start by alura

Aula 8

Quente frio dinâmico

▶ Unidade

Lógica de programação: criando arte interativa com p5.js

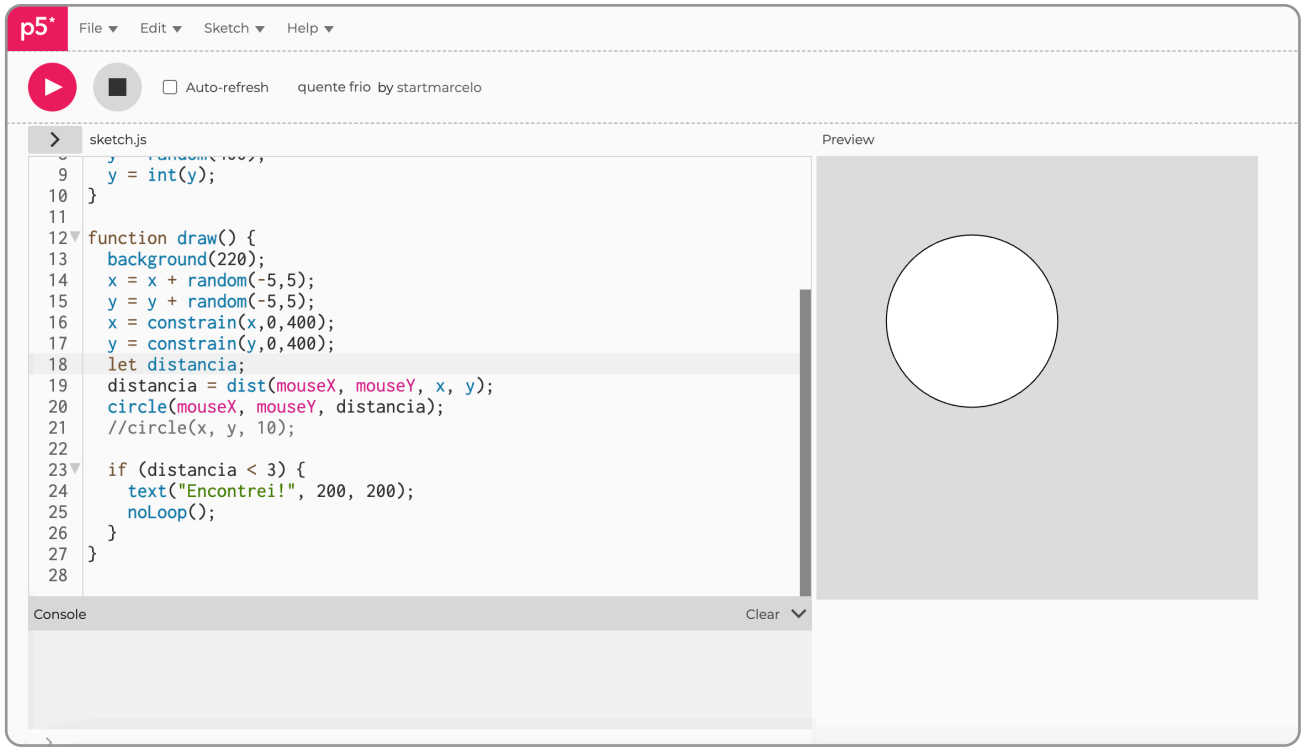
O que vamos aprender?

- Revisar o código para identificar e remover variáveis e comandos desnecessários.
- Aplicar o comando **`constrain`** para limitar o movimento de um objeto dentro de um plano cartesiano.
- Modificar a dificuldade de um projeto interativo através da modificação de parâmetros de distância.

ACESSE A PLATAFORMA START

Aumentando a dificuldade do jogo

Na aula passada, utilizamos as funções próprias do p5.js para realizar cálculos de distância e inserir elementos que deixaram nosso jogo ainda mais divertido. Nesta aula, vamos organizar o código e otimizar o jogo, tornando-o mais dinâmico.



Na aula passada, encontramos o nosso ponto e agora vocês podem analisar o código que já escreveram e refletir: será que existe algum trecho de código que não utilizamos mais?

Por exemplo, nas linhas 17 e 18 do meu código, temos o cálculo das variáveis **`distanciaX`** e **`distanciaY`**, que utilizamos para aplicar o teorema de Pitágoras, executando o cálculo na linha 19. Ao passarmos a usar a função **`dist()`**, deixamos de utilizar essas variáveis, portanto, podemos removê-las do código, assim como a execução.

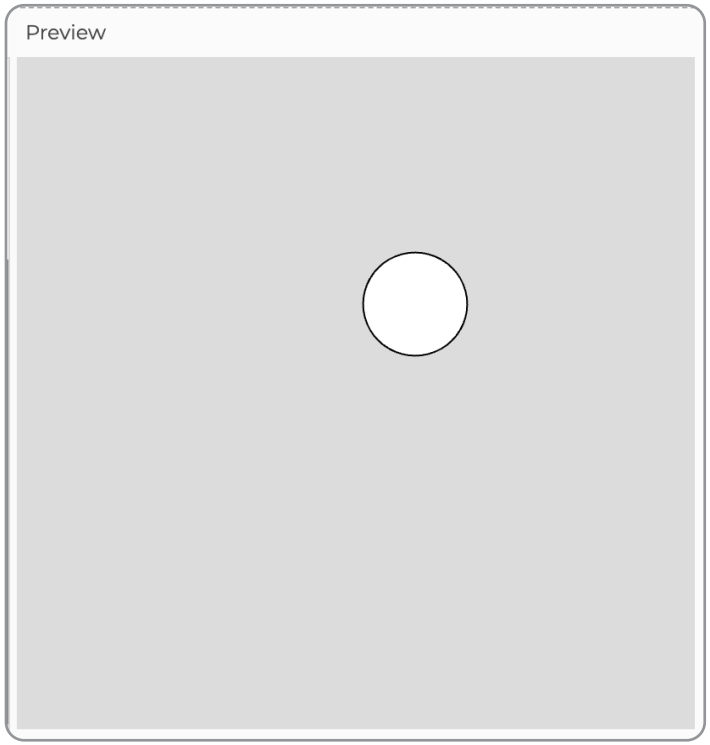
Já que apagaremos o cálculo, podemos também apagar as declarações dessas variáveis. No meu código, elas estão nas linhas 14 e 15, onde temos os comandos **`let distanciaX;`** e **`let distanciaY;`**. Além disso, podemos apagar o **`console.log`** para simplificar ainda mais o código.

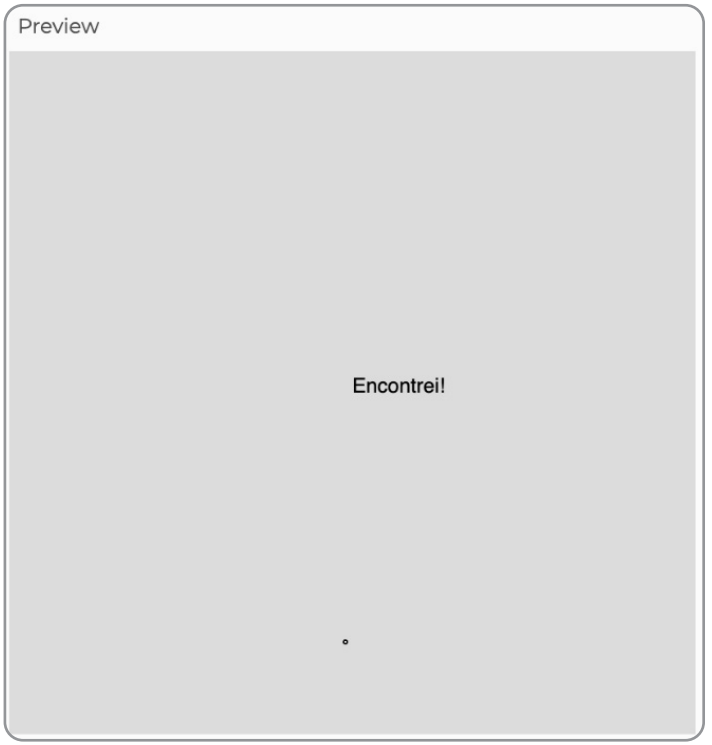
Após essas alterações, o código da função **`draw()`** ficou assim:

```
function draw() {
  background(220);
  let distancia;
  distancia = dist(mouseX, mouseY, x, y);
  circle(mouseX, mouseY, distancia);
  //circle(x, y, 10);

  if (distancia < 3) {
    text("Encontrei!", 200, 200);
    noLoop();
  }
}
```

O projeto segue funcionando por aqui. E aí, tudo certo também?





Você deve estar se perguntando por que eu não removemos a linha comentada `//circle(x, y, 10);`. Utilizaremos esse comando para nos ajudar nas próximas melhorias que faremos no jogo.

Imagine que temos esse círculo, que está oculto, mas que possui uma dinâmica em que ele se move pela tela de maneira aleatória. Para implementar essa dinâmica, precisaremos recalcular os valores de `X` e `Y` do ponto oculto.

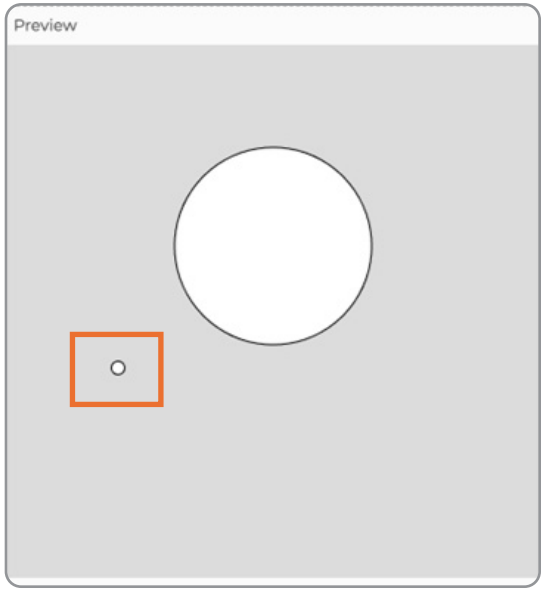
Consegue imaginar como podemos fazer isso? Dica: já utilizamos uma função que nos permite gerar números aleatórios nesse jogo.

Antes de posicionar o **`X`** no círculo, na linha 17, faremos o seguinte: o valor de **`X`** será ele mesmo mais **`random(-2,2)`**. Em seguida, faremos o mesmo cálculo para **`Y`**. Para visualizar se isso está funcionando, removeremos o comentário da linha 17 e chamaremos a função **`circle(x, y, 10);`**. Assim, podemos testar e ver como o resultado está ficando. Observe as alterações no código a seguir:

```
function draw() {
    background(220);
    x = x + random(-2,2);
    y = y + random(-2,2);
    let distancia;
    distancia = dist(mouseX, mouseY, x, y);
    circle(mouseX, mouseY, distancia);
    circle(x, y, 10);

    if (distancia < 3) {
      text("Encontrei!", 200, 200);
      noLoop();
    }
  }
```

Agora, nosso ponto oculto não está mais oculto, por enquanto, pois ele muda de posição na tela constantemente. A distância entre o mouse e o ponto, assim como a posição do ponto, é recalculada o tempo todo, porque implementamos isso dentro da função **`draw()`**. Ao testar, você perceberá que o ponto oculto desliza por várias áreas da tela. Observe a posição do ponto na imagem a seguir:

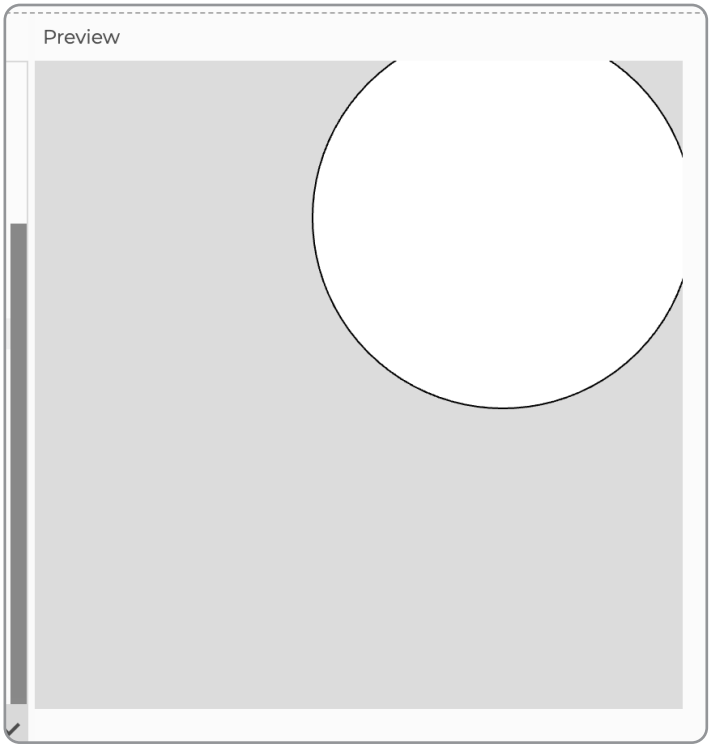


No entanto, a bolinha que está na parte inferior da tela pode se mover até sair dos limites. Isso ocorre porque, ao recalcular os valores de **`X`** e **`Y`**, não estamos garantindo que esses valores permaneçam dentro do intervalo entre 0 e 400.

Para entendermos melhor o que está acontecendo, alteraremos os valores **(-2,2)** por **(-20,20)**.

```
function draw() {
    background(220);
    x = x + random(-20,20);
    y = y + random(-20,20);
    let distancia;
    distancia = dist(mouseX, mouseY, x, y);
    circle(mouseX, mouseY, distancia);
    circle(x, y, 10);
  ...
  }
```

Você percebeu que a bolinha que representa o ponto oculto pode sair da tela? Observe a imagem a seguir:

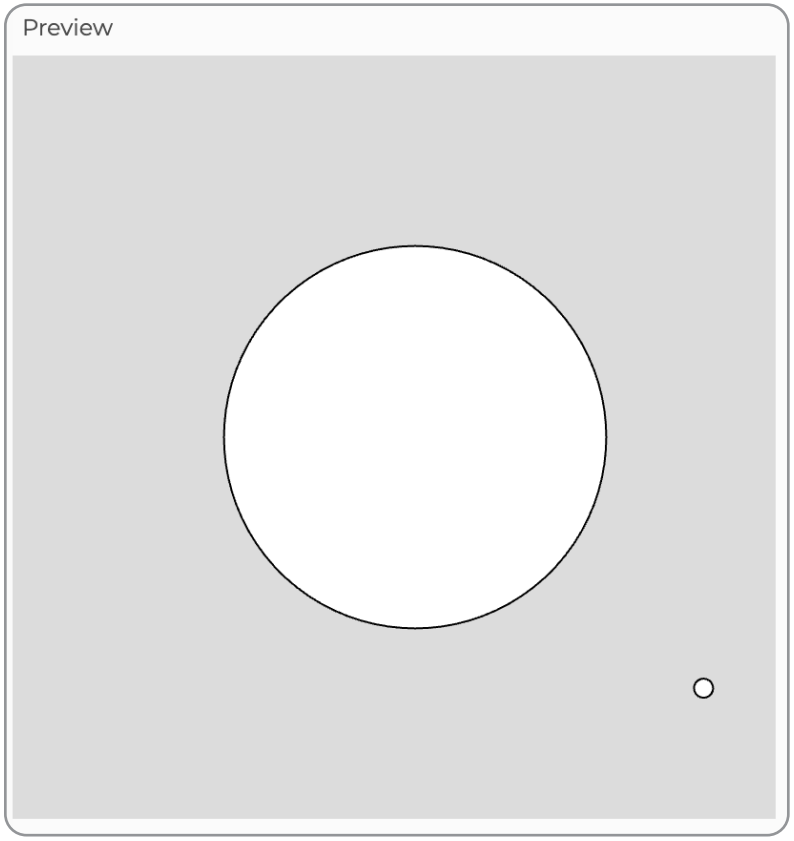


Desse jeito, vai ser mais difícil de encontrá-la. Então, o que precisamos fazer? Precisamos fixar os valores para a bolinha.

Na linha 16, por exemplo, podemos criar uma condição para garantir que, se **x** for maior que 400, o valor de **x** precise ser 400. Perceba que criamos nossa condição e trabalhamos com nosso operador lógico, maior ou menor, transformando esse valor em um valor máximo. Assim, se o número gerado pelo **random** for 420, por exemplo, ele será ajustado para 400. A mesma situação se aplica quando criamos um valor de **x** negativo. Então, se **x** for menor que 0, definiremos nosso padrão, que é **x = 0**. Dessa forma, o código ficará como o exemplo a seguir:

```
x = x + random(-20, 20);
y = y + random(-20, 20);
if (x > 400) {
  x = 400;
}
if (x < 0) {
  x = 0;
}
```

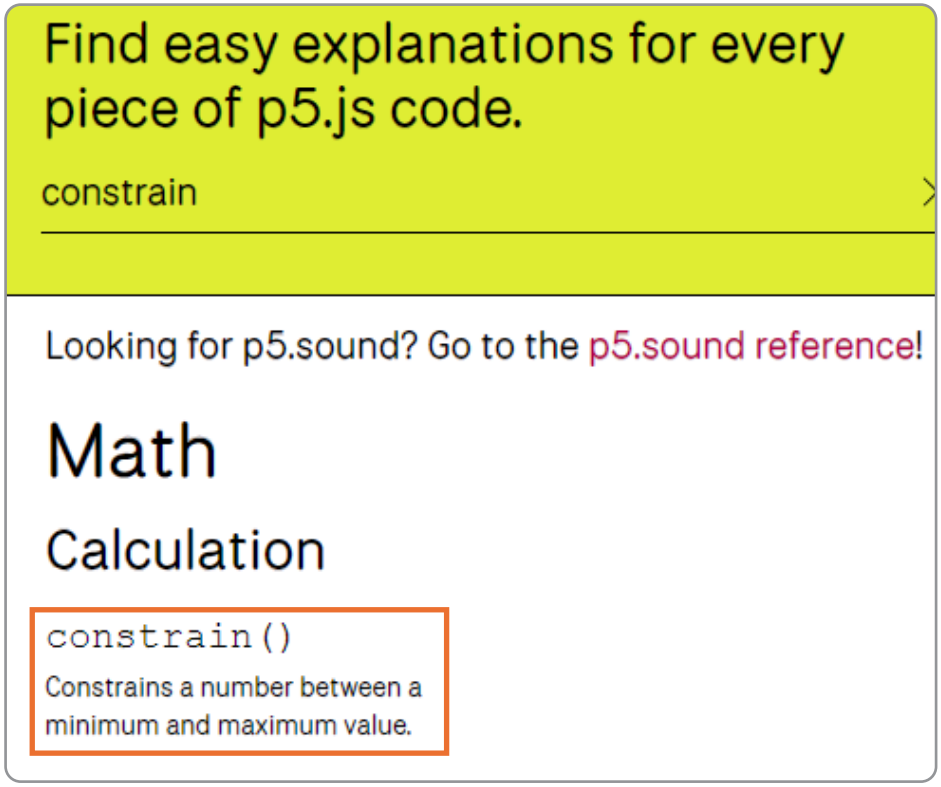
Quando executamos o nosso código, vemos que foi criado um limite que impede o valor de `X` de ser maior que o tamanho da tela. No entanto, o valor de `Y` ainda está saindo da tela, como podemos ver a seguir.



Estamos criando um `Y` maior que 400. Precisamos refazer a mesma estrutura de código que fizemos para `X`, mas agora para `Y` também. Mas você deve estar se perguntando: “não existe uma maneira mais fácil de fazer isso? Se colocarmos quatro `if`, teremos muito código. Será que ninguém resolveu esse problema, assim como existe um comando para calcular a dist?”.

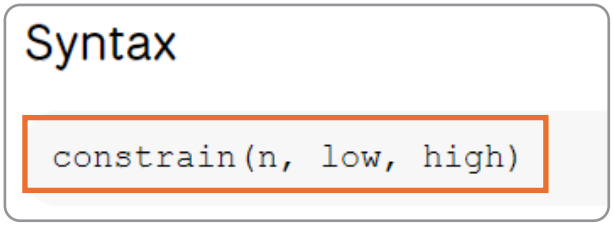
Sim, o próprio p5.js já tem esse comando pronto! Então, o que precisamos fazer? Vamos consultar a documentação.

Para isso, na barra de ferramentas do p5.js, clicaremos em Ajuda > Referência, como realizamos na aula passada, e no campo de pesquisa, digitaremos *constrain*. Como resultado, teremos a seguinte opção:



Ao acessarmos essa opção, encontraremos a seguinte explicação: “o *constrain* restringe um valor mínimo e um valor máximo”.

Além disso, encontraremos exemplos de usos. Em nosso caso, buscaremos pela opção *Syntax*. Para isso, rolaremos a barra de rolagem na lateral direita da tela até encontrar a opção desejada.



Essa sintaxe atribui ao comando a seguinte definição: **`n`** = número a ser restringido; **`low`** = limite mínimo; **`high`** = limite máximo. Com base nessa definição, copiaremos o código dessa sintaxe e o adicionaremos ao nosso código.

```
constrain (n, low, high)
```

Em seguida, usaremos o comando **constrain** para definir os valores que adicionaremos em **x** e **y**, utilizando 0 como valor mínimo e 400 como valor máximo. Voltando à aba do navegador com o nosso código, apagaremos as duas estruturas *if*, pois não precisamos mais delas.

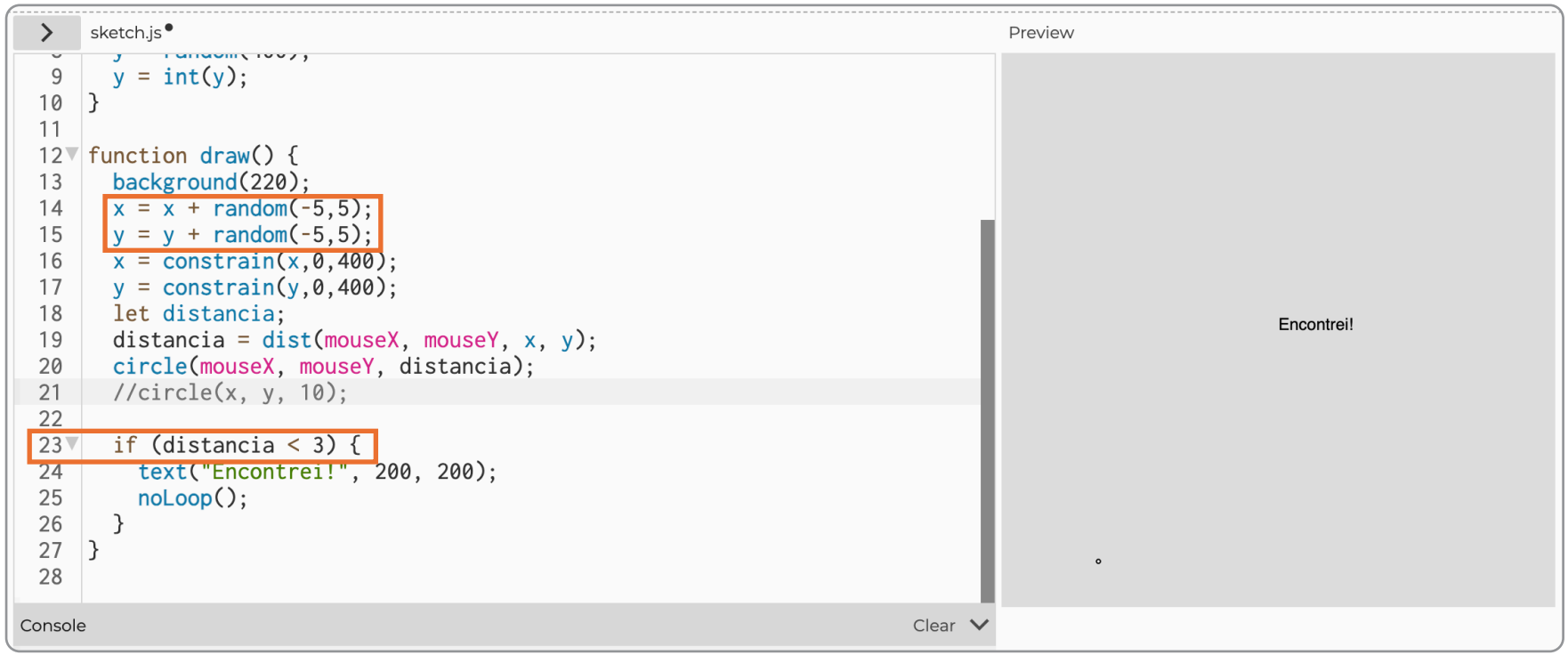
```
x = x + random(-20, 20);
y = y + random(-20, 20);
if (x > 400) {
  x = 400;
}
if (x < 0) {
  x = 0;
}
```

Depois disso, escreveremos o seguinte código:

```
function draw() {
    background(220);
    x = x + random(-20,20);
    y = y + random(-20,20);
    x = constrain(x,0,400);
    y = constrain(y,0,400);
...
}
```

Agora podemos executar o código e observar como o jogo está se comportando. Desta vez, o ponto oculto não vai mais sair da tela, que era exatamente o que queríamos, certo?

Percebam que, mesmo visível, ainda é difícil de alcançar o ponto oculto, pois a dificuldade está muito alta. Em vez de `(-20, 20)`, usaremos `(-5, 5)`, facilitando encontrar o ponto oculto. Agora, podemos ocultá-lo novamente, pois, com a dificuldade ajustada, confirmamos que o projeto está funcionando.




```
9   y = int(y);
10  }
11
12  function draw() {
13    background(220);
14    x = x + random(-5,5);
15    y = y + random(-5,5);
16    x = constrain(x,0,400);
17    y = constrain(y,0,400);
18    let distancia;
19    distancia = dist(mouseX, mouseY, x, y);
20    circle(mouseX, mouseY, distancia);
21    //circle(x, y, 10);
22
23    if (distancia < 3) {
24      text("Encontrei!", 200, 200);
25      noLoop();
26    }
27  }
28
```

Caso queira facilitar ainda mais a busca pelo ponto oculto, altere o valor de `if` na linha 23. Teste diferentes valores e encontre o que mais lhe agradar!

start by alura

Bons estudos!