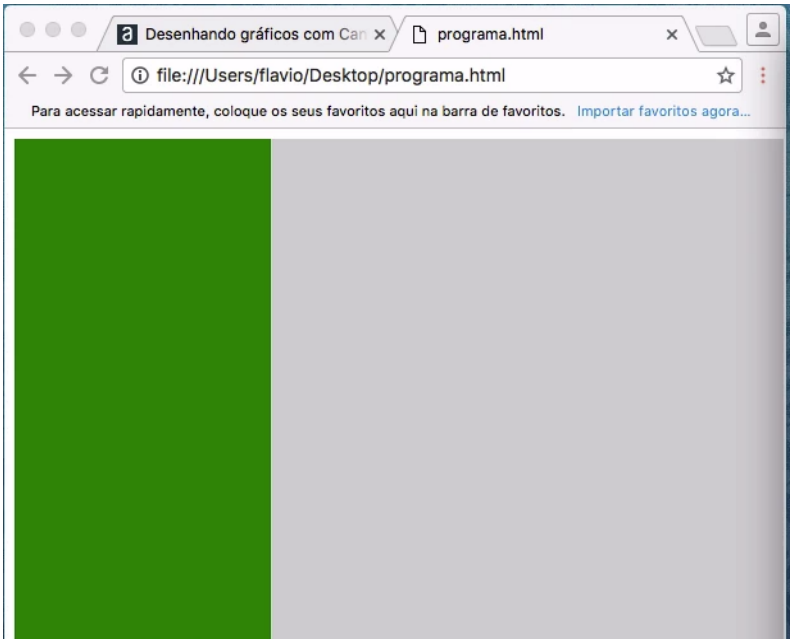
Criaremos um novo retângulo, que terá `200` de largura e `400` de altura, na cor verde. Primeiro, selecionaremos a nova cor, utilizando o `fillStyle`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'green';

</script>
```

Em seguida, pediremos ao `pincel` que se movimente em nossa tela, por meio do `fillRect` na posição `(0, 0, 200, 400)`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);

</script>
```

Salvaremos e recarregaremos a página. Temos em nossa tela a exibição de um retângulo menor, ao lado esquerdo, na cor verde, e o resto do `<canvas>` preenchido em cinza, algo análogo à representação abaixo:



O próximo passo será criar um novo retângulo, de mesmo tamanho, no lado direito, na cor vermelha. Para isso, criaremos um novo `pincel.fillStyle` e um novo `pincel.fillRect`.

Como queremos que ele comece em determinado ponto do retângulo, ou seja, do eixo x, precisamos representar isso no `fillRect`. Já que este lado mede `600`, e teremos três formas, cada uma delas terá `200` de largura, sendo assim, a terceira forma iniciará a partir do ponto `400`:

```
<canvas width="600" height="400"></canvas>

<script>

  var tela = document.querySelector('canvas');

  var pincel = tela.getContext('2d');

  pincel.fillStyle = 'lightgrey';

  pincel.fillRect(0, 0, 600, 400);
```

```
    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);

    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

</script>
```

Salvaremos e recarregaremos, teremos na tela um retângulo verde, como na imagem anterior, mas agora temos também um retângulo cinza ao seu lado direito, de mesmo tamanho e, a sua direita, um retângulo vermelho com as mesmas dimensões.

Comentaremos o bloco que contém as informações com a cor cinza do `<canvas>`, em JavaScript isso pode ser feito utilizando o comando `/*` indicando o ponto inicial do comentário, seguido do `*/` para finalizá-lo:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    /*

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    */

    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);
```

```
    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

</script>
```

Salvaremos e recarregaremos a página. Como podemos observar, não há mais a área cinza do `<canvas>`. Temos portanto a bandeira da Itália. Retornaremos com o código, para termos tudo funcionando:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);

    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

</script>
```

Salvaremos e recarregaremos.

Nosso próximo objetivo será desenhar um triângulo no centro do retângulo cinza, é o que veremos adiante.

005.A vida não é só feita de retângulos!

Nesta aula, aprenderemos a criar um **triângulo**.

Para a criação desta forma geométrica, não utilizaremos o `fillRect`, em vez disso, queremos começar um caminho. O que é isso?

Começar, em inglês, é *begin*, e caminho é *path*. Portanto, utilizaremos um *begin path* para dizermos qual será a direção seguida por nosso pincel. O primeiro passo é utilizar o `fillStyle`, com a cor amarela:

```
<canvas width="600" height="400"></canvas>

<script>

   var tela = document.querySelector('canvas');

   var pincel = tela.getContext('2d');

   pincel.fillStyle = 'lightgrey';

   pincel.fillRect(0, 0, 600, 400);

  pincel.fillStyle = 'green';

   pincel.fillRect(0, 0, 200, 400);

   pincel.fillStyle = 'red';

   pincel.fillRect(400, 0, 200, 400);

   pincel.fillStyle = 'yellow';

</script>
```

Com isso, indicaremos que nosso pincel iniciará seu caminho com o `pincel.beginPath()`:

```
<canvas width="600" height="400"></canvas>

<script>

   var tela = document.querySelector('canvas');
```

```
    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);

    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

    pincel.fillStyle = 'yellow';

    pincel.beginPath();

</script>
```

Para começarmos a escrever, indicaremos onde está o ponto inicial do nosso triângulo. Nossa primeira coordenada será centralizada na tela, para fazermos isso, utilizaremos a função `moveTo`:

```
<canvas width="600" height="400"></canvas>



<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'green';
```

```
    pincel.fillRect(0, 0, 200, 400);

    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

    pincel.fillStyle = 'yellow';

    pincel.beginPath();

    pincel.moveTo();

</script>
```

Pensando em nossas coordenadas, temos que posicionar nosso pincel exatamente no ponto 300, da coordenada X (que mede 600), e 200 na Y (que mede 400):

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);

    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

    pincel.fillStyle = 'yellow';
```

```
    pincel.beginPath();

    pincel.moveTo(300, 200);

</script>
```

Em seguida, moveremos o pincel para uma nova posição, com a função `lineTo()`. Esta posição será o ponto de encontro entre o verde e o cinza, e a linha inferior de nosso `<canvas>`, portanto, `200` no eixo X, e `400` no eixo Y:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);

    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

    pincel.fillStyle = 'yellow';

    pincel.beginPath();

    pincel.moveTo(300, 200);

    pincel.lineTo(200, 400);

</script>
```

O próximo movimento criará uma linha horizontal, do ponto que acabamos de criar até o ponto em que o cinza encontra o vermelho, de coordenadas 400 (X), e 400 (Y):

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);



    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);



    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);



    pincel.fillStyle = 'yellow';

    pincel.beginPath();

    pincel.moveTo(300, 200);

    pincel.lineTo(200, 400);

    pincel.lineTo(400, 400);
```

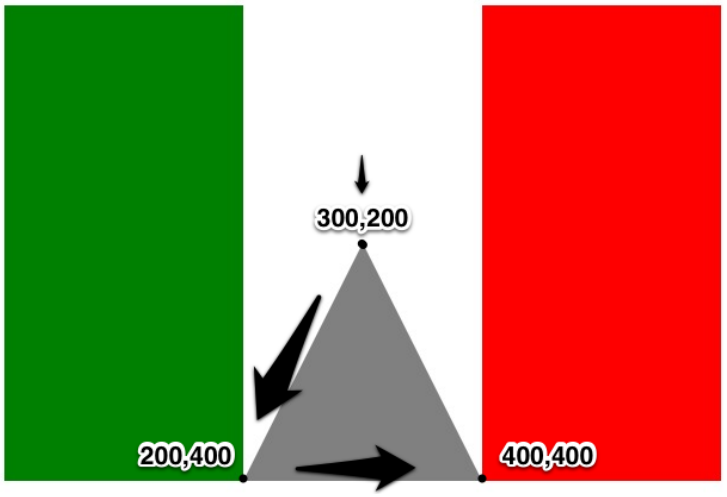
```
</script>
```

Por fim, basta inserirmos um comando para que o programa preencha esta forma, que é o `pincel.fill()`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);



    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);

    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

    pincel.fillStyle = 'yellow';

    pincel.beginPath();

    pincel.moveTo(300, 200);

    pincel.lineTo(200, 400);

    pincel.lineTo(400, 400);

    pincel.fill();
```

</script>

Salvaremos e recarregaremos a página. Temos nossa imagem, assim como antes, só que agora há um triângulo de cor cinza (apenas para ilustrar), com sua base no final do <canvas> e pico centralizado em relação a forma como um todo:

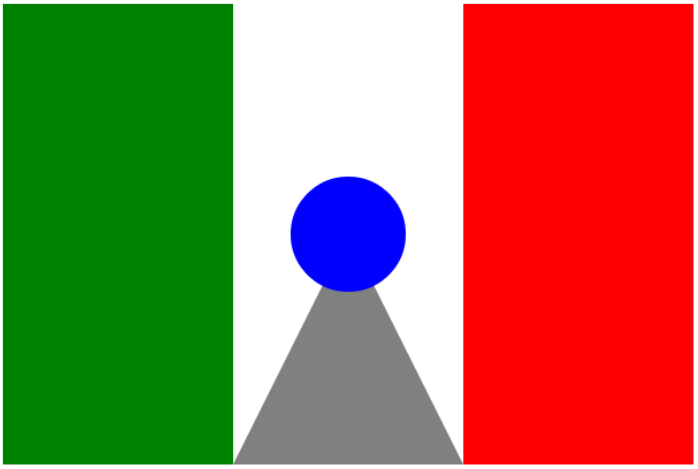


Como vimos, a construção do triângulo foi trabalhada a partir de um caminho, criamos o primeiro ponto, ou vértice, e em seguida os demais. Por último, fazemos com que o programa preencha esta forma com a cor que já escolhemos.

Nosso próximo desafio será criar uma esfera.

006.Tem espaço para círculo também?

Podemos fazer qualquer figura que tenha lados usando essa fórmula, não apenas triângulos. Mas e se precisarmos de algo arredondado? Vamos colocar uma circunferência azul no meio da nossa imagem, pra ficar assim:



Para começarmos, indicaremos um novo pincel, com o `fillStyle` na cor azul, e um novo caminho, com o `beginPath`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle = 'green';

    pincel.fillRect(0, 0, 200, 400);

    pincel.fillStyle = 'red';

    pincel.fillRect(400, 0, 200, 400);

    pincel.fillStyle = 'yellow';

    pincel.beginPath();

    pincel.moveTo(300, 200);
```

```
    pincel.lineTo(200, 400);

    pincel.lineTo(400, 400);

    pincel.fill();

    pincel.fillStyle = 'blue';

    pincel.beginPath();

</script>
```

Temos de iniciar um novo caminho, caso contrário, o programa entenderá que deve continuar o caminho anterior.

Com esta etapa concluída, partiremos para a utilização da função `arc()`, para traçarmos nossa esfera. Nela, incluiremos as seguintes informações:

- Posicionamento da esfera, que definiremos como 300 (X) e 200 (Y);
- Tamanho, ou seja, o raio - que definiremos como 50;
- O ângulo inicial, e o ângulo final, em radianos (multiplicado por PI - 3,14);

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgrey';

    pincel.fillRect(0, 0, 600, 400);
```

```
    pincel.fillStyle = 'green';
    pincel.fillRect(0, 0, 200, 400);
    pincel.fillStyle = 'red';
    pincel.fillRect(400, 0, 200, 400);
    pincel.fillStyle = 'yellow';
    pincel.beginPath();
    pincel.moveTo(300, 200);
    pincel.lineTo(200, 400);
    pincel.lineTo(400, 400);
    pincel.fill();
    pincel.fillStyle = 'blue';
    pincel.beginPath();
    pincel.arc(300, 200, 50, 0, 2 * 3.14);
</script>
```

Por enquanto, nosso foco não serão as questões matemáticas.

Em seguida, chamaremos a função `fill` para que a esfera seja preenchida:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
```

```
    pincel.fillStyle = 'lightgrey';
    pincel.fillRect(0, 0, 600, 400);
    pincel.fillStyle = 'green';
    pincel.fillRect(0, 0, 200, 400);
    pincel.fillStyle = 'red';
    pincel.fillRect(400, 0, 200, 400);
    pincel.fillStyle = 'yellow';
    pincel.beginPath();
    pincel.moveTo(300, 200);
    pincel.lineTo(200, 400);
    pincel.lineTo(400, 400);
    pincel.fill();
    pincel.fillStyle = 'blue';
    pincel.beginPath();
    pincel.arc(300, 200, 50, 0, 2 * 3.14);
    pincel.fill();
</script>
```

Salvaremos o programa e recarregaremos a página. Temos nossa esfera azul, exatamente no pico de nosso triângulo.

Para que pudéssemos construir esta imagem, precisamos conhecer a função `getContext()`, `fillRect()`, `lineTo()`, `beginPath()`, a propriedade `fillStyle`, a esse conjunto de

propriedades damos o nome de **APIs**, ou bibliotecas. Aqui, estamos manipulando a API específica para a criação de gráficos.

Para descobrirmos estas bibliotecas, temos que pesquisar pela documentação da linguagem na internet, utilizando os termos "canvas 2d api".

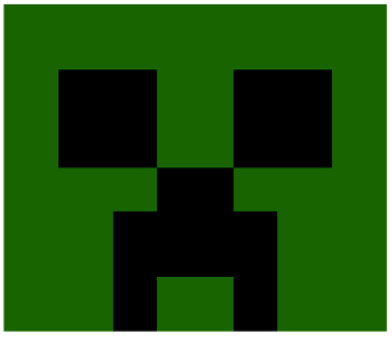
O resultado escolhido está disponível.

Clicamos no link das Web APIs, há materiais disponíveis sobre `fillRect`, `fillStyle`, entre outros. O que faremos é melhorar certos aspectos do nosso programa nas próximas aulas e organizar nosso código da melhor forma pois, se alguém que está iniciando na programação ler ele, não conseguirá identificar com clareza qual a sua função.

Adiante, organizaremos nosso código, para facilitar sua manutenção e legibilidade.

007.A cara do Creeper

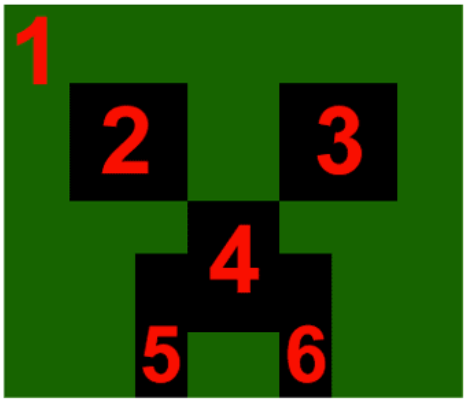
Vamos lá? O Johann é um menino de 11 anos e adora jogar Minecraft. Minecraft é aquele jogo onde você constrói um mundo com blocos, como se fosse um Lego virtual. Ele gosta tanto que pediu um pôster de uma das personagens principais do jogo: o Creeper. O Creeper na verdade é um monstro (que explode ao se aproximar de um jogador) e tem mais ou menos essa cara:



Será que você consegue (re)criar essa imagem e desenhar a cabeça do Creeper? Siga os passos a seguir :

- Criar um novo arquivo, por exemplo creeper.html.
- Definir a tag canvas.
- Criar um script com tela e pincel.
- Desenhar os retângulos através da função fillRect do pincel.
- Não esquecer de pintar os retângulos, aqui usamos as cores darkgreen e black.

Na tentativa de desenhar essa figura, uma observação importante: repare que esse desenho é composto de retângulos! São 6 retângulos:



Para facilitar um pouco mais, segue o tamanho (largura, altura) em pixels de cada retângulo:

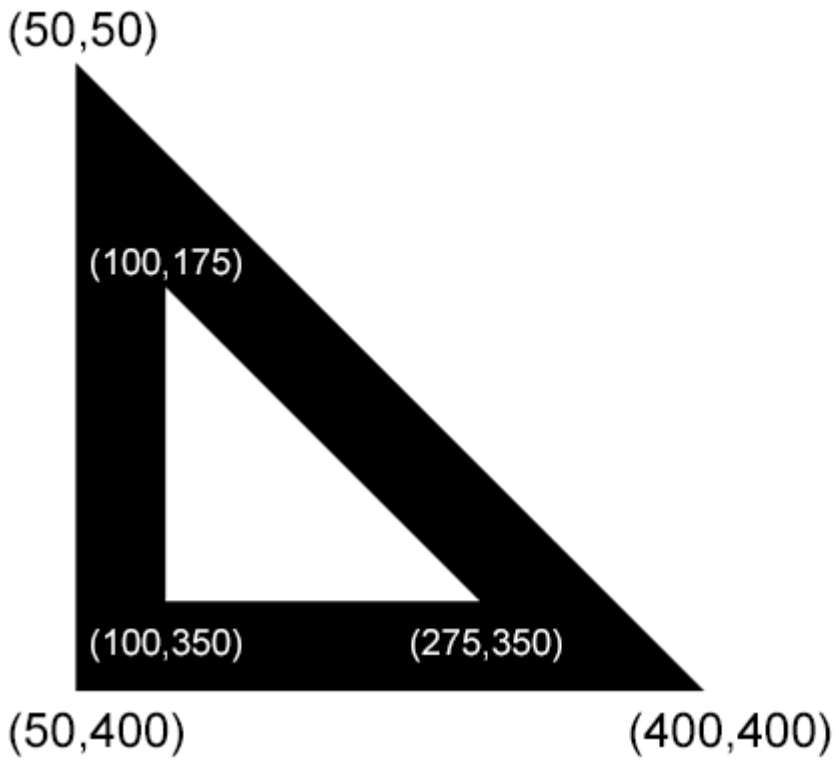
- Retângulo 1: 350, 300 (cabeça).
- Retângulos 2 e 3: 90, 90 (olhos).
- Retângulo 4: 70, 100 (nariz).
- Retângulos 5 e 6: 40, 110 (parte da boca).
- O seu canvas deve ter o tamanho de 600 x 400 pixels.

```
<canvas width="600" height="400"></canvas>

<script>
```

```
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    // cabeça
    pincel.fillStyle = 'darkgreen';
    pincel.fillRect(200,50,350,300);
    // olhos
    pincel.fillStyle = 'black';
    pincel.fillRect(250, 110, 90, 90);
    pincel.fillRect(410, 110, 90, 90);
    // nariz
    pincel.fillRect(340, 200, 70, 100);
    // boca ou barba
    pincel.fillRect(300, 240, 40, 110);
    pincel.fillRect(410, 240, 40, 110);
</script>
```

008.Esquadro

Já praticamos bastante com retângulos, agora é hora de desenhar uma outra figura. Nesse exercício a tarefa é **desenhar um esquadro**:



Repare que isso é nada mais do que dois triângulos, um dentro do outro. Lembrando também que desenhar um triângulo é diferente de um retângulo, pois é preciso desenhar linha por linha. Em outras palavras, a API é diferente!

Para desenhar é preciso inicializar um *path* e mover o pincel para uma posição:

```
var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');



pincel.beginPath();

pincel.moveTo(50, 50);
```

A partir daí podemos desenhar uma linha:

```
var tela = document.querySelector('canvas');
```

```
var pincel = tela.getContext('2d');

pincel.beginPath();

pincel.moveTo(50, 50);

pincel.lineTo(50, 400);
```

Tendo todas as linhas desenhadas, podemos preencher a figura:

```
pincel.fill();
```

Agora é com você!

Como ficará o código completo do esquadro?

Dica: Com as coordenadas em mãos é só mover o pincel para o lugar certo. É importante ressaltar que devemos começar um novo path para cada triângulo

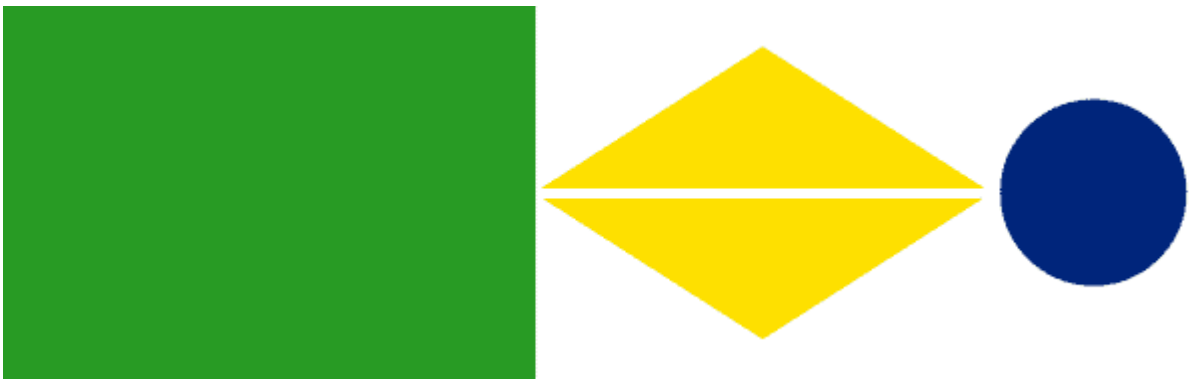
```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle='black';

    pincel.beginPath();

    pincel.moveTo(50, 50);

    pincel.lineTo(50, 400);

    pincel.lineTo(400, 400);

    pincel.fill();
```

```
    pincel.fillStyle='white';

    pincel.beginPath();

    pincel.moveTo(100, 175);

    pincel.lineTo(100, 350);

    pincel.lineTo(275, 350);

    pincel.fill();

</script>
```

009.Ordem e Progresso

Que tal desenhar a nossa bandeira? Tirando as estrelas e o dizer "Ordem e Progresso", sabemos tudo para desenhar a bandeira do Brasil:

Talvez você esteja pensando que não aprendemos a criar um losango, mas repare que o *losango* amarelo pode ser representado através de dois triângulos



Ou seja, temos um retângulo verde (`green`), dois triângulos amarelos (`yellow`) e um círculo azul escuro (`darkblue`).

Nos exercícios anteriores já praticamos como desenhar retângulos e triângulos. Agora só falta lembrar do círculo. Aqui também devemos começar um *path*:

```
var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle="darkblue";

pincel.beginPath();
```

E, para desenhar o círculo, usamos a função `arc`:

```
var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.beginPath();

pincel.arc(300, 200, 100, 0, 2*3.14);

pincel.fill();
```

Os primeiros dois parâmetros do método `arc` são as coordenadas X e Y do centro do círculo (no nosso caso, 300 e 200). O terceiro parâmetro é o valor do raio (no nosso caso, 100). O quarto e quinto parâmetros definem o ângulo inicial e final do

círculo. Como queremos desenhar um círculo completo, os parâmetros são, respectivamente, 0 e 2 * PI (cujo valor é 3.14).

Agora vamos desenhar nossa bandeira?

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle='green';

    pincel.fillRect(0, 0, 600, 400);

    pincel.fillStyle='yellow';

    pincel.beginPath();

    pincel.moveTo(300, 50);

    pincel.lineTo(50, 200);

    pincel.lineTo(550, 200);

    pincel.fill();

    pincel.beginPath();

    pincel.moveTo(300, 350);

    pincel.lineTo(50, 200);

    pincel.lineTo(550, 200);
```

```
    pincel.fill();

    pincel.fillStyle='darkblue';

    pincel.beginPath();

    pincel.arc(300, 200, 100, 0, 2*3.14);

    pincel.fill();

</script>
```

Extraindo funções

010.Repetir código não rola, não é mesmo?

Dando continuidade ao nosso trabalho com formas, criaremos um novo programa. Podemos salvá-lo como `programa2.html`.

Primeiramente, ele terá nossas tags `<script>` e `<canvas>`:

```
<canvas width="600" height="400"></canvas>

<script>

</script>
```

Como sabemos, temos que transportar o `<canvas>` para o JavaScript, e faremos isso por meio do `document.querySelector()`:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');
```

```
</script>
```

Importante ressaltar que estamos utilizando as "aspas simples" (apóstrofo) para escrevermos menos, e que isso **só pode acontecer no mundo JavaScript**, que aceita tanto as aspas simples quanto as duplas (").

Para podermos escrever na tela, precisaremos da variável `pincel`, que receberá um `getContext` para nos fornecer um pincel 2D:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

</script>
```

Nosso próximo objetivo será imprimir um quadrado pequeno.

O primeiro passo é determinar a cor, com o `fillStyle`, que será verde:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

</script>
```

O segundo passo será indicar **o que** será preenchido em verde, ou seja, nosso quadrado. Para isso, utilizaremos o`fillRect` e passaremos as coordenadas da forma como **parâmetros**:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

</script>
```

Salvaremos nosso programa. Abriremos o navegador e selecionaremos nosso arquivo, a partir do menu "Arquivo > Abrir arquivo...". Ao fazermos isso, temos a imagem do nosso quadrado.

E se quisermos colocar um outro quadrado ao lado deste que já existe? Podemos utilizar a mesma fórmula, mas precisamos ter cuidado com seu posicionamento no eixo X, para que não coincida com o que já temos.

Se nosso quadrado tem `50` de largura, isso significa que ele ocupa 50 pixels no eixo X, assim, nosso novo quadrado deve começar a partir do ponto 50 no eixo X:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.fillRect(50, 0, 50, 50);
```

```
</script>
```

Criaremos, ainda, um terceiro, que iniciará no ponto 100 do eixo X:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.fillRect(50, 0, 50, 50);

pincel.fillRect(100, 0, 50, 50);

</script>
```

Salvaremos e recarregaremos a página, e o que temos são os três quadrados, verdes, um ao lado do outros. Entretanto, para quem vê, parece ser um único retângulo posicionado horizontalmente. Colocaremos as bordas, para separar cada um dos quadrados.

Após fazermos o `fillRect`, podemos indicar ao `pincel` o tipo de `strokeStyle`, qual será a cor da borda, que no caso definiremos como preta:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';
```

```
pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.fillRect(50, 0, 50, 50);

pincel.fillRect(100, 0, 50, 50);

</script>
```

Em seguida, indicaremos ao `pincel` que ele deve desenhar a borda, utilizando o `strokeRect()`. A borda deve ser inserida na mesma posição do quadrado:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

pincel.fillRect(50, 0, 50, 50);

pincel.fillRect(100, 0, 50, 50);

</script>
```

COPIAR CÓDIGO

Salvaremos o programa e recarregaremos a página. O primeiro quadrado agora tem borda, enquanto os outros dois permanecem sem divisões ou delimitações.

Repetiremos o `strokeRect` para os demais quadrados, lembrando que devemos replicar as respectivas posições:

```
<canvas width="600" height="400"></canvas>



<script>



var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');



pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

pincel.fillRect(50, 0, 50, 50);

pincel.strokeRect(50, 0, 50, 50);

pincel.fillRect(100, 0, 50, 50);

pincel.strokeRect(100, 0, 50, 50);

</script>
```

Salvaremos e recarregaremos o programa. Temos a imagem de três quadrados verdes, com bordas pretas, um ao lado do outro.

Vamos imaginar que nossa intenção fosse criar apenas um quadrado, nosso código ficaria então da seguinte forma:

```
<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

</script>
```

É bastante código para desenhar apenas uma forma simples. Um iniciante em programação nem seria capaz de identificar, com facilidade, a finalidade deste código.

Idealmente, seria interessante se pudéssemos, com apenas uma linha de código, atingir esta mesma funcionalidade, por exemplo:

```
<script>

//outras propriedades...

desenhaQuadradoVerde();

</script>
```

COPIAR CÓDIGO

Qualquer um que analisasse nosso código, saberia identificar o que esta linha faz - ela **desenha um quadrado verde**. Como aprendemos no módulo anterior, é possível fazermos isso por meio do uso de **funções**.

A **função** é um código que foi guardado para ser chamado posteriormente. Criaremos portanto a função `desenhaQuadradoVerde()`:

```
<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde() {

}

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

desenhaQuadradoVerde();

</script>
```

O conteúdo da nossa função será todo o bloco que cria nosso quadrado:

```
<canvas width="600" height="400"></canvas>

<script>

  function desenhaQuadradoVerde() {
```

```
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
        pincel.fillStyle = 'green';
        pincel.fillRect(0, 0, 50, 50);
        pincel.strokeStyle = 'black';
        pincel.strokeRect(0, 0, 50, 50);
}
desenhaQuadradoVerde();
</script>
```

Como já havíamos pré-definido, chamamos a função, sem esquecer dos parênteses (()).

Salvaremos e recarregaremos o programa. Ele continua funcionando, ainda temos nosso quadrado verde com borda preta. Entretanto, se agora quisermos criar um novo quadrado basta chamarmos a função novamente, e assim por diante, para quantos quadrados quisermos criar.

Por exemplo, para criarmos três quadrados chamaremos a função três vezes:

```
<canvas width="600" height="400"></canvas>
<script>
   function desenhaQuadradoVerde() {
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
        pincel.fillStyle = 'green';
```

```
        pincel.fillRect(0, 0, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(0, 0, 50, 50);

}

desenhaQuadradoVerde();

desenhaQuadradoVerde();

desenhaQuadradoVerde();

</script>
```

Salvaremos e recarregaremos o programa. Entretanto, continuamos vendo apenas um quadrado! Por que isso aconteceu?

Isso ocorre porque todas as vezes que chamamos a mesma função, teremos os mesmos parâmetros, assim, os quadrados serão criados exatamente no mesmo lugar! Não é essa a intenção, queremos desenhar os quadrados lado a lado. É o que veremos adiante.

011.fillStroke ou strokeStyle?

Revisar faz parte do processo de aprendizagem.

Quando não definimos a cor do stroke a cor padrão utilizada é `black`. No entanto, para mudarmos a cor do stroke para uma cor diferente de preto, precisamos realizar essa operação através da propriedade `strokeStyle`. Exemplo:

```
pincel.strokeStyle = 'red'
```

O código acima adotará a cor vermelha para o stroke.

Você pode consultar a documentação do MDN no link:
CanvasRenderingContext2D

012.Uma função mais genérica

Nesta aula, veremos como podemos deixar nossa função genérica, para que possa funcionar para diferentes objetos.

O `fillRect` e o `strokeRect` não poderão ser o mesmo para todos, deverá receber parâmetros diferentes para cada objeto que o usuário deseje criar. Faremos com que ela seja capaz de aceitar o parâmetro `x`, que será aceito em `fillRect()` e `strokeRect()`:

```
<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadradoVerde(x) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = 'green';

        pincel.fillRect(x, 0, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, 0, 50, 50);

}

desenhaQuadradoVerde();

desenhaQuadradoVerde();

desenhaQuadradoVerde();



</script>
```

Assim, da próxima vez que chamarmos a função `desenhaQuadradoVerde()` faremos com que o primeiro inicie em `0`, o segundo em `50`, e o terceiro em `100`:

```
<canvas width="600" height="400"></canvas>



<script>

   function desenhaQuadradoVerde(x) {

       var tela = document.querySelector('canvas');

       var pincel = tela.getContext('2d');

       pincel.fillStyle = 'green';

       pincel.fillRect(x, 0, 50, 50);

       pincel.strokeStyle = 'black';

       pincel.strokeRect(x, 0, 50, 50);

}

desenhaQuadradoVerde(0);

desenhaQuadradoVerde(50);

desenhaQuadradoVerde(100);

</script>
```

Salvaremos e recarregaremos. Temos assim os três quadrados, lado a lado.

Recapitulando:

Ao chamarmos a função desenhaQuadradoVerde() e passarmos 0 como parâmetro, este número ocupa o parâmetro x e é utilizado na coordenada X do fillRect e do strokeRect. Na próxima chamada da função, o parâmetro passado será 50, e este será o número considerado no fillRect e no strokeRect, e assim sucessivamente.

Com isso, conseguimos escrever um código genérico, para desenhar o quadrado verde.

Mas e se quisermos alinhar os quadrados não mais no eixo X, e sim no eixo Y? Nesta hipótese, além do x, teremos também o parâmetro y:

```
<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadradoVerde(x, y) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = 'green';

        pincel.fillRect(x, 0, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, 0, 50, 50);

}

desenhaQuadradoVerde(0);

desenhaQuadradoVerde(50);

desenhaQuadradoVerde(100);

</script>
```

Portanto, substituiremos os `0` por `y` em nosso código construtor e, no chamamento da função, teremos que passar dois parâmetros, da seguinte forma:

```
<canvas width="600" height="400"></canvas>

<script>

   function desenhaQuadradoVerde(x, y) {

       var tela = document.querySelector('canvas');

       var pincel = tela.getContext('2d');

       pincel.fillStyle = 'green';

       pincel.fillRect(x, y, 50, 50);

       pincel.strokeStyle = 'black';

       pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadradoVerde(0, 0);

desenhaQuadradoVerde(0, 50);

desenhaQuadradoVerde(0, 100);

</script>
```

Salvaremos e recarregaremos o programa. Temos todos os quadrados alinhados verticalmente, no lado esquerdo da tela.

E se agora quisermos criar um quadrado vermelho? Podemos copiar o código da função que já temos e realizar algumas alterações.

Primeiro, alteraremos o nome dela, para `desenhaQuadradoVermelho`, e onde temos `'green'` alteraremos para `'red'`:

```
<canvas width="600" height="400"></canvas>


<script>
    function desenhaQuadradoVerde(x, y) {
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
       pincel.fillStyle = 'green';
        pincel.fillRect(x, y, 50, 50);
        pincel.strokeStyle = 'black';
        pincel.strokeRect(x, y, 50, 50);
}
    function desenhaQuadradoVermelho(x, y) {
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
        pincel.fillStyle = 'red';
        pincel.fillRect(x, y, 50, 50);
        pincel.strokeStyle = 'black';
        pincel.strokeRect(x, y, 50, 50);
}
desenhaQuadradoVerde(0, 0);
```

```
desenhaQuadradoVerde(0, 50);

desenhaQuadradoVerde(0, 100);


</script>
```

Em seguida, alteraremos um dos chamados da função para `desenhaQuadradoVermelho`:

```
<canvas width="600" height="400"></canvas>

<script>

  function desenhaQuadradoVerde(x, y) {

      var tela = document.querySelector('canvas');

      var pincel = tela.getContext('2d');

      pincel.fillStyle = 'green';

      pincel.fillRect(x, y, 50, 50);

      pincel.strokeStyle = 'black';

      pincel.strokeRect(x, y, 50, 50);

}

  function desenhaQuadradoVermelho(x, y) {

      var tela = document.querySelector('canvas');

      var pincel = tela.getContext('2d');

      pincel.fillStyle = 'red';

      pincel.fillRect(x, y, 50, 50);
```

```
        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadradoVerde(0, 0);

desenhaQuadradoVermelho(0, 50);

desenhaQuadradoVerde(0, 100);

</script>
```

Salvaremos e recarregaremos a página. Assim, temos dois quadrados verdes nas pontas, e um vermelho ao meio.

E se quisermos quadrados de outras cores? Podemos passar a cor também como um parâmetro, assim não precisaremos alterar todas as vezes que quisermos um quadrado com uma nova cor.

Assim, a função passará a se chamar apenas `desenhaQuadrado`:

```
<canvas width="600" height="400"></canvas>



<script>


   function desenhaQuadrado(x, y) {


        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');
```

```
        pincel.fillStyle = 'green';
        pincel.fillRect(x, y, 50, 50);
        pincel.strokeStyle = 'black';
        pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadrado(0, 0);
desenhaQuadrado(0, 50);
desenhaQuadrado(0, 100);
</script>
```

Primeiro, teremos que passar um parâmetro a mais, que será a cor:

```
<canvas width="600" height="400"></canvas>
<script>
    function desenhaQuadrado(x, y) {
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
        pincel.fillStyle = 'red';
        pincel.fillRect(x, y, 50, 50);
```

```
        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadrado(0, 0, 'blue');

desenhaQuadrado(0, 50, 'red');

desenhaQuadrado(0, 100, 'yellow');

</script>
```

Em seguida, teremos que informar à função que ela receberá mais um parâmetro, na própria construção da função, que chamaremos de `cor`, e em vez de termos a cor no `fillStyle`, substituiremos também por `cor`:

```
<canvas width="600" height="400"></canvas>

<script>

   function desenhaQuadrado(x, y, cor) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = cor;

        pincel.fillRect(x, y, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadrado(0, 0, 'blue');
```

```
desenhaQuadrado(0, 50, 'red');

desenhaQuadrado(0, 100, 'yellow');

</script>
```

Salvaremos e recarregaremos a página. Temos a exibição dos três quadrados alinhados verticalmente, o primeiro na cor azul, em seguida na cor vermelha, e por último na cor amarela.

Como podemos observar, a ideia de função é importante para economizarmos linhas de código, e deixá-lo mais legível.

Retomaremos o desenho com os quadrados verdes, alinhados horizontalmente, no eixo X:

```
<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadrado(x, y, cor) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = cor;

        pincel.fillRect(x, y, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadrado(0, 0, 'green');

desenhaQuadrado(50, 0, 'red');
```

```
desenhaQuadrado(100, 0, 'green');

</script>
```

Salvaremos e recarregaremos, confirmando a linha horizontal com três quadrados verdes.

013.Como eu amo esse tal de loop!

(Para esse exercício, foi copiado o arquivo programa2.html e criado o arquivo programa2a.html)

Agora que aprendemos a criar uma função genérica, proponho uma pergunta: e se quisermos preencher `<canvas>` com quadrados?

Como nosso tela tem 600 de largura, teríamos que colar linha por linha, de `50` em `50`, até completarmos 600 - seriam 30 linhas de código. Mas nós temos um padrão, em que a coordenada `X` deve ser repetida de `50` em `50` no código.

Como sabemos, podemos trabalhar com estruturas de repetição, chamadas de **laços**. Um deles é o **while**, que funciona enquanto alguma condição for verdadeira. A sua função será inserida em um bloco, entre chaves (`{}`):

```
<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadrado(x, y, cor) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = cor;

        pincel.fillRect(x, y, 50, 50);
```

```
        pincel.strokeStyle = 'black';
        pincel.strokeRect(x, y, 50, 50);
    }
    while() {
        desenhaQuadrado(0, 0, 'green');
    }
</script>
```

Enquanto a condição do `while` for verdadeira, o `desenhaQuadrado` será repetido.

Primeiro, declararemos o a variável `x`, que iniciará em `0`:

```
<canvas width="600" height="400"></canvas>
<script>
    function desenhaQuadrado(x, y, cor) {
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
        pincel.fillStyle = cor;
        pincel.fillRect(x, y, 50, 50);
        pincel.strokeStyle = 'black';
        pincel.strokeRect(x, y, 50, 50);
    }
```

```
    var x = 0;

    while() {

        desenhaQuadrado(0, 0, 'green');

    }

</script>
```

A condição de validade será que o `while` deve ser repetido enquanto `x` for menor que `600`. Então, passaremos como parâmetro a própria variável `x`:

```
<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadrado(x, y, cor) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = cor;

        pincel.fillRect(x, y, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, y, 50, 50);

}

    var x = 0;

    while(x < 600) {
```

```
        desenhaQuadrado(x, 0, 'green');
    }
</script>
```

Para podermos passar para o próximo quadrado, temos que avançar `50` pixels, assim, diremos que, ao ser executada a primeira função, `x` será acrescido de `50`:

```
<canvas width="600" height="400"></canvas>
<script>
    function desenhaQuadrado(x, y, cor) {
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
        pincel.fillStyle = cor;
        pincel.fillRect(x, y, 50, 50);
        pincel.strokeStyle = 'black';
        pincel.strokeRect(x, y, 50, 50);
    }
    var x = 0;
    while(x < 600) {
        desenhaQuadrado(x, 0, 'green');
        x = x + 50;
```

```
    }

</script>
```

Todas as vezes que o *loop* for executado, `x` será acrescido de `50`, até o limite de `600`.

Para criarmos, abaixo da linha de quadrados verdes, uma linha de quadrados vermelhos, basta declararmos a função `desenhaQuadrado()` novamente, com um parâmetro `y` correspondente à nova posição:

```
<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadrado(x, y, cor) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = cor;

        pincel.fillRect(x, y, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, y, 50, 50);

    }

    var x = 0;

    while(x < 600) {

    desenhaQuadrado(x, 0, 'green');

    desenhaQuadrado(x, 50, 'red');
```

```
    x = x + 50;

    }

</script>
```

Salvaremos e recarregaremos a página. Podemos ver uma linha de quadrados verdes, e abaixo, uma linha de quadrados vermelhos. Podemos fazer o mesmo processo com a cor azul:

```
<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadrado(x, y, cor) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = cor;

        pincel.fillRect(x, y, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, y, 50, 50);

    }

    var x = 0;

    while(x < 600) {

    desenhaQuadrado(x, 0, 'green');

    desenhaQuadrado(x, 50, 'red');
```

```
    desenhaQuadrado(x, 100, 'blue');

    x = x + 50;

  }

</script>
```

Ao salvarmos e recarregarmos as páginas, veremos as três linhas de quadrados. Temos que prestar atenção aos valores do eixo Y, pois se houver dois iguais, prevalecerá o que foi desenhado por último.

Podemos obter este mesmo resultado utilizando um `for`. Deixaremos o método `while()` comentado. O trecho será destacado com `*/`:

```
<canvas width="600" height="400"></canvas>

<script>

  function desenhaQuadrado(x, y, cor) {

      var tela = document.querySelector('canvas');

      var pincel = tela.getContext('2d');

      pincel.fillStyle = cor;

      pincel.fillRect(x, y, 50, 50);

      pincel.strokeStyle = 'black';

      pincel.strokeRect(x, y, 50, 50);

  }

  /*
```

```
    var x = 0;



    while(x < 600) {

        desenhaQuadrado(x, 0, 'green');

    x = x + 50;

    }

    */

</script>
```

O `for` nos permite criar uma variável internamente, portanto nossa `var x = 0` estará inserida nele, assim como a condição `x < 600`, e o incremento `x = x + 50`:

```
<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadrado(x, y, cor) {

        var tela = document.querySelector('canvas');

        var pincel = tela.getContext('2d');

        pincel.fillStyle = cor;

        pincel.fillRect(x, y, 50, 50);

        pincel.strokeStyle = 'black';

        pincel.strokeRect(x, y, 50, 50);
```

```
}

  /*

  var x = 0;

  while(x < 600) {

      desenhaQuadrado(x, 0, 'green');

  x = x + 50;

  }

 */

  for(var x = 0; x < 600; x = x + 50) {

  desenhaQuadrado(x, 0, 'green');

  }

</script>
```

COPIAR CÓDIGO

Salvaremos e recarregaremos a página. Temos o mesmo resultado que tivemos com o `while`, uma linha com quadrados verdes. Portanto, utilizamos as duas estruturas de repetição possíveis para conseguir um mesmo resultado, com foco organizacional para o nosso código.

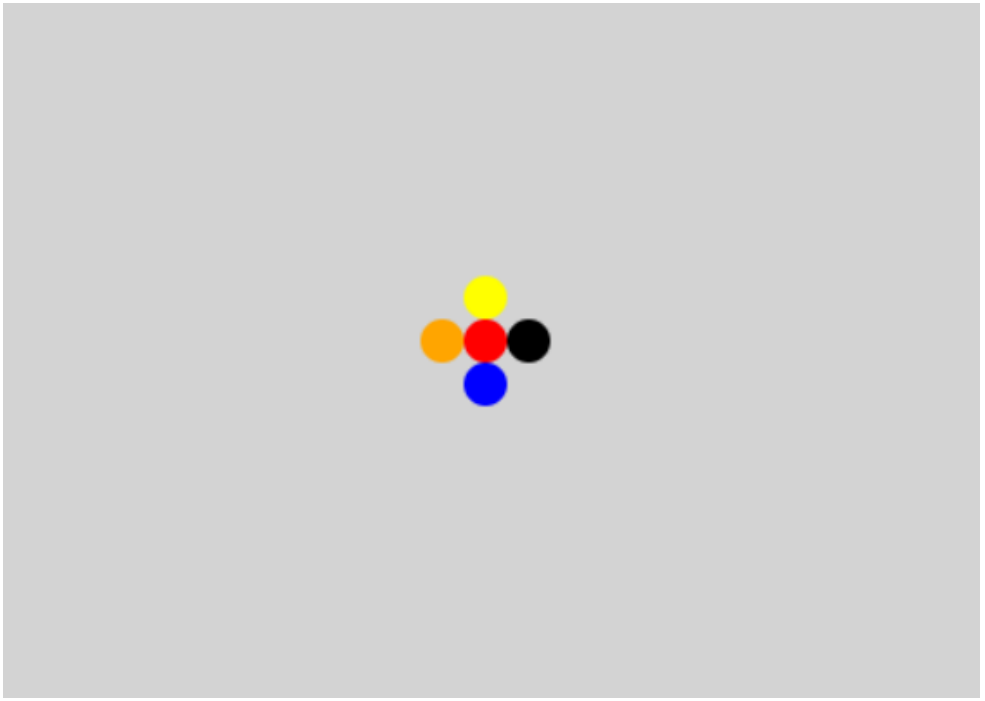
014.Uma inofensiva flor…

Temos o seguinte esboço de código que declara a função `desenhaCirculo`. Essa função permite desenhar na tela um círculo no eixo X e Y, inclusive permite ainda definir a sua cor:

```
<canvas width="600" height="400"></canvas>
```

```
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2*3.14);
        pincel.fill();
    }
</script>
```

A função `desenhaCirculo` ainda não é usada. **Faça uso da função para desenhar uma flor conforme a imagem abaixo:* *



Utilize como ponto de referência para o centro da flor (círculo vermelho) as coordenadas 300 (x) e 200 (y).

```
<canvas width="600" height="400"></canvas>
```

```
<script>
```

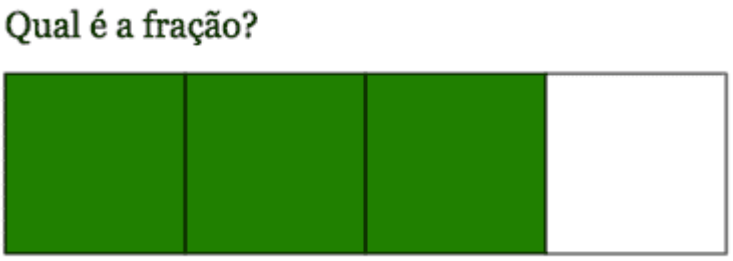
```
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio, cor) {
```

```
        pincel.fillStyle = cor;
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }

    function desenhaFlor(x, y) {

        desenhaCirculo(x, y+20, 10, 'blue');
        desenhaCirculo(x, y, 10, 'red');
        desenhaCirculo(x, y-20, 10, 'yellow');
        desenhaCirculo(x-20, y, 10, 'orange');
        desenhaCirculo(x+20, y, 10, 'black');
    }

    desenhaFlor(300, 200);

</script>
```

Você desenhou a nossa flor! Perceba que o círculo vermelho é o centro da nossa flor e ele está posicionado em 300 (x), 200 (y). Veja que os demais círculos se posicionam 20 pixels do círculo vermelho, tanto acima/abaixo ou direita/esquerda, a distância é sempre a mesma. Sendo assim, podemos indicar a posição x e y de onde o centro da flor será plotado/desenhado e dentro da função ajustar esse valor somando/subtraindo 20 pixels.

015.Frações

Acabei de conversar com uma professora de matemática do ensino fundamental. Ela me disse que os alunos estão com dificuldades de aprender frações por falta de exercícios. Para ensinar o conceito de uma fração, são utilizados desenhos parecidos com abaixo:



Para conseguirmos reproduzir a imagem com o exemplo da professora, vamos verificar algumas dicas e requisitos para construirmos nosso código:

Escreva uma função `desenhaQuadrado`, que recebe *x*, y, o tamanho e a cor. Cada quadrado desenhado deve utilizar um stroke.

```
function desenhaQuadrado(x, y, tamanho, cor) {

    // aqui precisamos usar fillRect, strokeRect, etc

}
```

Ah, não custa nada explicar rapidamente como desenhar um texto. Para tal existe a função `fillText()`, que recebe o texto e as coordenadas no ponto em que deve aparecer:

```
pincel.font='20px Georgia';
```

```
pincel.fillStyle='black';

pincel.fillText("Qual é a fração?", 50, 30);
```

Assim até podemos criar uma segunda função para desenhar o texto:

```
function desenhaTexto(texto, x , y) {
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');


    pincel.font='20px Georgia';
    pincel.fillStyle='black';
    pincel.fillText(texto, x, y);
}

desenhaTexto("Qual é a fração?", 50, 30);
```

Com base no modelo de desenho e as dicas de código acima, vamos ajudar a professora?

O desenho da fração para ajudar a professora é:

```
<meta charset="UTF-8">

<canvas width="700" height="500"></canvas>

<script>
    function desenhaQuadrado(x, y, tamanho, cor) {
        var tela = document.querySelector('canvas');
```

```
        var pincel = tela.getContext('2d');
        pincel.fillStyle=cor;
        pincel.fillRect(x,y, tamanho, tamanho);
        pincel.strokeStyle='black';
        pincel.strokeRect(x,y, tamanho, tamanho);
    }
    function desenhaTexto(texto, x , y) {
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
        pincel.font='20px Georgia';
        pincel.fillStyle='black';
        pincel.fillText(texto, x, y);
    }
    desenhaTexto("Qual é a fração?", 50, 30);
    var y = 50;
    desenhaQuadrado(50, y, 100, 'green');
    desenhaQuadrado(150, y, 100, 'green');
    desenhaQuadrado(250, y, 100, 'green');
    desenhaQuadrado(350, y, 100, 'white');
```

</script>

O código segue os requisitos e necessidades da professora. Você desenhou os quadrados para representar as frações e a função para trabalhar os textos. É um código legível e organizado. Continue praticando!

016.Gráfico de barras

Através de gráficos podemos expressar visualmente dados ou valores numéricos, e assim facilitar a nossa compreensão. Gráficos ajudam a entender o relacionamento entre valores e nos ajudam a analisar dados para chegarmos em determinadas conclusões.

Existem vários tipos de gráficos, um bastante famoso é o **gráfico de barras**, que é o objeto de estudo desse exercício.

Para alimentarmos nosso infográfico, procuramos em alguns relatórios da Alura quais navegadores os nossos alunos mais utilizavam em 2015 e 2016, e separamos os dados:

- Em 2015: 50% usava Chrome, 25% Firefox, 20% Safari, 5% Outros (Opera, IE, etc).
- Em 2016: 65% usava Chrome, 20% Firefox, 13% Safari, 2% Outros (Opera, IE, Edge, etc).

Ou seja, no ano de 2015, **50%** dos nossos alunos usaram o navegador Chrome, 25% Firefox** e assim em diante. Já em 2016, **65%** usavam **Chrome** e apenas **20% Firefox**.

Um gráfico de barras nada mais é do que um monte de retângulos representando os valores, algo assim:

Esse gráfico ainda está incompleto, pois falta uma legenda que mostraria que a cor azul representa *Chrome*, verde é *Firefox*, amarelo *Safari* e vermelho *Outros*. Mesmo assim, já é algo interessante para praticar.

Você já aprendeu a desenhar um retângulo e até criamos uma função com esse propósito. Desenhar um texto também já sabemos, mas vamos relembrar com o código abaixo:

```
<canvas width="600" height="400"></canvas>

<script>

   function desenhaRetangulo(x, y, largura, altura, cor) {

      var tela = document.querySelector('canvas');

      var pincel = tela.getContext('2d');

      pincel.fillStyle=cor;

      pincel.fillRect(x,y, largura, altura);

      pincel.strokeStyle='black';

      pincel.strokeRect(x,y, largura, altura);

   }

   function desenhaTexto(x, y, texto) {
```

```
        var tela = document.querySelector('canvas');
        var pincel = tela.getContext('2d');
        pincel.font='15px Georgia';
        pincel.fillStyle='black';
        pincel.fillText(texto, x, y);
    }
    // aqui vem mais
</script>
```

Também já vimos como representar vários valores dentro de um `Array`. Assim podemos guardar o percentual que corresponde a cada ano. No mundo de gráficos, os valores são chamados de **série**:

```
var serie2015 = [50, 25, 20, 5];
var serie2016 = [65, 20, 13, 2];
```

Cada valor no array representa ao percentual (%). No ano 2016 (ou melhor, na série 2016), 65% dos alunos usavam Chrome, 20% Firefox etc.

Dessa maneira, assim como podemos representar o percentual de cada ano, também podemos apresentar todas as cores utilizadas no gráfico de barras. Repare que o nosso gráfico possui 4 cores, que são “azul”, “verde”, “amarelo” e “vermelho”. Nesse sentido, é possível criar uma lista com as respectivas cores:

```
var cores = ['blue', 'green', 'yellow', 'red'];
```

Tendo isso em mãos vem agora a sua tarefa:** Escreva uma função** `desenhaBarra` que cria uma barra com as seguintes especificações: 4 retângulos - o primeiro azul, o segundo verde, etc., como apresentado na imagem acima.

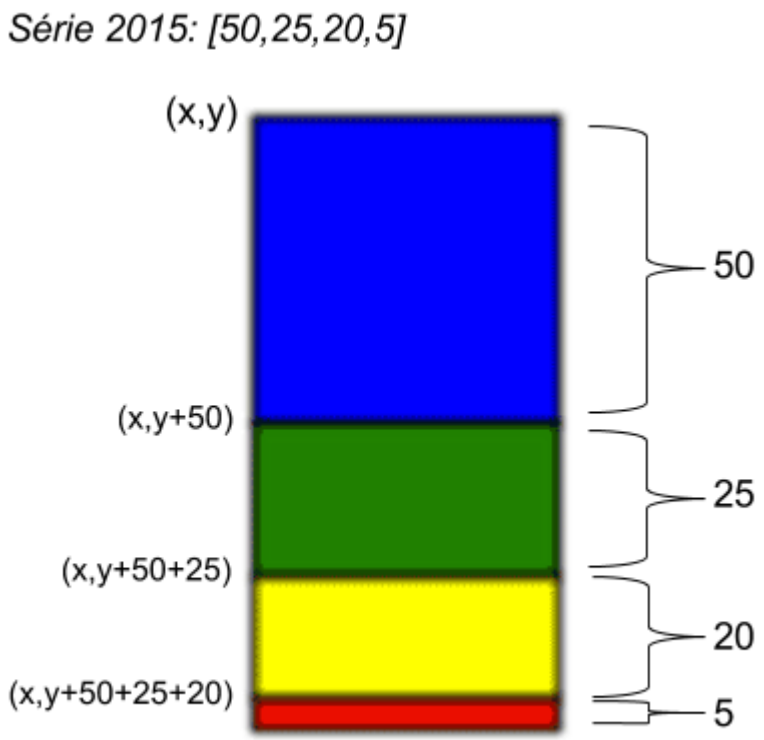
A chamada da função deve ser:

```
desenhaBarra(50, 50, serie2015, cores, '2015');

desenhaBarra(150, 50, serie2016, cores, '2016');
```

A primeira barra começa com `x=50` e `y=50`, recebe a série de 2015, as cores e o texto '2015'. Baseado nisso já podemos escrever a função:

```
function desenhaBarra(x, y, serie, cores, texto) {

   // aqui precisamos desenhar vários retangulas!

}
```

Dentro da função você precisa implementar um laço e nele precisa recuperar o valor da série, que nada mais é do que a altura do retângulo:



Agora é com você!

Implemente a função `desenhaBarra` **e chame-a para desenhar duas barras.**

O desenho deverá ficar dessa maneira:

O código completo é:

```
<canvas width="600" height="400"></canvas>

<script>

   function desenhaRetangulo(x, y, largura, altura, cor) {

       var tela = document.querySelector('canvas');

       var pincel = tela.getContext('2d');

       pincel.fillStyle=cor;

       pincel.fillRect(x,y, largura, altura);

       pincel.strokeStyle='black';

       pincel.strokeRect(x,y, largura, altura);

   }

   function desenhaTexto(x , y, texto) {

       var tela = document.querySelector('canvas');
```

```
        var pincel = tela.getContext('2d');
        pincel.font='15px Georgia';
        pincel.fillStyle='black';
        pincel.fillText(texto, x, y);
    }
    function desenhaBarra(x, y, serie, cores, texto) {
        desenhaTexto(x, y - 10, texto);
        var somaAltura = 0;
        for (var i = 0; i < serie.length; i++) {
            var altura = serie[i];
            desenhaRetangulo(x, y + somaAltura, 50, altura, cores[i]);
            somaAltura = somaAltura + altura;
        }
    }
    var cores = ['blue','green','yellow', 'red'];
    var serie2015 = [50,25,20,5];
    var serie2016 = [65,20,13,2];
    desenhaBarra(50, 50, serie2015, cores, '2015');
    desenhaBarra(150, 50, serie2016, cores, '2016');
```

</script>

017.Nossa tela está viva, ela responde!

Nesta aula, iniciaremos um novo programa, ao qual demos o título `programa3.html`. É um `<canvas>`, de dimensão `600x400`, no qual utilizamos o `pincel` e preenchemos em cinza:

```
<canvas width="600"  height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

</script>
```

Até então, nós havíamos trabalhado com formas geométricas, como esferas e quadrados, pensando também na organização de nosso código. Nosso objetivo a partir de agora será fazer com que o nosso `<canvas>` interaja com o usuário.

Se você pensa em seguir e construir jogos, temos que ter uma maneira de interagir com o usuário. Ele deve ser capaz de clicar no `<canvas>` e isso dar início a alguma funcionalidade.

De início, faremos com que, ao clicar no `<canvas>`, seja exibido um alerta.

Para começar, criaremos uma função chamada `exibeAlerta()`, que exibirá a mensagem `Cliquei`:

```
<canvas width="600"  height="400"></canvas>

<script>
```

```
    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

    function exibeAlerta() {

      alert('Cliquei');

    }

</script>
```

Salvaremos e recarregaremos a página, porém, nada acontecerá se clicarmos nela. Para que o alerta seja exibido, temos que chamar a função:

```
<canvas width="600"  height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

    function exibeAlerta() {

      alert('Cliquei');

    }

    exibeAlerta();
```

```
</script>
```

Se recarregarmos a página agora, o `exibeAlerta` é exibido automaticamente. Não é este o objetivo, queremos que o alerta seja exibido somente quando a página for clicada.

Como vimos, o JavaScript nos permite trabalhar com **eventos**. No caso, utilizaremos a propriedade `onclick`, tudo que atribuirmos a ela - se for uma função - esta será chamada pelo clique. Assim, conectaremos o `exibeAlerta` ao `onClick`:

```
<canvas width="600"  height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

      pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

    function exibeAlerta() {

        alert('Cliquei');

    }

    tela.onclick = exibeAlerta;

</script>
```

Lembrando que não podemos utilizar os parênteses neste último caso pois, assim executaríamos a função automaticamente, em vez de guardá-la para ser executada ao clique.

Salvaremos o programa e recarregaremos a página. Ao clicarmos na tela, é exibido um alerta com a mensagem "Cliquei". Funcionou!

Neste caso, quem chama a função `exibeAlerta`? O próprio navegador, ao clicarmos sobre o `<canvas>`. Além de exibirmos um alerta, é importante que nosso programa seja capaz, também, de identificar em qual posição a tela foi clicada.

Como podemos fazer isso?

Ao clicarmos na tela, como dito anteriormente, o próprio navegador é quem chama a função `exibeAlerta`. Toda vez que faz isso, ele passa um parâmetro especial para a função - até então, não havíamos feito isso em nosso código.

Nós criaremos este parâmetro, que chamaremos de `evento`, e utilizaremos o `console.log(evento)` para que ele seja exibido no navegador:

```
<canvas width="600"  height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

    function exibeAlerta(evento) {

        alert('Cliquei');

        console.log(evento);

    }

    tela.onclick = exibeAlerta;

</script>
```

Isso fará com que tenhamos acesso a este parâmetro passado pelo navegador, e assim poderemos descobrir a posição exata do clique.

Após recarregarmos o programa, clicaremos na tela, onde será exibido o alerta. Em seguida, abriremos o console - utilizando o atalho "F12" -, e teremos o `MouseEvent`, com todos os detalhes do evento.

O nome do parâmetro que nós passamos poderia ser qualquer um, o importante é **haver um parâmetro**.

Se expandirmos o `MouseEvent`, clicando na seta antes de seu nome, há diversos detalhes sobre ele, inclusive a posição do mouse dentro do `<canvas>`.

Antes de aprendermos a obter estas informações, vamos recapitular:

- Criamos uma função chama `exibeAlerta`;
- Ela recebe como parâmetro um `evento`;
- Em seu bloco, ela exibe o alerta '`Cliquei`' apenas, e faz um `console.log(evento)`;

Se chamarmos o `exibeAlerta`, temos que passar um parâmetro, mas não temos como saber de antemão qual ponto da tela será clicado. Assim, quem chama essa função é, exclusivamente, o navegador, ele quem tem o parâmetro que trará para nós as coordenadas da posição do cursor no momento do clique.

Adiante, veremos como obter tais coordenadas.

018.Mouse, diga-me em que posição estás

O que fizemos aqui foi definir uma função que será chamada quando um determinado evento ocorrer. Já vimos isso anteriormente, quando pegamos o clique do mouse em um botão do nosso HTML, para dentro do JavaScript! Esse tipo de função é o que chamamos de *callback*. No nosso caso, definimos que, quando alguém clicar na tela (`tela.onclick`), vamos chamar uma função que por sua vez chama o `alert`.

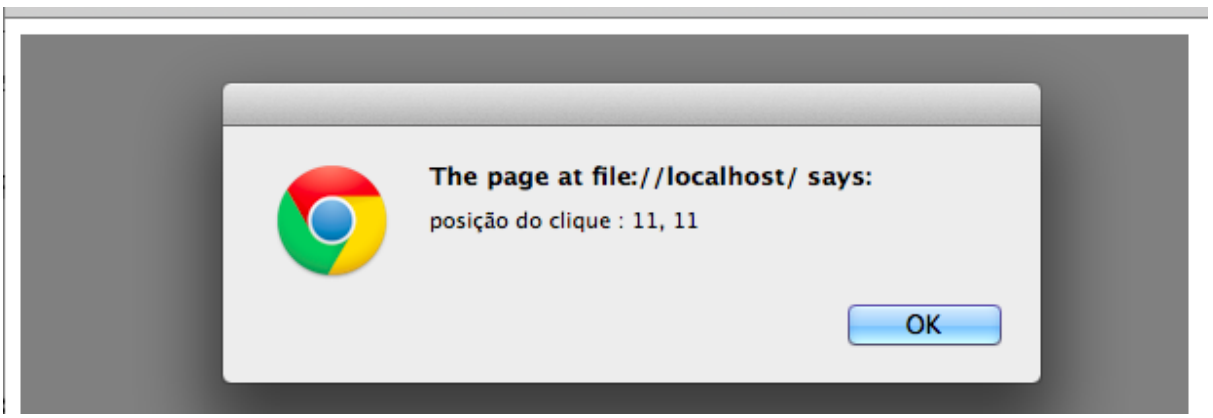
Podemos descobrir a coordenada em que o usuário clicou. Muitas vezes, quando uma função de *callback* é chamada, são passados argumentos descrevendo o

evento que acabou de acontecer. Neste caso é passado um evento chamado `MouseEvent`, com o qual podemos descobrir a posição *x,y* do clique através de variáveis de dentro desse evento. Altere seu código da seguinte forma:

```
// código anterior omitido



function exibeAlerta(evento) {

    var x = evento.pageX;

    var y = evento.pageY;

    alert("posição do clique : " + x + ", " + y);

}

// código posterior omitido
```

Repare que agora estamos recebendo como parâmetro uma variável a que demos o nome de `evento`. É comum dar o nome a ela de `mouseEvent` ou até mesmo um simples `e`. Abra o seu HTML e clique em algum lugar da tela, qual é o resultado?

Isso mesmo! Ele te dá a posição do seu clique. Mesmo assim, perceba que há algo estranho. Tente, por exemplo, clicar no canto superior esquerdo da sua imagem. Observe o resultado abaixo:



No nosso caso, mesmo clicando bem no canto superior esquerdo do nosso canvas cinza, obtivemos *11, 11*. Algumas vezes, com mais precisão, obtivemos *10,10*. Por que o resultado não foi *0,0*? Isso ocorre pois, como as próprias variáveis *evento.pageX. evento.pageY* nos dizem, essa é a posição do clique em relação à página! Se quisermos as coordenadas relativas ao canvas, basta subtrairmos a posição em que o canvas (nossa *tela*) foi desenhado na página:

```
// código anterior omitido

function exibeAlerta(evento) {

    var x = evento.pageX - tela.offsetLeft;

    var y = evento.pageY - tela.offsetTop;

    alert("posição do clique : " + x + ", " + y);

}

// código posterior omitido
```

Ler apenas essa informação não é tão interessante. Que tal desenhar um círculo azul em cada ponto que o usuário clicar? Basta, dentro dessa função, fazer uso aquela função `arc`, que já conhecemos:

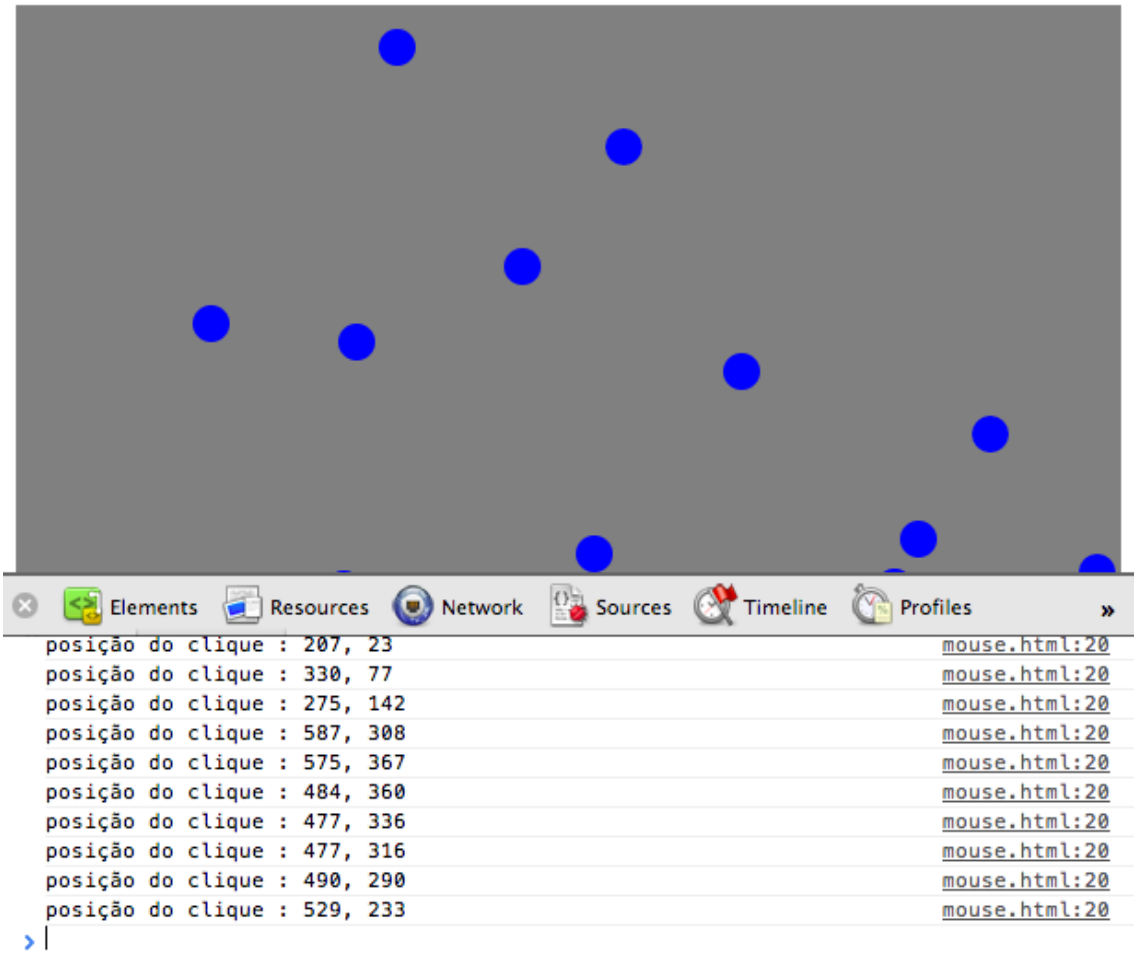
```
// código anterior omitido

function exibeAlerta(evento) {

    var x = evento.pageX - tela.offsetLeft;

    var y = evento.pageY - tela.offsetTop;

    pincel.fillStyle="blue";

    pincel.beginPath();
```

```
    pincel.arc(x, y, 10, 0, 2*3.14);

    pincel.fill();

    console.log("posição do clique : " + x + ", " + y);

}

// código posterior omitido
```

Recarregue seu arquivo no navegador. Clique em alguns pontos na tela e veja o resultado que obtemos!

Veja que, além de desenhar o círculo onde você clica, estamos colocando a informação das coordenadas no console do navegador. Vimos esse recurso no começo. É muito mais prático utilizá-lo do que imprimir valores com `document.write` ou o inoportuno `alert`. Para ver as posições onde estamos clicando, basta abrir o *Console JavaScript* do Chrome. Relembrando: clique no ícone de menus/ferramentas, depois acesse o menu *Ferramentas* (*Tools*) e por último *Console JavaScript*.

Após alguns cliques, você obterá um resultado como o seguinte:



Já podemos fazer um software parecido com o Paint, não? Ou como o Photoshop, para os mais ousados!

Veja que o nome da função não reflete mais sua funcionalidade. Ela não exibe mais um alerta, pelo contrário, ela agora desenha um círculo. Sendo assim, para que nosso código fique mais fácil de ler, vamos trocar o nome da função para `desenhaCirculo`. Só não podemos esquecer de mudar também na linha que atribui essa função ao evento `tela.onclick`:

```
// código anterior omitido

function desenhaCirculo(evento) {

  var x = evento.pageX - tela.offsetLeft;

  var y = evento.pageY - tela.offsetTop;
```

```
    pincel.fillStyle="blue";
    pincel.beginPath();
    pincel.arc(x, y, 10, 0, 2*3.14);
    pincel.fill();


    console.log("posição do clique : " + x + ", " + y);
}
// não esqueça de mudar aqui
tela.onclick = desenhaCirculo;
```

019.Uma forma diferente de testar um if

Temos o seguinte código:

```
<script>
    var aprendi = true;
    if(aprendi == true) {
        alert('O instrutor fica muito contente');
    } else {
        alert('O instrutor não vai desistir até você virar um cangaceiro!');
    }
</script>
```

COPIAR CÓDIGO

Não há nenhuma novidade, aliás lidamos com códigos como esse no curso anterior. Nossa condição `if` testa se o valor da variável `aprendi` é verdadeira, ou seja, se é `true`. Usamos inclusive o operador `==`.

No entanto, podemos simplificar esse código para:

```
<script>

    var aprendi = true;

    if(aprendi) {

        alert('O instrutor fica muito contente');

    } else {

        alert('O instrutor não vai desistir até você virar um cangaceiro!');

    }

</script>
```

Veja que não usei mais o operador `==`. Você deve estar intrigado por ter funcionado. Primeiro, lembre-se que o `if()` espera receber `true` ou `false` para saber se executa o código do seu bloco ou o código do bloco do `else`. Se `aprendi` já é `true`, é redundante realizarmos o teste `aprendi == true`.

Se uma variável já guarda `true` ou `false` podemos usá-la diretamente no if poupando alguns caracteres. No entanto, se você se sente mais seguro com a forma anterior, continue com ela. O importante é você ir sentindo o que prefere e o que não prefere em programação. Até porque, não desejo que meus alunos sejam "carbonos" de mim.

Programar é uma arte e como toda arte começamos a enxergar beleza onde não havia.

020.Trocando de cor

Aprendemos nesta aula a interagir com o usuário, por exemplo, desenhando uma bolinha azul toda vez que ele clicar no **canvas**, nossa tela:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(evento) {

        var x = evento.pageX - tela.offsetLeft;

        var y = evento.pageY - tela.offsetTop;

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, 10, 0, 2 * 3.14);

        pincel.fill();

        console.log(x + ',' + y);

    }

    tela.onclick = desenhaCirculo;


</script>
```

Para isso, foi necessário associar a função `desenhaCirculo` ao evento `onclick` da nossa tela. Aprendemos também que será o navegador que chamará a nossa função, pois ele sabe identificar cliques no `canvas`. Além de chamar a função, ele passará sempre um parâmetro para ela. Graças a esse parâmetro, temos acesso a várias informações sobre o evento disparado e, no caso, podemos descobrir a posição do eixo `x` e `y` da tela que o usuário clicou.

Agora que já recapitulamos o que fizemos neste capítulo, tenho uma proposta de mudança do código acima bem interessante e quero que você quebre a cabeça para implementá-la.

Vamos permitir que o usuário altere a cor da bolinha que é desenhada na tela. As cores serão obrigatoriamente **`blue`**, **`red`** e **`green`**. Perceba que temos uma **lista** de cores e isso deve remetê-lo à aula de `Array` do primeiro módulo do curso.

Como essa escolha será feita? A cada clique do botão **DIREITO** do mouse, a cor padrão, que é `blue`, deverá se tornar `red`. Se o usuário clicar mais uma vez com o botão **DIREITO**, a cor escolhida deverá ser `green`, **nessa ordem**. Por fim, se clicarmos novamente, voltamos para a cor `blue` e repetimos a ordem de seleção de cores.

Obs: com o botão **ESQUERDO** faremos os cliques para as bolinhas aparecerem.

É claro que lhe darei uma dica, pois não ensinei a você a associar a execução de uma função ao clique do botão direito do mouse. O evento responsável por isso é o **`oncontextmenu`**. Para você começar bem, crie um programa com o código anterior e faça um teste desse evento conforme meu exemplo:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');
```

```
    pincel.fillStyle = 'grey';
    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(evento) {
        var x = evento.pageX - tela.offsetLeft;
        var y = evento.pageY - tela.offsetTop;
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, 10, 0, 2 * 3.14);
        pincel.fill();
        console.log(x + ',' + y);
    }
    tela.onclick = desenhaCirculo;
    function mudaCor() {
        alert('Funcionou!');
        return false;
    }
    tela.oncontextmenu = mudaCor;
</script>
```

Recarregue o programa e experimente clicar com o botão **direito** na tela. A mensagem **"Funcionou!"** será exibida. Não estranhe a última linha com a instrução **return false**. Ela é importante para que o menu contextual padrão do **canvas** não seja exibido, ou seja, queremos apenas trocar de cor com o clique do botão e não exibir um menu para o usuário.

Combine os diversos aprendizados que você conquistou até agora nesse exercício e marque a alternativa que mais se aproxima com a proposta de implementação.

```
<canvas width="600" height="400"></canvas>

<script>

  var tela = document.querySelector('canvas');

  var pincel = tela.getContext('2d');

  pincel.fillStyle = 'grey';

  pincel.fillRect(0, 0, 600, 400);

  var cores = ['blue', 'red', 'green']

  var indiceCorAtual = 0; // começa com blue

  function desenhaCirculo(evento) {

    var x = evento.pageX - tela.offsetLeft;

    var y = evento.pageY - tela.offsetTop;

    pincel.fillStyle = cores[indiceCorAtual];

    pincel.beginPath();

    pincel.arc(x, y, 10, 0, 2 * 3.14);
```

```
  pincel.fill();
  console.log(x + ',' + y);
 }
 tela.onclick = desenhaCirculo;

 function mudaCor() {
  indiceCorAtual++;
  if (indiceCorAtual >= cores.length) {
   indiceCorAtual = 0; // volta para a primeira cor, azul
  }
  return false; // para não exibir o menu padrão do canvas
 }
 tela.oncontextmenu = mudaCor;
</script>
```

021.Era uma vez uma bolinha que virou um bolão!

Que tal ainda continuarmos com nossas inofensivas e não menos importantes bolinhas? Mais uma vez, segue o código que desenha uma bolinha quando clicamos na tela:

```
<canvas width="600" height="400"></canvas>

<script>
```

```
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'grey';
    pincel.fillRect(0, 0, 600, 400);
    function desenhaCirculo(evento) {
        var x = evento.pageX - tela.offsetLeft;
        var y = evento.pageY - tela.offsetTop;
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, 10, 0, 2 * 3.14);
        pincel.fill();
        console.log(x + ',' + y);
    }
    tela.onclick = desenhaCirculo;
</script>
```

Dessa vez quero propor algo diferente. Quando clicarmos na tela pressionando a tecla **SHIFT**, vamos desenhar uma bolinha acrescida de 20 ao seu raio. Se soltarmos a tecla e clicarmos, ela voltará com seu raio que hoje é 10.

Você deve estar se perguntando como saber se o usuário está pressionando a tecla SHIFT, certo? Você lembra que uma função passada para o evento `onclick` tem acesso às coordenadas do ponteiro do mouse? Pois é, além de sabermos as coordenadas, podemos perguntar também se a tecla `SHIFT` está pressionada usando

`evento.shiftKey`. Dessa forma, se for `true`, é porque ela está sendo pressionada, se for `false`, é porque não foi.

Você deve testar a condição dentro da função `desenhaCirculo` do código anterior.

A partir desse ponto é com você. Assinale a alternativa que corresponde ao código com a função `desenhaCirculo` implementada.

Lembre-se: ela deve aumentar o raio da bolinha ao clicar na tecla `SHIFT` do seu teclado.

022.Era uma vez um bolão que quase explodiu!

Em um dos exercícios anteriores, aprendemos a desenhar uma bolinha de raio maior toda vez que um clique era feito na tela com a tecla **SHIFT** pressionada.

Vamos recapitular esse código:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'gray';

    pincel.fillRect(0, 0, 600, 400);


    function desenhaCirculo(evento) {

        var x = evento.pageX - tela.offsetLeft;

        var y = evento.pageY - tela.offsetTop;
```

```
        var raio = 10;
        console.log(x + ',' + y);
        if (evento.shiftKey) {
           raio = raio + 20; // raio agora passa a valer 30!
        }
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * 3.14);
        pincel.fill();
    }
    tela.onclick = desenhaCirculo;


</script>
```

Teste o código para lembrar o resultado. Quero propor uma mudança nesse código que é a seguinte: toda vez que o usuário clicar na tela com o `SHIFT` pressionado, vamos somar 10 ao valor do `raio` atual. Sendo assim, mesmo soltando o `SHIFT` as próximas bolinhas utilizarão o mesmo valor de raio. Se clicarmos diversas vezes segurando `SHIFT` teremos uma bola cada vez maior.

o novo código é:

```
<canvas width="600" height="400"></canvas>

<script>
  var tela = document.querySelector('canvas');
  var pincel = tela.getContext('2d');
  pincel.fillStyle = 'grey';
  pincel.fillRect(0, 0, 600, 400);
  var raio = 10;

  function desenhaCirculo(evento) {
    var x = evento.pageX - tela.offsetLeft;
    var y = evento.pageY - tela.offsetTop;
    console.log(x + ',' + y);
    if (evento.shiftKey) {
      raio = raio + 10;
    }
    pincel.fillStyle = 'blue';
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * 3.14);
    pincel.fill();
```

```
  }

  tela.onclick = desenhaCirculo;

</script>
```

A primeira coisa que precisamos fazer é extrair a inicialização da variável raio para fora da função desenhaCirculo. Se ela continuasse dentro da função, a cada clique ela voltaria para o valor inicial 10.

E a última, no lugar de somarmos 20 com o raio, realizamos a soma de 10 a cada clique com o SHIFTpressionado!

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

    var raio = 10;

    function desenhaCirculo(evento) {

        var x = evento.pageX - tela.offsetLeft;

        var y = evento.pageY - tela.offsetTop;

        console.log(x + ',' + y);

        if (evento.shiftKey && evento.altKey) {

            alert('Só aperte uma tecla por vez, por favor!');

        } else if(evento.shiftKey && raio + 10 <= 40) {
```

```
            raio = raio + 10;
        } else if(evento.altKey && raio - 5 >= 10) {
            raio = raio - 5;
        }
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * 3.14);
        pincel.fill();
    }
    tela.onclick = desenhaCirculo;
</script>
```

Parabéns! Vamos analisar uma solução ainda um pouco bruta. Nela, usamos o clássico if para saber se as teclas estão pressionadas, além disso usamos o operador && para verificar se SHIFT e ALT estão sendo pressionadas ao mesmo tempo. Um ponto a destacar é que antes de incrementarmos ou decrementarmos o raio, fazemos uma simulação aumentando ou diminuindo o raio sem guardar o resultado na variável. Esse preview do valor nos ajuda a saber se podemos continuar aumentando ou diminuindo o raio.

023.Desenhando com o mouse

Dessa vez, só temos o código que exibe o `canvas` e desenha um círculo, mas a função `desenhaCirculo` não está associada a nenhum evento do JavaScript:

```
<canvas width="600" height="400"></canvas>

<script>
    var tela = document.querySelector('canvas');
```

```
    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(evento) {

        var x = evento.pageX - tela.offsetLeft;

        var y = evento.pageY - tela.offsetTop;

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, 10, 0, 2 * 3.14);

        pincel.fill();

        console.log(x + ',' + y);

    }

</script>
```

O desafio agora é o seguinte. Se andarmos com o ponteiro do mouse sobre a tela enquanto o botão esquerdo é pressionado, vamos desenhar um círculo. Isso significa que enquanto não soltarmos o botão esquerdo, desenharemos um círculo ao lado do outro, que na verdade dará um efeito que estamos passando um pincel na tela. Se soltarmos o botão esquerdo, o ato de mover o mouse sob a tela não deverá desenhar nada. No final, queremos um efeito parecido com ferramentas como Paint Brush ou Photoshop, que permite o usuário desenhar na tela.

O evento do JavaScript que permite capturar o movimento do mouse, e logo sua coordenada, é o **onmousemove**, contudo esse evento não é capaz de saber se o botão do mouse está clicando ou não. E agora?

Existem os eventos, `onmousedown` e `onmouseup`. O primeiro é disparado toda vez que o botão esquerdo do mouse é pressionado e o segundo quando ele é solto. Sendo assim, de alguma maneira, nossa função `desenhaCirculo` só pode desenhar na tela se o botão estiver pressionado, se não estiver, nada acontecerá.

E agora, como resolver? **DICA:** as funções passadas para `onmousedown` e `onmouseup` podem alterar o valor de uma variável, que será usada por `desenhaCirculo` para saber se ele desenhará ou não um círculo. Complicou? Nada que um momento de reflexão não resolva.

Veja que a problemática está toda em saber se desenhamos ou não na tela enquanto passamos o mouse sob a mesma. E sabemos que a condição está atrelada ao botão esquerdo do mouse estar pressionado.

Sendo assim, vamos declarar uma variável booleana chamada `desenha`, que começa como `false`. É essa variável que será utilizada pela função `desenhaCirculo` para saber se ela deve ou não desenhar:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';

    pincel.fillRect(0, 0, 600, 400);

    var desenha = false;

    function desenhaCirculo(evento) {



        if(desenha) {

            var x = evento.pageX - tela.offsetLeft;
```

```
            var y = evento.pageY - tela.offsetTop;

            pincel.fillStyle = 'blue';

            pincel.beginPath();

            pincel.arc(x, y, 10, 0, 2 * 3.14);

            pincel.fill();

        }

        console.log(x + ',' + y);

    }

    tela.onmousemove = desenhaCirculo;

</script>
```

Se fizermos um teste, vamos passar o mouse pela tela e nada será desenhado. Faz todo sentido, porque a variável `desenha` é falsa e nunca a condição `if` de `desenhaCirculo`, que depende que essa variável seja verdadeira, será executada.

Na explicação do exercício, falei sobre `onmousedown` e sobre o `onmouseup` e que o primeiro serve para executar um código quando o mouse é pressionado e o segundo quando o botão do mouse é solto. Que tal então criar duas funções, a primeira `habilitaDesenho` e a segunda `desabilitaDesenho` e associar uma com cada um desses eventos. Qual será o código de cada uma delas? A primeira atribui `true` para a variável `desenha` e a segunda atribui `false`. Como a função `desenhaCirculo` depende da variável que foi alterada, conseguimos o resultado esperado:

```
<canvas width="600" height="400"></canvas>



<script>

    var tela = document.querySelector('canvas');
```

```
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'grey';
    pincel.fillRect(0, 0, 600, 400);
    var desenha = false;
    function desenhaCirculo(evento) {
        if(desenha) {
            var x = evento.pageX - tela.offsetLeft;
            var y = evento.pageY - tela.offsetTop;
            pincel.fillStyle = 'blue';
            pincel.beginPath();
            pincel.arc(x, y, 10, 0, 2 * 3.14);
            pincel.fill();
        }
        console.log(x + ',' + y);
    }
    tela.onmousemove = desenhaCirculo;
    function habilitaDesenhar() {
        desenha = true;
    }
    function desabilitaDesenhar() {
```

```
        desenha = false;

    }

    tela.onmousedown = habilitaDesenhar;

    tela.onmouseup = desabilitaDesenhar;

</script>
```

Faça um teste! Quando você clicar com o botão esquerdo do mouse sem soltá-lo, a variável `desenha` será `true`. Enquanto você não soltar, ela continuará com esse valor, sendo assim, ao mover o mouse pela tela com o botão pressionado, a função `desenhaCirculo` desenhará o circulo a cada movimento que fizermos. Contudo, se soltarmos o botão, a variável `desenha` receberá `false`, pois a função `desabilitaDesenhar` mudará a variável `desenha` para `false`.

BÔNUS: FUNÇÕES ANÔNIMAS

Se você chegou até aqui é porque quer ficar ainda melhor em lógica de programação e talvez esteja querendo se preparar para aprender com mais detalhes a linguagem JavaScript. E por isso vou antecipar um conhecimento que você só veria em um curso específico de JavaScript. Mesmo este curso sendo de lógica e focar no essencial da linguagem para conseguirmos criar coisas, esse assunto é interessante.

Veja que declaramos as funções `habilitaDesenhar` e `desabilitaDesenhar` para lidar com o "liga e desliga" da habilidade de desenharmos com o mouse. Certo? Contudo, o que acontece se eu chegar nessas duas funções e remover os seus nomes? Nosso código, **que não funcionará**, ficará assim:

```
<canvas width="600" height="400"></canvas>



<script>

    var tela = document.querySelector('canvas');
```

```
var pincel = tela.getContext('2d');
pincel.fillStyle = 'grey';
pincel.fillRect(0, 0, 600, 400);
var desenha = false;
function desenhaCirculo(evento) {
    if(desenha) {
        var x = evento.pageX - tela.offsetLeft;
        var y = evento.pageY - tela.offsetTop;
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, 10, 0, 2 * 3.14);
        pincel.fill();
    }
    console.log(x + ',' + y);
}
tela.onmousemove = desenhaCirculo;
// função não tem mais nome, se não tem nome é anônima
function() {
    desenha = true;
}
```

```
    // função não tem mais nome, se não tem nome é anônima

    function() {

        desenha = false;

    }

    tela.onmousedown = habilitaDesenhar; // habilitaDesenhar é um nome que não existe mais

    tela.onmouseup = desabilitaDesenhar; // desabilitaDesenhar é um nome que não existe mais

</script>
```

Como as duas funções não possuem mais um nome, são chamadas de **funções anônimas**. Isso mesmo, uma função anônima é aquela que não possui um nome e nosso código dá um erro porque se elas não possuem nome, como iremos chamá-las ou atribuí-las aos eventos `onmousedown` e `onmouseup`, certo?

Então, como iremos passar um código para `tela.onmousedown` e `tela.onmouseup`, se não temos mais o nome da função? É aqui que eu quero pedir a atenção de vocês. Uma função anônima não pode existir solta, porque como não tem um nome, ninguém será capaz de chamá-la, mas podemos atribuir a função anônima diretamente a uma variável ou a uma propriedade, como `tela.onmousedown` e `tela.onmouseup`. Veja:

```
<canvas width="600" height="400"></canvas>



<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'grey';
```

```
    pincel.fillRect(0, 0, 600, 400);
    var desenha = false;
    function desenhaCirculo(evento) {
        if(desenha) {
            var x = evento.pageX - tela.offsetLeft;
            var y = evento.pageY - tela.offsetTop;
            pincel.fillStyle = 'blue';
            pincel.beginPath();
            pincel.arc(x, y, 10, 0, 2 * 3.14);
            pincel.fill();
        }
        console.log(x + ',' + y);
    }
    tela.onmousemove = desenhaCirculo;
    // atribuindo diretamente a função anônima
    tela.onmousedown = function() {
        desenha = true;
    }
    // atribuindo diretamente a função anônima
     tela.onmouseup = function() {
```

```
        desenha = false;
    }
</script>
```

Podemos fazer a mesma coisa com a função `desenhaCirculo`:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'grey';
    pincel.fillRect(0, 0, 600, 400);
    var desenha = false;
  // atribuindo diretamente a função anônima
   tela.onmousemove = function(evento) {
        if(desenha) {
            var x = evento.pageX - tela.offsetLeft;
            var y = evento.pageY - tela.offsetTop;
            pincel.fillStyle = 'blue';
            pincel.beginPath();
            pincel.arc(x, y, 10, 0, 2 * 3.14);
```

```
            pincel.fill();
        }
        console.log(x + ',' + y);
    }
    tela.onmousedown = function() {
        desenha = true;
    }
    tela.onmouseup = function() {
        desenha = false;
    }
</script>
```

A ideia aqui é já ir plantando na cabeça de vocês que isso é possível, para quando vocês saírem do mundo da lógica e forem aprender com mais detalhes a linguagem JavaScript entender o que isso significa.

024.Acertando o alvo

Teste o seguinte código abaixo, que desenha um alvo:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
```

```
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    desenhaCirculo(300,200, raio + 20, 'red'); // maior círculo
    desenhaCirculo(300,200, raio + 10, 'white');
    desenhaCirculo(300,200, raio, 'red'); // menor circulo
    function dispara(evento) {
        var x = evento.pageX - tela.offsetLeft;
        var y = evento.pageY - tela.offsetTop;
       // lógica de acerto?
    }

    tela.onclick = dispara;

</script>
```

Repare que o alvo é feito a partir de três círculos, todos com as mesmas coordenadas x e y, mas que possuem raios diferentes. Começamos desenhando primeiro do maior para o menor, se tivéssemos feito ao contrário o círculo maior vermelho seria desenhado em cima dos demais.

Muito bem. O desafio desse exercício é exibir um alerta toda vez que um clique do mouse for realizado no centro do alvo. Para isso, quero que você extraia o código a partir do seguinte algoritmo, que nada mais é do que a lógica de acerto.

O algoritmo, a lógica de acerto.

A lógica de acerto é a seguinte. Sabemos que as coordenadas x e y do alvo são 300 e 200, respectivamente, certo? Então, precisamos comparar essas coordenadas com as coordenadas do clique do mouse. O problema é que não podemos testar se o x e y do clique é igual ao x e y do alvo, porque estaríamos obrigando o jogador a clicar exatamente no centro do alvo. Queremos considerar como acerto qualquer coordenada x e y dentro do círculo menor, aquele com raio 10.

Então, se o x do alvo é 300, qualquer clique entre 290 e 310 é válido. Mas de onde eu tirei essa faixa de acerto? Levando em consideração o raio do círculo. Levo em consideração 10 pontos à esquerda do centro do menor círculo e 10 pontos à direita. Então, basta fazermos a mesma coisa com o y!

Quebre a cabeça um pouco e assinale a alternativa que mais se aproxima do código com o comportamento esperado:

Nosso programa, levando em consideração a regra da lógica de acerto fica assim:

```
<canvas width="600" height="400"></canvas>



<script>

  var tela = document.querySelector('canvas');

  var pincel = tela.getContext('2d');
```

```
pincel.fillStyle = 'lightgray';
pincel.fillRect(0, 0, 600, 400);
var raio = 10;

function desenhaCirculo(x, y, raio, cor) {
  pincel.fillStyle = cor;
  pincel.beginPath();
  pincel.arc(x, y, raio, 0, 2 * Math.PI);
  pincel.fill();
}
desenhaCirculo(300, 200, raio + 20, 'red');
desenhaCirculo(300, 200, raio + 10, 'white');
desenhaCirculo(300, 200, raio, 'red');

function dispara(evento) {
  var x = evento.pageX - tela.offsetLeft;
  var y = evento.pageY - tela.offsetTop;
  if ((x > 300 - raio) &&
    (x < 300 + raio) &&
    (y > 200 - raio) &&
```

```
    (y < 200 + raio)) {

    alert('Acertou no centro do alvo');

   }

 }

 tela.onclick = dispara;

</script>
```

Parabéns, este é o raciocínio correto. Você também tem a possibilidade de deixar sem ou com parênteses em cada condição do if, para deixar o código mais legível, optamos por manter a estrutura condicional entre parênteses.

```
<canvas width="600" height="400"></canvas>

<script>

   var tela = document.querySelector('canvas');

   var pincel = tela.getContext('2d');

   pincel.fillStyle = 'lightgray';

   pincel.fillRect(0, 0, 600, 400);

   var raio = 10;

   function desenhaCirculo(x, y, raio, cor) {

      pincel.fillStyle = cor;

      pincel.beginPath();

      pincel.arc(x, y, raio, 0, 2 * Math.PI);

      pincel.fill();

   }

   desenhaCirculo(300,200, raio + 20, 'red');
```

```
    desenhaCirculo(300,200, raio + 10, 'white');

    desenhaCirculo(300,200, raio, 'red');

    function dispara(evento) {

        var x = evento.pageX - tela.offsetLeft;

        var y = evento.pageY - tela.offsetTop;

        if (x > 300 - raio

            && x < 300 + raio

                && y > 200 - raio

                    && y < 200 + raio) {

            alert('Acertou');

        }

    }

    tela.onclick = dispara;

</script>
```

Movendo elementos: animações simples

025.Desenhamos um círculo, e daí?

Nesta aula, daremos início ao `programa4.html`. Nele, temos um código na linha dos que já utilizamos em aulas anteriores:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');
```

```
    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

</script>
```

Aplicaremos nossos conhecimentos de lógica de programação para animarmos uma esfera na tela, e termos ferramentas para conseguirmos criar um jogo simples.

A primeira coisa a fazer é criar uma esfera na tela.

Inverteremos um pouco a ordem, e chamaremos primeiro a função, para analisarmos o que ela precisará receber como parâmetro:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    desenhaCirculo();

</script>
```

Passaremos as coordenadas X e Y, e o tamanho do raio, que será `10`:

```
<canvas width="600" height="400"></canvas>
```

```
<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    desenhaCirculo(20, 20, 10);

</script>
```

Como ainda não definimos a função, ela não poderá ser executada. Começaremos a construção dela passando os parâmetros, que serão `x, y, raio` - de acordo com o tipo de valor que cada um receberá -, em seu bloco, incluiremos `fillStyle` na cor azul, o `beginPath`, e o `arc`, que receberá o ponto onde queremos plotar a esfera no `<canvas>`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {
```

```
        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * 3.14);

    }

    desenhaCirculo(20, 20, 10);

</script>
```

Como sabemos, na matemática, o valor de Pi é equivalente a `3.14`. Como sabemos, no JS temos algumas funções que iniciam com `Math.` e nos dão funcionalidades básicas, justamente, temos o `Math.PI`, que é o que utilizaremos em nosso código:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();
```

```
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
    }
    desenhaCirculo(20, 20, 10);
</script>
```

Para preenchermos o círculo, utilizaremos o `pincel.fill`:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    function desenhaCirculo(x, y, raio) {
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
```

```
    desenhaCirculo(20, 20, 10);

</script>
```

Salvaremos e recarregaremos o programa, temos um círculo azul no canto superior esquerdo da tela. Para desenharmos outra esfera, basta repetirmos o chamado à função, com novos parâmetros, que sejam diferentes dos primeiros.

Adiante, aprenderemos a animar esta esfera.

026.Desenhando

Nesta aula, daremos continuidade ao processo de animação.

Para chegarmos a este resultado, faremos com que a esfera saía de um ponto nas coordenadas iniciais, e chegue em um novo, que definiremos com novas coordenadas. Para movimentarmos nossa esfera horizontalmente, manipularemos o valor da coordenada X.

Como vimos, há o método `for` para fazermos este tipo de alteração reiterada de uma única propriedade. Criaremos nele uma variável `x`, que receberá `20`, uma condição `x < 600`, e incremento de `x = x + 1`, que, como sabemos, pode ser feito com o `x++`:

```
<canvas width="600" height="400"></canvas>



<script>



    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';
```

```
    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }



        for(var x = 20; x < 600; x++) {



        }

    desenhaCirculo(20, 20, 10);

</script>
```

Colocaremos a função `desenhaCirculo()` dentro do `for()`, substituiremos o parâmetro `20` pelo `x` e manteremos os demais, pois queremos que a esfera se mova somente no eixo X:

```
<canvas width="600" height="400"></canvas>

<script>
```

```
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    function desenhaCirculo(x, y, raio) {
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
        for(var x = 20; x < 600; x++) {
            desenhaCirculo(x, 20, 10);
        }
</script>
```

Salvaremos e recarregaremos o programa. Em vez da animação temos a imagem de uma linha em azul. Isso aconteceu porque demos o espaçamento de apenas `1` pixel entre as esferas, assim, quase todas ficaram umas sobre as outras, dando a ilusão de formarem uma única linha.

Para resolvermos isso, utilizaremos o `pincel.clearRect()`, o que indicará ao JS que ele deve limpar o retângulo:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

            for(var x = 20; x < 600; x++) {

            pincel.clearRect(0, 0, 600, 400);

            desenhaCirculo(x, 20, 10);

        }

</script>
```

Após salvarmos e recarregarmos, veremos que o fundo de `<canvas>` ficou completamente branco, e visualizamos apenas uma esfera, no canto superior direito.

Por que ela ainda não está animada? Isso ocorre porque a nossa CPU faz este processo muito rapidamente, antes que possamos visualizar o processo, conseguimos ver apenas o resultado final.

A ideia é podermos esperar alguns segundos entre o surgimento das esferas.

Além disso, para esclarecer a função do `clearRect`, criaremos a função `limpaTela`, que receberá nossa função. Assim, chamaremos o `limpaTela()` já dentro do nosso `for()`:

```
<canvas width="600" height="400"></canvas>



<script>



    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);



    function desenhaCirculo(x, y, raio) {
```

```
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
        function limpaTela() {
            pincel.clearRect(0, 0, 600, 400);
        }
        for(var x = 20; x < 600; x++) {
            limpaTela();
            desenhaCirculo(x, 20, 10);
        }
</script>
```

Assim as funcionalidades ficam mais evidentes para quem lê nosso código. Vimos que o `for` é uma solução ruim, porque fará com que o processo de animação ocorra rápido demais para que sejamos capazes de perceber. Adiante, veremos como podemos resolver este problema.

027.Animações simples

Dando continuidade ao nosso projeto, aprenderemos a animar nossa esfera para que ela viaje de um ponto da tela até outro.

Removeremos o for:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

        function limpaTela() {

            pincel.clearRect(0, 0, 600, 400);

        }

</script>
```

Primeiro, trabalharemos com o conceito de como podemos chamar um código com intervalos de tempo.

Em seguida, criaremos a função `exibeMensagemNoConsole()` que terá um `console.log()` com a mensagem "Chamei função", e chamaremos em seguida:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

        function limpaTela() {

        pincel.clearRect(0, 0, 600, 400);

    }

    function exibeMensagemNoConsole() {

        console.log('Chamei função');
```

```
    }

    exibeMensagemNoConsole();

</script>
```

Salvaremos a página e recarregaremos. Em seguida, abriremos o console e leremos:

```
Chamei função
```

Faremos com que esta mensagem seja exibida com intervalos de tempo. Para isso, utilizaremos a função `setInterval`. Ela aceita receber, como parâmetro, a função que você deseja chamar e, depois, a quantidade de tempo que desejamos dar de intervalo em milissegundos. Utilizaremos `1000` milissegundos, que equivalem a 1 segundo:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);
```

```
    pincel.fill();
  }
  function limpaTela() {
  pincel.clearRect(0, 0, 600, 400);
  }
  function exibeMensagemNoConsole() {
  console.log('Chamei função');
  }
  setInterval(exibeMensagemNoConsole, 1000);
</script>
```

Após recarregarmos a página, observe que o console exibe a mensagem, assim como anteriormente, e que surge um contador ao lado esquerdo da frase "Chamei função". Este contador se refere ao número de vezes em que ele está realizando a impressão, o JavaScript faz uso de um recurso mais elegante, em vez de gerar isto visualmente.

Escolhemos o tempo de `1` segundo mas poderíamos ter delimitado qualquer outro intervalo, isso dependerá do projeto e exigências específicas.

Entretanto, nosso objetivo não é imprimir a mensagem, e sim realizar a função `atualizaTela`, que criaremos agora:

```
<canvas width="600" height="400"></canvas>
<script>
  var tela = document.querySelector('canvas');
```

```
    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

    function limpaTela() {

    pincel.clearRect(0, 0, 600, 400);

    }

    function atualizaTela() {

    }

</script>
```

Sua funcionalidade será atualizar a tela em determinados intervalos de tempo. Ela chamará a função `desenhaCirculo()`, que terá como parâmetros as posições X, Y, e o raio da circunferência:

```
<canvas width="600" height="400"></canvas>
```

```
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    function desenhaCirculo(x, y, raio) {
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
    }
    function atualizaTela() {
        desenhaCirculo(20, 20, 10);
    }
    setInterval(atualizaTela, 20);
</script>
```

Lembre-se de salvar a página. Após recarregá-la, temos a exibição da esfera azul no canto superior esquerdo da tela. Ela não está animada, isso significa que não deu certo?

A nossa função está correta, entretanto, como passamos o mesmo valor de `x`, estão sendo desenhadas várias esferas sobrepostas. Para consertar isso, a cada chamada de tela, precisamos passar um novo valor de `x`, ou seja, incrementar `x` (lembrando de declarar a variável `x` com valor `20` antes):

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

    function limpaTela() {

        pincel.clearRect(0, 0, 600, 400);
```

```
    }
    var x = 20;
    function atualizaTela() {
    desenhaCirculo(x, 20, 10);
    x++;
    }
    setInterval(atualizaTela, 20);
</script>
```

Para que não tenhamos o problema de esferas sobrepostas, precisamos inserir, antes de `desenhaCirculo()`, o `limpaTela()`:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    function desenhaCirculo(x, y, raio) {
        pincel.fillStyle = 'blue';
        pincel.beginPath();
```

```
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
    }
    var x = 20;
    function atualizaTela() {
    limpaTela();
    desenhaCirculo(x, 20, 10);
    x++;
    }
    setInterval(atualizaTela, 20);
</script>
```

Salvaremos e recarregaremos a tela. Funcionou! Temos nossa esfera viajando do canto superior esquerdo da tela, até o canto superior direito. Para que a animação fique mais rápida, basta alterar o segundo parâmetro de `setInterval`, deixaremos como `10`:

```
<canvas width="600" height="400"></canvas>
```

```
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    function desenhaCirculo(x, y, raio) {
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
    }
    var x = 20;
    function atualizaTela() {
    limpaTela();
    desenhaCirculo(x, 20, 10);
    x++;
```

```
    }

    setInterval(atualizaTela, 10);

</script>
```

Recapitulando:

- O método `setInterval()` chama `atualizaTela()`;
- Este, quando executado, faz o `limpaTela()`, `desenhaCirculo()` e incrementa o valor de `x`;
- O `setInterval()` aguarda `20` milisegundos, para repetir o processo, sempre incrementando o valor de `x`, o que faz com que nossa esfera "se mova" para a direita.

Esta lógica é fundamental quando pensamos em animações, ou mesmo em criar jogos. Vamos para os exercícios!

028.Para saber mais - Onde está o erro, mais uma vez?

Herculano seguiu tudo o que foi apresentado no vídeo e tem certeza que entendeu cada detalhe. Contudo ele não consegue perceber onde errou no seu código. Aliás, isso acontece com qualquer pessoa programadora: ele sabe o que é correto fazer, mas na hora em que digita, comete esse ou aquele erro, e só vai perceber quando o código não funciona como o esperado.

Vamos analisar juntos o código de Herculano abaixo?

```
<canvas width="600" height="400"></canvas>

<script>

  var tela = document.querySelector('canvas');

  var pincel = tela.getContext('2d');

  pincel.fillStyle = 'lightgray';
```

```
  pincel.fillRect(0, 0, 600, 400);
  function desenhaCirculo(x, y, raio) {
    pincel.fillStyle = 'blue';
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
  }
  function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
  }
  var x = 20;
  function atualizaTela() {
    limpaTela();
    desenhaCirculo(x, 20, 10);
    x++;
  }
  setInterval(atualizaTela(), 10);
</script>
pt>
```

Nesse exercício de reflexão, vamos detectar onde ele errou?

O problema no código de Herculano está nesta linha:

```
setInterval(atualizaTela(), 10);
```

Lembre-se mais uma vez que a função `setInterval` ,que já vem pronta no JavaScript, precisa receber dois parâmetros para funcionar. O primeiro é a função que ela deve chamar e o segundo o intervalo de tempo no qual a função deve ser chamada.

Veja que não foi passada a função como parâmetro, mas o retorno dela. ***Como*** assim Flávio? Se você fizer `atualizaTela()` está chamando uma função, certo? Mas a função `atualizaTela` retorna alguma coisa? Ela possui algum return? Não! Sendo assim, para que fique mais claro ainda o motivo disso não funcionar, vamos dividir o código em duas etapas:

```
var retorno = atualizaTela();

setInterval(retorno, 10);
```

O código acima faz sentido? Não faz, porque `atualizaTela` não possui retorno, sendo assim, a variável `retorno` será `undefined` e será esse valor passado para o `setInterval`. Faz sentido `setInterval` chamar a cada 10 milissegundos algo que é `undefined`? Não mesmo!

Outro ponto é que queremos que o JavaScript, através do `setInterval`, chame a função e não faz sentido nós mesmos chamarmos `atualizaTela()`.

Então, para corrigir, basta fazermos:

```
// veja que não há mais o ()

 setInterval(atualizaTela, 10);
```

COPIAR CÓDIGO

A pergunta que lhe faço é. Se não estamos usando o `()` estamos chamando a função? Não, não estamos. O que fizemos foi passar a função inteira como parâmetro para `setInterval` que internamente a chamará para nosso programa.

Quando queremos passar uma função como parâmetro para outra, precisamos omitir seus parênteses (), chamamos isso de callback em programação

029.Para saber mais - Movendo a bolinha pelo teclado

O foco deste treinamento é na lógica de programação, mais do que trabalhar com o canvas em si. A ideia sempre foi fazer com que o aluno aplicasse seu conhecimento para fazer algo um pouco diferente do que calcular IMC ou pontos de vitória de um time. A parte de animação é muito mais complexa do que estamos vendo nesse treinamento, contudo acredito que você ficaria muito feliz se aprendesse a movimentar uma bolinha na tela com o teclado, não?

Vamos então dividir esse aprendizado em duas partes. A primeira eu lhe ensinarei como capturar as *arrow keys*, ou seja, aquelas teclas que são setas, geralmente usadas em joguinhos para andar com algo pela tela. Na segunda parte, vem o desafio de lógica que você deve implementar. Preparado?

Primeiro, vemos o seguinte trecho de código:

```
<canvas width="600" height="400"></canvas>

<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var x = 20;
    var y = 20;
    function desenhaCirculo(x, y, raio) {
        pincel.fillStyle = 'blue';
        pincel.beginPath();
```

```
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function atualizaTela() {
        limpaTela();
        desenhaCirculo(x, y, 10);
    }
    setInterval(atualizaTela, 20);
</script>
```

É um código que não é nenhuma novidade para nós. A cada 20 milissegundos a função `atualizaTela` é chamada e desenha um círculo na tela. Veja que a posição do círculo é sempre a mesma, sendo definida nas variáveis `x` e `y`, ambas com o valor 20. Do jeito que está, a bolinha será desenhada uma vez e nada mais acontecerá, porque a função `atualizaTela` apaga a tela, desenhando a bolinha mais uma vez.

E se fôssemos capazes de mudar o valor de `x` e `y` pelo teclado? Se a cada 20 milissegundos a tela é atualizada e a função `desenhaCirculo` sempre utiliza as variáveis x e y, qualquer mudança nesses valores pelo teclado fará com que o círculo desenhado mude de posição.

Identificando qual tecla foi pressionada

Em JavaScript, existe o evento `onkeydown`, que permite identificar qual tecla está pressionada. Esse evento é o único capaz de identificar também as setas do teclado. Contudo, até agora todos os eventos que associamos foi com nossa `tela`, mas dessa vez quem deve responder ao evento é `document`. E `document`, nosso oráculo que sabe tudo o que a página possui, é o cara que fica escutando ao teclado. Então, vou alterar o código e criar a função `leDoTeclado` e associá-la ao `document` através do evento `onkeydown`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    var x = 20;

    var y = 20;

    // códigos do teclado

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }
```

```
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function atualizaTela() {
        limpaTela();
        desenhaCirculo(x, y, 10);
    }
    setInterval(atualizaTela, 20);
    function leDoTeclado(evento) {
        // como saber qual tecla foi pressionada?
    }
   document.onkeydown = leDoTeclado;
</script>
```

A função `leDoTeclado` será chamada toda vez que uma tecla for pressionada. Mas para podermos identificar as setas do teclado, precisamos saber qual é seu código correspondente. Isso porque na função `leDoTeclado` podemos acessar `evento.keyCode`. O `evento.keyCode` traz o código da tecla que foi pressionada. Vamos declarar quatro variáveis que guardam os códigos que correspondem às nossas setinhas:

Obs: Cada tecla possui um KeyCode (código de tecla) respectivo e isso foi catalogado em uma tabela. Essa tabela deve ser usada pelos navegadores web para que usem os mesmos valores.

```
<canvas width="600" height="400"></canvas>
<script>
```

```
var tela = document.querySelector('canvas');
var pincel = tela.getContext('2d');
pincel.fillStyle = 'lightgray';
pincel.fillRect(0, 0, 600, 400);
var x = 20;
var y = 20;
// códigos do teclado
var esquerda = 37;
var cima = 38;
var direita = 39;
var baixo = 40;
function desenhaCirculo(x, y, raio) {
    pincel.fillStyle = 'blue';
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
}
function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
}
```

```
    function atualizaTela() {
        limpaTela();
        desenhaCirculo(x, y, 10);
    }
    setInterval(atualizaTela, 20);
    function leDoTeclado(evento) {
        // sabemos que é através de evento.keyCode que temos acesso ao código da tecla pressionada
    }
   document.onkeydown = leDoTeclado;
</script>
```

Estamos quase lá. Outra coisa importante é que a taxa de atualização de `x` e `y` seja 10, isto é, toda vez que teclarmos com a seta esquerda, por exemplo, precisamos subtrair -10 do valor de `x` atual. Se teclarmos a seta direita, precisamos incrementar +10 com o `x` atual. A mesma lógica se aplica ao eixo `y`.

Então, vamos declarar a variável chamada `taxa`, que guarda o valor do incremento dos eixos:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
```

```
    pincel.fillRect(0, 0, 600, 400);
    var x = 20;
    var y = 20;
    // códigos do teclado
    var esquerda = 37;
    var cima = 38;
    var direita = 39;
    var baixo = 40;
    // taxa de incremento
    var taxa = 10;
    function desenhaCirculo(x, y, raio) {
        pincel.fillStyle = 'blue';
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function atualizaTela() {
```

```
        limpaTela();
        desenhaCirculo(x, y, 10);
    }
    setInterval(atualizaTela, 20);
    function leDoTeclado(evento) {
        // sua lógica virá aqui
    }
    document.onkeydown = leDoTeclado;
</script>
```

Agora precisamos finalizar a função `leDoTeclado`, que deve testar cada tecla pressionada e atualizar `x` e `y` de acordo com a tecla.

O código para essa funcionalidade é uma sucessão de `if`'s que identifica qual tecla foi pressionada, incrementando ora `x`, ora `y`, positivamente ou negativamente. Nessa implementação podemos andar com a bolinha até que suma da tela:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var x = 20;
```

```
var y = 20;
// códigos do teclado
var esquerda = 37
var cima = 38
var direita = 39
var baixo = 40
var taxa = 10;
function desenhaCirculo(x, y, raio) {
    pincel.fillStyle = 'blue';
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
}
function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
}
function atualizaTela() {
    limpaTela();
    desenhaCirculo(x, y, 10);
}
```

```
    setInterval(atualizaTela, 20);

    function leDoTeclado(evento) {

        if(evento.keyCode == cima) {

            y = y - taxa;

        } else if (evento.keyCode == baixo) {

            y = y + taxa;

        } else if (evento.keyCode == esquerda) {

            x = x - taxa;

        } else if (evento.keyCode == direita) {

            x = x + taxa;

        }

    }

    document.onkeydown = leDoTeclado;

</script>
```

Muito interessante exercitar a lógica! Continue com os estudos!

030.Trocando bandeiras

Você se lembra das bandeiras que desenhamos no primeiro capítulo? Nesse exercício vamos revê-las em detalhes, vamos usar as bandeiras do Brasil e da Alemanha. A ideia é homenagear a final de futebol masculino das Olimpíadas e claro, aprender e praticar mais lógica de programação.

Repare no código abaixo, que já temos duas funções preparadas que desenham as bandeiras:

```
<!-- bandeiras.html -->
<canvas width="600" height="400"></canvas>
<script>
function desenhaBandeiraBrasil() {
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle="green";
    pincel.fillRect(0, 0, 600, 400);
    pincel.fillStyle="yellow";
    pincel.beginPath();
    pincel.moveTo(300, 50);
    pincel.lineTo(50, 200);
    pincel.lineTo(550, 200);
    pincel.fill();
    pincel.beginPath();
    pincel.moveTo(50, 200);
    pincel.lineTo(300, 350);
    pincel.lineTo(550, 200);
```

```
    pincel.fill();
    pincel.fillStyle="darkblue";
    pincel.beginPath();
    pincel.arc(300, 200, 100, 0, 2* 3.14);
    pincel.fill();
}
function desenhaBandeiraAlemanha() {
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'black';
    pincel.fillRect(0, 0, 600, 133);
    pincel.fillStyle = 'red';
    pincel.fillRect(0, 133, 600, 133);
    pincel.fillStyle = 'yellow';
    pincel.fillRect(0, 266, 600, 133);
}
</script>
```

Para realmente ver uma bandeira, basta chamar uma função, mas a sua tarefa não é essa :)

Você deve mostrar a bandeira do Brasil por 3 segundos e depois da Alemanha, sempre intercalando! Para tal, tente escrever uma função `trocaBandeira` e use a função `setInterval` para chamá-la a cada 3 segundos.

O primeiro passo é escrever a função que fará a troca:

```
function trocaBandeira() {

    // aqui devemos chamar desenhaBandeiraAlemanha() ou desenhaBandeiraBrasil()

}
```

Ainda falta a implementação, mas nada impede de já usá-la na função `setInterval`:

```
function trocaBandeira() {

    //aqui devemos chamar desenhaBandeiraAlemanha() ou desenhaBandeiraBrasil()

}

setInterval(trocaBandeira, 3000); // a cada 3 segundos, chama trocaBandeira
```

COPIAR CÓDIGO

Esse foi a parte fácil, agora é preciso pensar como fazer a troca. O segredo é se lembrar de alguma forma qual bandeira devemos desenhar. Então vamos guardar um valor dentro de uma variável que ajudará nessa decisão. Como temos apenas duas bandeiras, utilizarei um *booleano* que pode ter os valores `true` ou `false` apenas:

```
var mostraBrasil = true; // a ideia é começar com a bandeira do Brasil


function trocaBandeira() {
   // aqui vem mais
}
```

Se a variável for verdadeira (true) devemos desenhar a bandeira do Brasil, **senão** da Alemanha:

```
var mostraBrasil = true;
function trocaBandeira() {
   if(mostraBrasil == true) {
      desenhaBandeiraBrasil();
   } else {
      desenhaBandeiraAlemanha();
   }
}
```

Ótimo, mas o problema ainda é que a variável mostraBrasil sempre está true, ou seja, nunca desenhamos a outra bandeira. Por isso devemos alterar o valor em cada bloco do if/else:

```
var mostraBrasil = true;
```

```
function trocaBandeira() {
    if(mostraBrasil == true) {
        desenhaBandeiraBrasil();
        mostraBrasil = false;
    } else {
        desenhaBandeiraAlemanha();
        mostraBrasil = true;
    }
}
setInterval(trocaBandeira, 3000); // a cada 3 segundos, chama trocaBandeira
```

Agora sim, o código funciona e troca as bandeiras!

O código é totalmente funcional mas podemos melhorá-lo para deixá-lo mais elegante e enxuto. Repare que estamos comparando no `if` se o valor da variável `mostraBrasil` é verdadeiro:

```
if(mostraBrasil == true) {
```

Se `mostraBrasil` for `true`, o resultado da comparação é `true`. Se `mostraBrasil` for `false`, o resultado da comparação é `false`. Ou seja, a comparação sempre tem o mesmo resultado do valor da variável, por isso podemos simplificar e eliminar a comparação:

```
if(mostraBrasil) {
```

Além disso, repare que estamos invertendo *sempre* o valor do `mostraBrasil`. Em outras palavras, `mostraBrasil` sempre muda o valor, ou de `true` para `false`, ou de `false` para `true`. O melhor seria colocar essa troca de valor em um lugar só, algo assim:

```
function trocaBandeira() {
  if(mostraBrasil) {
      desenhaBandeiraBrasil();
  } else {
      desenhaBandeiraAlemanha();
  }
  mostraBrasil = // aqui inverte o valor
}
```

A pergunta é: como podemos inverter um booleano? A resposta é através do operador lógico **NOT**, que é representado no mundo JavaScript pelo símbolo `!`. Sabendo disso chegamos à versão final da função `trocaBandeira`:

```
function trocaBandeira() {
  if(mostraBrasil) {
      desenhaBandeiraBrasil();
  } else {
```

```
        desenhaBandeiraAlemanha();
    }
    mostraBrasil = !mostraBrasil;
}
```

O importante não é só escrever o código, é também pensar como podemos simplificá-lo e deixá-lo mais legível. O desenvolvedor gasta muito mais tempo lendo código do que realmente escrevendo. Ao escrever um código mais limpo, facilitamos a compreensão do mesmo.

Nosso primeiro jogo

031.Nosso primeiro jogo

Nesta aula aprenderemos a criar um jogo com mais complexidade.

Como podemos observar, na tela branca temos apenas a imagem de um alvo vermelho, formado por três círculos, uma esfera vermelha central, um círculo branco exterior, e por fim, um círculo vermelho.

Este alvo se move aleatoriamente na tela, pulando entre as posições dentro do `<canvas>`. Ou seja, se estamos falando de aleatoriedade, lembrando das aulas anteriores, sabemos que temos o recurso do `Math.random`, com o qual podemos gerar coordenadas para os eixos X e Y.

O alvo, a cada `1` segundo ou milissegundo, é plotado em diversas posições na tela. Sendo assim, há uma função para desenhar o alvo, em determinados intervalos de tempo, e o fará com base em posições aleatórias.

O objetivo do jogo é que o usuário clique no alvo antes que este desapareça. Ao clicar no centro do alvo, surge um pop up com a mensagem "acertou!". Clicando em qualquer outro lugar da tela resulta em erro.

É um jogo simples, no qual utilizaremos o raciocínio lógico e alguns conceitos de elaboração de jogos. Assim, você será capaz de criar suas próprias brincadeiras.

Nas próximas aulas passaremos pelo processo de criação, passo a passo.

032.O alvo aleatório!

Para darmos início à elaboração de nosso jogo, temos que começar pelo desenho do alvo.

Este objeto é construído pela função `desenhaCirculo()`, que já temos em nosso código:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio) {

        pincel.fillStyle = 'blue';

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

    function limpaTela() {

        pincel.clearRect(0, 0, 600, 400);
```

```
    }
</script>
```

Ela deverá receber mais um parâmetro (a `cor`) e será utilizado no `fillStyle`. Temos isso porque haverá círculos na cor branca e vermelha:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
</script>
```

Além disso, temos que saber que nosso raio padrão terá o valor inicial de `10`, os demais serão maiores, para criar a imagem de um alvo.

Para efetivamente desenharmos o círculos, temos que chamar a função `desenhaCirculo()`, com os parâmetros `x` e `y` ambos com valor `100`, raio `10` e a cor vermelha (`red`), já que estamos lidando com o círculo central:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio, cor) {

        pincel.fillStyle = cor;

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

    function limpaTela() {

        pincel.clearRect(0, 0, 600, 400);

    }

    desenhaCirculo(100, 100, 10, 'red');
```

```
</script>
```

O próximo círculo será construído na mesma posição, entretanto, o seu raio será maior, equivalente a `30`, e sua cor será branca (`white`):

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio, cor) {

        pincel.fillStyle = cor;

            pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

    function limpaTela() {

        pincel.clearRect(0, 0, 600, 400);

    }

    desenhaCirculo(100, 100, 10, 'red');

    desenhaCirculo(100, 100, 30, 'white');
```

```
</script>
```

Por fim, desenharemos um círculo ainda maior, com raio `40`, e novamente na cor vermelha (`red`):

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio, cor) {

        pincel.fillStyle = cor;

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();

    }

    function limpaTela() {

        pincel.clearRect(0, 0, 600, 400);

    }

    desenhaCirculo(100, 100, 10, 'red');

    desenhaCirculo(100, 100, 30, 'white');
```

```
    desenhaCirculo(100, 100, 40, 'red');

</script>
```

Salvaremos e recarregaremos a página. Por que será exibido somente um círculo vermelho? Porque a última função que passamos foi para a criação de um círculo maior que todos os demais, assim, ela cobriu todas as outras. Como consertaremos isso?

Basta invertermos a ordem de chamamento das funções, se chamarmos o círculo vermelho maior primeiro, e o círculo vermelho menor por último, da seguinte forma:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    function desenhaCirculo(x, y, raio, cor) {

    pincel.fillStyle = cor;

    pincel.beginPath();

    pincel.arc(x, y, raio, 0, 2 * Math.PI);

    pincel.fill();

    }

    function limpaTela() {
```

```
    pincel.clearRect(0, 0, 600, 400);
    }
    desenhaCirculo(100, 100, 40, 'red');
    desenhaCirculo(100, 100, 30, 'white');
    desenhaCirculo(100, 100, 10, 'red');
</script>
```

Aprimoraremos nosso código, usando o raio inicial de `10` como parâmetro, guardando-o em uma variável. Em seguida, acrescentaremos os valores necessários para compor os raios das demais circunferências. Criaremos a variável `raio = 10` a seguir:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
  function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
```

```
        pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    desenhaCirculo(100, 100, 40, 'red');
    desenhaCirculo(100, 100, 30, 'white');
    desenhaCirculo(100, 100, 10, 'red');
</script>
```

Nos chamamentos das funções, trocaremos os valores pela variável:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
```

```
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
    }
    function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
    }
    desenhaCirculo(100, 100, raio+20, 'red');
    desenhaCirculo(100, 100, raio+10, 'white');
    desenhaCirculo(100, 100, raio, 'red');
</script>
```

Salvaremos e recarregaremos a página. Pronto! Temos nosso alvo.

Com o alvo criado, precisamos fazer com que seu aparecimento seja aleatório, ou seja, que sejam criadas coordenadas arbitrárias. Para isso, vamos inserir as instruções para a criação do alvo dentro de uma nova função chamada `desenhaAlvo()`, e a chamaremos logo em seguida:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
```

```
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo() {
        desenhaCirculo(100, 100, raio+20, 'red');
        desenhaCirculo(100, 100, raio+10, 'white');
        desenhaCirculo(100, 100, raio, 'red');
    }
    desenhaAlvo();
</script>
```

Salvaremos e recarregaremos e podemos ver que o alvo permanece inalterado.

Como queremos que o alvo seja aleatório, a função `desenhaAlvo()` receberá parâmetros referentes às coordenadas X (até o limite de 600) e Y (até o limite de 400), com relação ao `desenhaCirculo()`, elas receberão estes parâmetros, respectivamente:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    var raio = 10;

    function desenhaCirculo(x, y, raio, cor) {

    pincel.fillStyle = cor;

    pincel.beginPath();

    pincel.arc(x, y, raio, 0, 2 * Math.PI);

    pincel.fill();

    }

    function limpaTela() {

    pincel.clearRect(0, 0, 600, 400);

    }

    function desenhaAlvo(x, y) {
```

```
    desenhaCirculo(x, y, raio+20, 'red');
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
    }
    desenhaAlvo();
</script>
```

Para chamar o `desenhaAlvo()`, passaremos os parâmetros `(200, 200)`:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
    pincel.fillStyle = cor;
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
    }
```

```
    function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo(x, y) {
    desenhaCirculo(x, y, raio+20, 'red');
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
    }
    desenhaAlvo(200, 200);
</script>
```

A posição do alvo variará de acordo com os valores que passarmos como parâmetros em `desenhaAlvo()`, passaremos, por exemplo `(50, 200)`, e teremos uma posição diferente.

O próximo passo será fazer com que os valores das coordenadas sejam gerados aleatoriamente.

Criaremos uma função chamada `sorteiaPosicao()`, que receberá um valor `maximo` que pode sortear entre 0 a 600 (para o eixo X), e 0 a 400 (para o eixo Y):

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
```

```
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo(x, y) {
        desenhaCirculo(x, y, raio+20, 'red');
        desenhaCirculo(x, y, raio+10, 'white');
        desenhaCirculo(x, y, raio, 'red');
    }
    function sorteiaPosicao(maximo) {
    }
    desenhaAlvo(50, 200);
</script>
```

Criaremos uma variável `xAleatorio` que recebe `sorteiaPosicao`, já que estamos trabalhando com a variável X, então o valor máximo será `600`. Do mesmo modo, criaremos uma variável `yAleatorio` que receberá `sorteiaPosicao`, cujo máximo será `400`:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    var raio = 10;

    function desenhaCirculo(x, y, raio, cor) {

    pincel.fillStyle = cor;

    pincel.beginPath();

    pincel.arc(x, y, raio, 0, 2 * Math.PI);

    pincel.fill();

    }

    function limpaTela() {

    pincel.clearRect(0, 0, 600, 400);

    }

    function desenhaAlvo(x, y) {

    desenhaCirculo(x, y, raio+20, 'red');
```

```
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
}
    function sorteiaPosicao(maximo) {
    }
    var xAleatorio = sorteiaPosicao(600);
    var yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(50, 200);
</script>
```

O próximo passo é passarmos estes valores para a função `desenhaAlvo()`:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
    pincel.beginPath();
```

```
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
    }
    function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo(x, y) {
    desenhaCirculo(x, y, raio+20, 'red');
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
    }
     function sorteiaPosicao(maximo) {
    }
    var xAleatorio = sorteiaPosicao(600);
    var yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(xAleatorio, yAleatorio);
</script>
```

Ainda precisamos programar a função `sorteiaPosicao()`. Ela nos dará um retorno - `return` - de `Math.floor`, que arredonda o número para baixo, diferente do `Math.round`, que o arredonda para cima. Em seguida temos `Math.random() * maximo):

```
<canvas width="600" height="400"></canvas>
```

```
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo(x, y) {
    desenhaCirculo(x, y, raio+20, 'red');
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
    }
```

```
    function sorteiaPosicao(maximo) {

    return Math.floor(Math.random() * maximo);

    }

    var xAleatorio = sorteiaPosicao(600);

    var yAleatorio = sorteiaPosicao(400);

    desenhaAlvo(xAleatorio, yAleatorio);

</script>
```

Sempre que salvarmos e recarregaremos a página, o alvo aparecerá em um novo ponto do nosso `<canvas>`.

Nosso próximo objetivo será fazer com que o alvo seja redesenhado em certos intervalos de tempo, pré-determinados.

Para isso, criaremos uma função `atualizaTela()`, que receberá as variáveis referentes ao X e Y aleatórios, bem como o chamamento de `desenhaAlvo`, da seguinte forma:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    var raio = 10;

    function desenhaCirculo(x, y, raio, cor) {
```

```
    pincel.fillStyle = cor;
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();


    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo(x, y) {
    desenhaCirculo(x, y, raio+20, 'red');
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
    }
    function sorteiaPosicao(maximo) {
    return Math.floor(Math.random() * maximo);
    }
    function atualizaTela() {
    var xAleatorio = sorteiaPosicao(600);
    var yAleatorio = sorteiaPosicao(400);
```

```
    desenhaAlvo(xAleatorio, yAleatorio);
    }
</script>
```

Em seguida, utilizaremos o `setInterval()`, com os parâmetros `atualizaTela` e o intervalo de tempo `500`, sem esquecer de incluir na função o `limpaTela`:

```
<canvas width="600" height="400"></canvas>
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
    pincel.fillStyle = cor;
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
```

```
    }
    function desenhaAlvo(x, y) {
        desenhaCirculo(x, y, raio+20, 'red');
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
    }
    function sorteiaPosicao(maximo) {
    return Math.floor(Math.random() * maximo);
    }
    function atualizaTela() {
    limpaTela();
        var xAleatorio = sorteiaPosicao(600);
    var yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(xAleatorio, yAleatorio);
    }
    setInterval(atualizaTela, 500);
</script>
```

Salvaremos e recarregaremos. Pronto! Temos nosso alvo viajando pela tela, em pontos aleatórios. Como ainda está rápido, trocaremos o intervalo de tempo para `1000`:

```
<canvas width="600" height="400"></canvas>
```

```
<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo(x, y) {
    desenhaCirculo(x, y, raio+20, 'red');
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
```

```
    }
    function sorteiaPosicao(maximo) {
        return Math.floor(Math.random() * maximo);
    }
    function atualizaTela() {
    limpaTela();
    var xAleatorio = sorteiaPosicao(600);
    var yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(xAleatorio, yAleatorio);
    }
    setInterval(atualizaTela, 1000);
</script>
```

Nosso próximo objetivo é tornar o objeto responsivo ao clique, com o alerta de que o usuário acertou ao clicar sobre o alvo.

033.Disparando contra o alvo

Dando continuidade às aulas anteriores, nesta, aprenderemos a tornar nosso jogo responsivo ao clique do usuário.

Criaremos uma função chamada dispara, que será executada ao clique de um botão, ou seja, ela pode receber o evento como parâmetro, para sabermos qual é a posição do mouse:

```
// parte de cima do código omitida
```

```
function atualizaTela() {
    limpaTela();
    var xAleatorio = sorteiaPosicao(600);
    var yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(xAleatorio, yAleatorio);
}
setInterval(atualizaTela, 1000);
function dispara(evento) {
}
</script>
```

Queremos disparar ao clicar no <canvas>, para isso, temos que fazer com que o tela.onclick receba dispara, assim, todas as vezes em que o <canvas> for clicado o código da função dispara será executado:

```
// parte de cima do código omitida
function atualizaTela() {
    limpaTela();
    var xAleatorio = sorteiaPosicao(600);
    var yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(xAleatorio, yAleatorio);
}
```

```
setInterval(atualizaTela, 1000);

function dispara(evento) {

}

tela.onclick = dispara;

</script>
```

Precisamos saber, também, onde o usuário clicou, para determinar se foi ou não sobre o alvo. Para isso, teremos - dentro da função `dispara`, uma variável `x` que recebe `evento.pageX` e outra `y` que recebe `evento.pageY`. Além disso, precisaremos remover o `tela.offsetLeft`, para o eixo X, e o `tela.offsetTop` para o eixo Y:

```
// parte de cima do código omitida

function atualizaTela() {

    limpaTela();

    var xAleatorio = sorteiaPosicao(600);

    var yAleatorio = sorteiaPosicao(400);

    desenhaAlvo(xAleatorio, yAleatorio);

}

setInterval(atualizaTela, 1000);

function dispara(evento) {

    var x = evento.pageX - tela.offsetLeft;

    var y = evento.pageY - tela.offsetTop;

}
```

```
tela.onclick = dispara;

</script>
```

Faremos então a lógica de detecção. Criaremos um `if`, com um `alert` contendo a mensagem `Acertou!`. Precisamos nos certificar de que as coordenas X e Y estão dentro do escopo do alvo, entretanto, temos de saber também o X aleatório e o Y aleatório do alvo. Só que estas variáveis foram declaradas dentro do `atualizaTela`:

```
// parte de cima do código omitida

function atualizaTela() {

    limpaTela();

    var xAleatorio = sorteiaPosicao(600);

    var yAleatorio = sorteiaPosicao(400);

    desenhaAlvo(xAleatorio, yAleatorio);

}

setInterval(atualizaTela, 1000);

function dispara(evento) {

    var x = evento.pageX - tela.offsetLeft;

    var y = evento.pageY - tela.offsetTop;

    if() {

        alert('Acertou!');

    }

}
```

```
tela.onclick = dispara;

</script>
```

E uma variável declarada com `var`, dentro de uma função, existe **somente** dentro desta, só pode ser acessada neste contexto, portanto, se tentarmos acessar o `xAleatorio` fora da função `atualizaTela`, o programa informará que `xAleatorio` não foi definido.

Como podemos então ter acesso a estas variáveis, para que possamos executar nossa lógica?

Primeiro, removeremos o `var` de dentro da função `atualizaTela`, e as inicializaremos logo abaixo de `var raio = 10`, e elas serão iniciadas sem valor nenhum:

```
<canvas width="600" height="400"></canvas>



<script>
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var raio = 10;
        var xAleatorio = sorteiaPosicao;
        var yAleatorio = sorteiaPosicao;
    function desenhaCirculo(x, y, raio, cor) {
        pincel.fillStyle = cor;
```

```
        pincel.beginPath();
        pincel.arc(x, y, raio, 0, 2 * Math.PI);
        pincel.fill();
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo() {
        desenhaCirculo(x, y, raio+20, 'red');
        desenhaCirculo(x, y, raio+10, 'white');
        desenhaCirculo(x, y, raio, 'red');
    }
    function sorteiaPosicao(maximo) {
        return Math.floor(Math.random() * maximo);
    }
function atualizaTela() {x
    limpaTela();
    xAleatorio = sorteiaPosicao(600);
    yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(xAleatorio, yAleatorio);
```

```
}

setInterval(atualizaTela, 1000);

function dispara(evento) {

    var x = evento.pageX - tela.offsetLeft;

    var y = evento.pageY - tela.offsetTop;

    if() {

        alert('Acertou!');

    }

}

tela.onclick = dispara;

</script>
```

Como agora as variáveis foram inicializadas fora da função, elas podem ser lidas e inicializadas por quaisquer uma das funções presentes. Agora, podemos acessar o `xAleatorio` e o `yAleatorio` dentro da função `dispara`. Por enquanto, deixaremos o `if` em comentários:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);
```

```
var raio = 10;
    var xAleatorio = sorteiaPosicao;
    var yAleatorio = sorteiaPosicao;
function desenhaCirculo(x, y, raio, cor) {
    pincel.fillStyle = cor;
    pincel.beginPath();
    pincel.arc(x, y, raio, 0, 2 * Math.PI);
    pincel.fill();
}
function limpaTela() {
    pincel.clearRect(0, 0, 600, 400);
}
function desenhaAlvo() {
    desenhaCirculo(x, y, raio+20, 'red');
    desenhaCirculo(x, y, raio+10, 'white');
    desenhaCirculo(x, y, raio, 'red');
}
function sorteiaPosicao(maximo) {
    return Math.floor(Math.random() * maximo);
}
```

```
function atualizaTela() {
    limpaTela();
    xAleatorio = sorteiaPosicao(600);
    yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(xAleatorio, yAleatorio);
}
setInterval(atualizaTela, 1000);
function dispara(evento) {
    var x = evento.pageX - tela.offsetLeft;
    var y = evento.pageY - tela.offsetTop;
    alert(xAleatorio);
    alert(yAleatorio);
    /*
    if() {
        alert('Acertou!');
    }
    */
}
tela.onclick = dispara;
</script>
```

Salvaremos e recarregaremos. Ao clicarmos no alvo, recebemos um alerta com a mensagem "574". Este é o valor de X, e se pressionarmos "Ok", temos o valor de Y, que é "227".

Agora que temos acesso às posições aleatórias, podemos fazer testes.

Usaremos o if para ver se x é maior que xAleatorio subtraído do raio, e, utilizando o &&, se x é menor que xAleatorio somado ao raio. Além disso, teremos as mesmas condições, replicadas para o y:

```
<canvas width="600" height="400"></canvas>

<script>

    var tela = document.querySelector('canvas');

    var pincel = tela.getContext('2d');

    pincel.fillStyle = 'lightgray';

    pincel.fillRect(0, 0, 600, 400);

    var raio = 10;

    var xAleatorio;

    var yAleatorio;

    function desenhaCirculo(x, y, raio, cor) {

        pincel.fillStyle = cor;

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * Math.PI);

        pincel.fill();
```

```
    }
    function limpaTela() {
        pincel.clearRect(0, 0, 600, 400);
    }
    function desenhaAlvo(x,y) {
        desenhaCirculo(x, y, raio+20, 'red');
        desenhaCirculo(x, y, raio+10, 'white');
        desenhaCirculo(x, y, raio, 'red');
    }
    function sorteiaPosicao(maximo) {
        return Math.floor(Math.random() * maximo);
    }
function atualizaTela() {
    limpaTela();
    xAleatorio = sorteiaPosicao(600);
    yAleatorio = sorteiaPosicao(400);
    desenhaAlvo(xAleatorio, yAleatorio);
}
setInterval(atualizaTela, 1000);
function dispara(evento) {
```

```
    var x = evento.pageX - tela.offsetLeft;

    var y = evento.pageY - tela.offsetTop;

    if((x > xAleatorio - raio)

    && (x < xAleatorio + raio)

    && (y > yAleatorio - raio)

    && (y < yAleatorio + raio)) {

        alert('Acertou!');

    }

}

tela.onclick = dispara;

</script>
```

Salvaremos e recarregaremos a página. Temos nosso alvo pulando entre pontos aleatórios na página e, ao clicarmos sobre ele, surge uma alerta com a mensagem "Acertou!". Se clicarmos em qualquer outro ponto, nada acontece.

Com isso, concluímos a implementação de nosso primeiro jogo!

034.Desenhe obras de arte

No terceiro capítulo aprendemos a desenhar com o mouse. Temos aqui um código muito semelhante com algumas diferenças. A primeira é que declarei primeiro todas as funções no início para depois então utilizá-las, algo que não fizemos desde o início do treinamento, para que você conseguisse enxergar passo a passo no arquivo tudo o que foi feito.

Outra mudança é que agora temos a função `lidaComMovimentoDoMouse`, que possui apenas a responsabilidade de obter as coordenadas e desenhar ou não um círculo. Sendo assim, a função `desenhaCirculo` só precisa se preocupar em desenhar um círculo e receber as coordenadas x e y de `lidaComMovimentoDoMouse`.

Por fim, visualmente, a grande diferença é que temos agora uma paleta com três cores sendo exibida na tela em forma de quadrado. Salve o código a seguir e visualize em seu navegador:

```
<meta charset="UTF-8">

<canvas width="600" height="400"></canvas>

<script>

    function desenhaQuadrado(x, y, tamanho, cor) {

        pincel.fillStyle = cor;

        pincel.fillRect(x, y, tamanho, tamanho)

        pincel.fill();

    }

    function desenhaCirculo(x, y, raio, cor) {

        pincel.fillStyle = cor;

        pincel.beginPath();

        pincel.arc(x, y, raio, 0, 2 * 3.14);

        pincel.fill();

    }

    function desenhaPaletaDeCores() {

        desenhaQuadrado(xVermelho, yQuadrados, tamanhoQuadrados, 'red');

        desenhaQuadrado(xVerde, yQuadrados, tamanhoQuadrados, 'green');

        desenhaQuadrado(xAzul, yQuadrados, tamanhoQuadrados, 'blue');
```

```
    }
    function lidaComMovimentoDoMouse(evento) {
        var x = evento.pageX - tela.offsetLeft;
        var y = evento.pageY - tela.offsetTop;
        if(desenha) {
            desenhaCirculo(x, y, 5, corAtual);
        }
    }
    function habilitaDesenhar() {
        desenha = true;
    }
    function desabilitaDesenhar() {
        desenha = false;
    }
    var tela = document.querySelector('canvas');
    var pincel = tela.getContext('2d');
    pincel.fillStyle = 'lightgray';
    pincel.fillRect(0, 0, 600, 400);
    var desenha = false;
    var corAtual = 'blue';
```

```
    var xVermelho = 0;

    var xVerde  = 50;

    var xAzul = 100;

    var yQuadrados = 0;

    var tamanhoQuadrados = 50;

    desenhaPaletaDeCores(); // mostra os quadrados de seleção de cores

    tela.onmousemove = lidaComMovimentoDoMouse;

    tela.onmousedown = habilitaDesenhar;

    tela.onmouseup = desabilitaDesenhar;

</script>
```

O desafio desse exercício é trocar a cor utilizada para desenhar o círculo de acordo com o quadrado da cor que clicarmos. Se clicarmos no verde, usaremos a cor verde quando formos desenhar e por aí vai. Outro ponto é que não podemos desenhar em cima da nossa paleta, ou melhor, não podemos desenhar na linha inteira na qual faz parte.

Para solucionar esse problema, você precisará identificar qual quadrado foi clicado, implementando uma lógica de colisão parecida com a que usamos para definir se acertamos um alvo ou não. Veja que como estamos trabalhando com um quadrado, não usamos um raio na lógica de colisão, usamos o tamanho do quadrado, guarde essa dica!

Documentação do Canva:

https://developer.mozilla.org/pt-BR/docs/Web/API/CanvasRenderingContext2D