start by alura

Aula 6

Será que podemos utilizar Pitágoras?

▶ Unidade

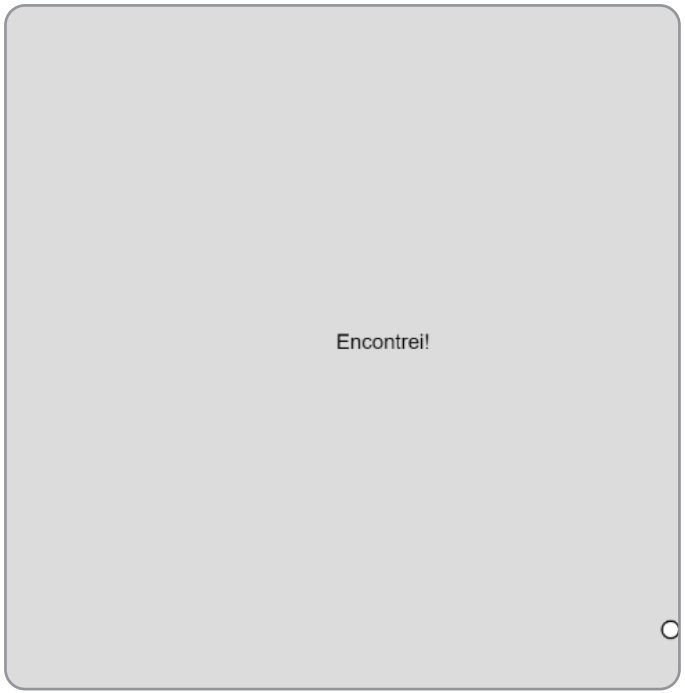
Lógica de programação: criando arte interativa com p5.js

O que vamos aprender?

- Aplicar operadores lógicos para combinar múltiplas condições em uma estrutura condicional.
- Compreender o conceito de distância e sua aplicação em projetos interativos.
- Implementar o Teorema de Pitágoras para calcular distâncias em um plano cartesiano.

ACESSE A PLATAFORMA START

Aplicando o Teorema de Pitágoras



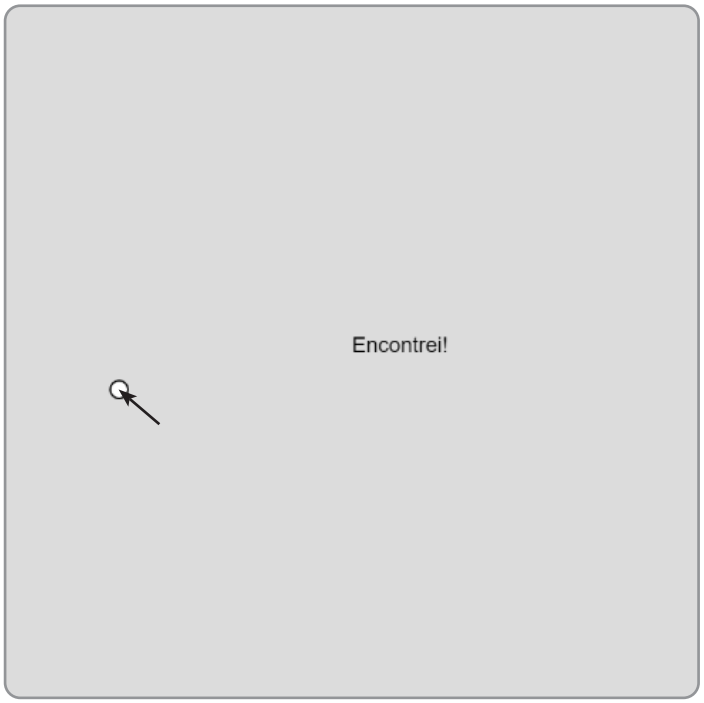
Na aula passada, começamos o projeto Quente e frio adicionando um círculo ao cenário, além de uma função que verifica quando a posição X do círculo e a do ponteiro do mouse são iguais na tela. O desafio agora será utilizar nossos conhecimentos de trigonometria para encontrar a menor distância entre o ponteiro do mouse e o círculo.

O projeto já consegue verificar se o ponteiro do mouse está na mesma posição que o círculo para o eixo X. Dessa forma, mesmo que o círculo esteja em uma posição mais acima ou mais abaixo do mouse, se eles estiverem horizontalmente alinhados, a mensagem “Encontrei!” ainda aparece. Para corrigir isso, precisamos incluir uma verificação também para a posição no eixo Y.

Faremos essa verificação dentro da condicional que já existe no projeto, utilizando o operador **&&** (que significa “e”). Dessa forma, a mensagem só será exibida caso as duas condições sejam atendidas ao mesmo tempo, ou seja: tanto a posição X quanto a posição Y do ponteiro do mouse devem ser iguais às posições X e Y do círculo. O código ficará assim:

```
if(mouseX == x && mouseY == y) {
    Text('Encontrei!', 200, 200);
}
```

Agora, ao executar o programa, para que a mensagem seja exibida, você precisará posicionar o mouse exatamente no centro do círculo:



Você deve ter percebido que a mensagem “Encontrei!” aparece e, logo em seguida, é apagada. Vamos programar para que, após o círculo ser encontrado, o projeto pare. Isso pode ser feito por meio da função **noLoop**. Veja o código:

```
if(mouseX == x && mouseY == y) {
    Text('Encontrei!', 200, 200);
    noLoop();
}
```

Ao executar o programa, você verá que a mensagem fica fixa na tela após o ponteiro do mouse e o círculo se encontrarem.

Para tornar nosso jogo mais desafiador, precisamos ocultar da tela o círculo que criamos dentro da função **`circle`**. Para isso, comentaremos a linha na qual o círculo é desenhado. Comentaremos também a linha do **`console.log`**, pois não precisamos dela por enquanto:

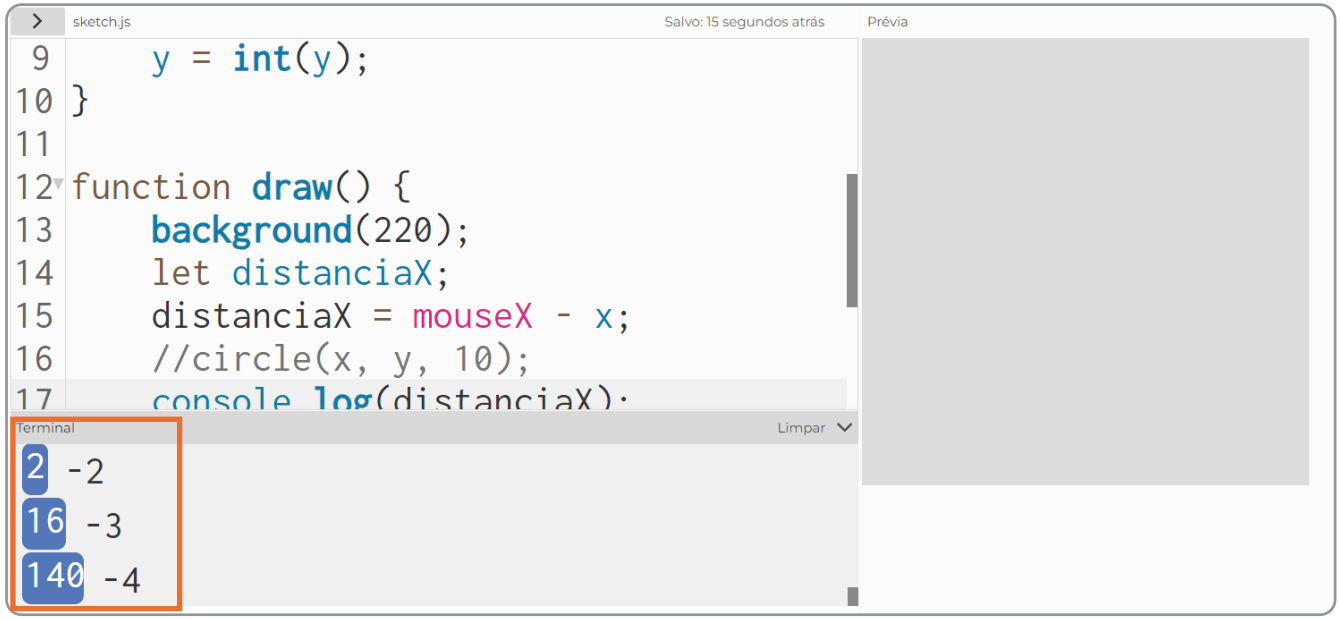
```
function draw() {
    background(220);
    //circle(x, y, 10);
    //console.log(mouseX,x);
```

Agora, observe que o círculo não aparece mais na tela. Assim, precisamos encontrar uma maneira de inserir dicas na tela que nos ajudem a encontrá-lo sem que ele esteja visível no canvas. Para isso, trabalharemos calculando distâncias entre o ponteiro do mouse e o centro do círculo nos eixos X e Y.

Começaremos criando uma variável chamada **distanciaX**, que será igual à posição X do ponteiro do mouse menos o valor da variável **x**. Podemos, inclusive, exibir esse valor no terminal para saber se estamos perto ou longe do ponto X do círculo. O código será o seguinte:

```
function draw() {
    background(220);
    let distanciaX;
    distanciaX = mouseX - x;
    //circle(x, y, 10);
    console.log(distanciaX);
```

Ao testar o projeto, perceba que há um valor sendo alterado no terminal. Esse valor representa a distância do ponteiro do mouse ao ponto que queremos encontrar no eixo X. Quanto menor o valor, significa que estamos mais próximos.

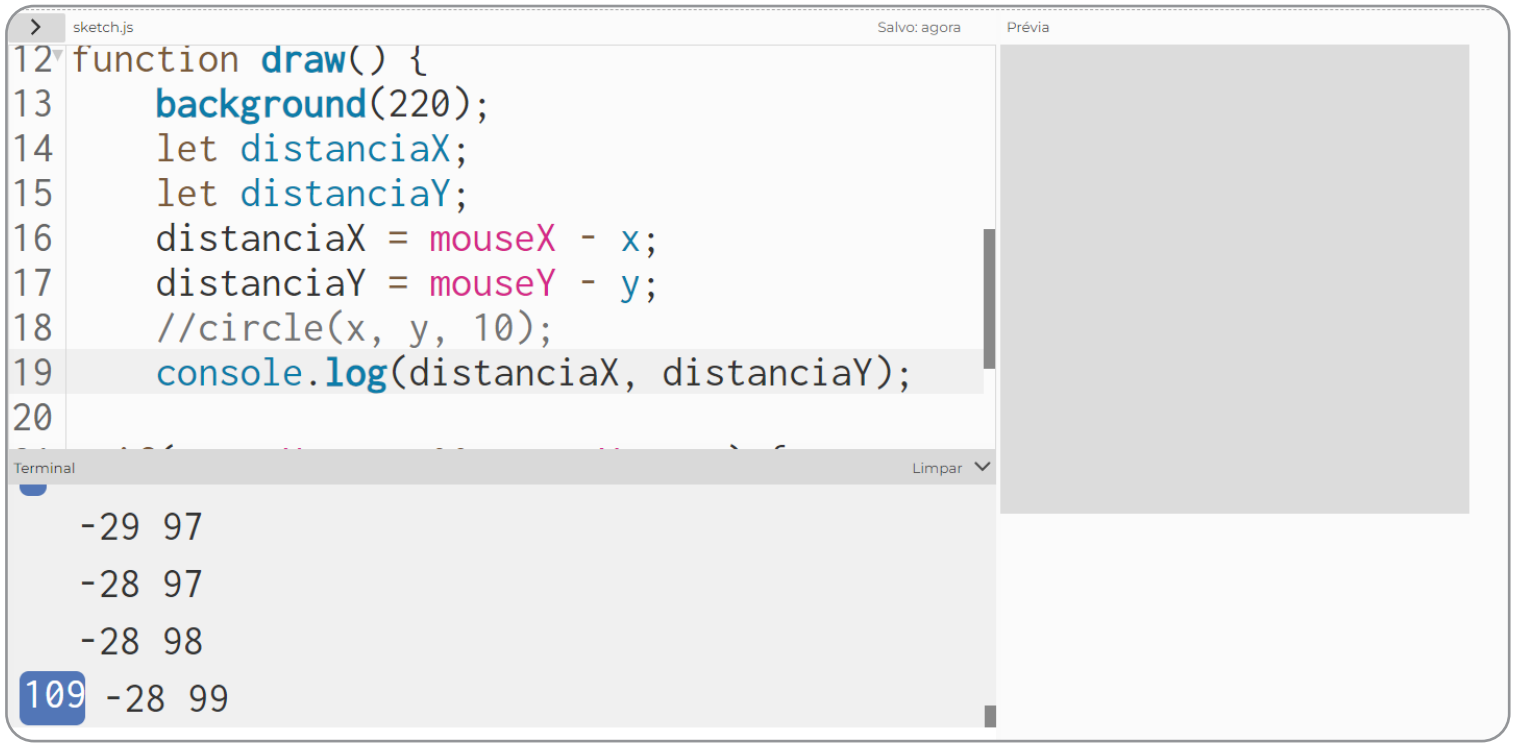


Agora, precisamos fazer o mesmo para o eixo Y!

Desse modo, criaremos uma variável **distanciaY** e atribuiremos a ela o valor do **mouseY** menos a variável **y**. Vamos também exibir essa variável no console da página. O código ficará assim:

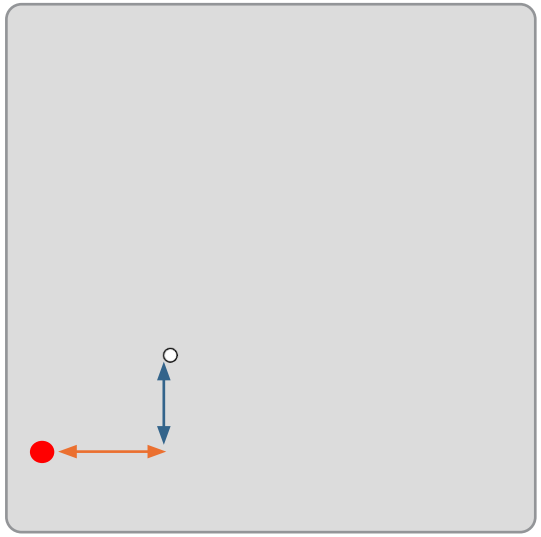
```
function draw() {
    background(220);
    let distanciaX;
    let distanciaY;
    distanciaX = mouseX - x;
    distanciaY = mouseY - y;
    //circle(x, y, 10);
    console.log(distanciaX, distanciaY);
```

Observe que o terminal agora exibe dois valores:



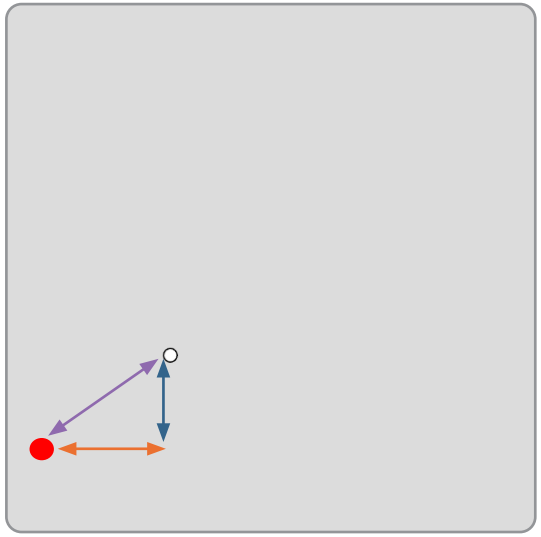
É por meio deles que conseguimos verificar se estamos perto ou longe do ponto que queremos encontrar!

Perceba que, em alguns momentos, a distância exibida é um valor negativo. Precisamos entender melhor esse conceito de distância e, para isso, faremos com que o círculo apareça novamente removendo o sinal `//` que colocamos antes da função **`circle`**. Com o círculo aparecendo novamente na tela, observe a imagem a seguir:



O círculo vermelho representa a posição do ponteiro do mouse, enquanto a seta laranja representa a distância do ponteiro do mouse até o círculo apenas no eixo X. Por sua vez, a seta azul representa a distância do ponteiro do mouse até o círculo apenas no eixo Y. Porém, qual seria o caminho mais curto entre esses dois pontos?

Se traçarmos uma terceira seta na diagonal, ligando os dois pontos, formaremos um triângulo retângulo. Esse é o caminho mais curto entre os dois pontos e, para encontrar essa distância, aplicaremos o Teorema de Pitágoras!



As variáveis **`distanciaX`** e **`distanciaY`** representam os catetos; precisamos apenas encontrar a hipotenusa. Para isso, programaremos a fórmula do Teorema de Pitágoras: a soma do quadrado dos catetos é igual ao quadrado da hipotenusa.

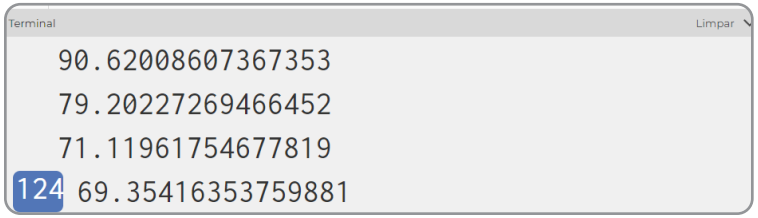
O código ficará da seguinte forma:

```
let distanciaX;
let distanciaY;
let distancia;
distanciaX = mouseX - x;
distanciaY = mouseY - y;
distancia = sqrt(distanciaX*distanciaX + distanciaY*distanciaY);

circle(x, y, 10);
console.log(distancia);
```

Perceba que criamos uma nova variável chamada **`distancia`** e aplicamos a ela a fórmula do Teorema de Pitágoras usando a função **`srqt`** para calcular a raiz quadrada. Além disso, na função **`console.log`**, estamos exibindo apenas essa nova distância.

Portanto, no terminal, será exibido apenas o valor da distância (hipotenusa) que encontramos:



Nesta aula, aplicamos nossos conhecimentos em trigonometria e programação para encontrar o caminho mais curto entre a posição do ponteiro do mouse e um ponto no canvas.

Teste o seu projeto e verifique se você consegue percorrer a distância da hipotenusa para chegar até o círculo e exibir a mensagem “Encontrei!”.

Se desejar aumentar o nível de dificuldade, oculte novamente o círculo utilizando os sinais **//**.

Bons estudos!

start by alura

Bons estudos!