start by alura

Aula 4

Efeito Monalisa

▶ Unidade

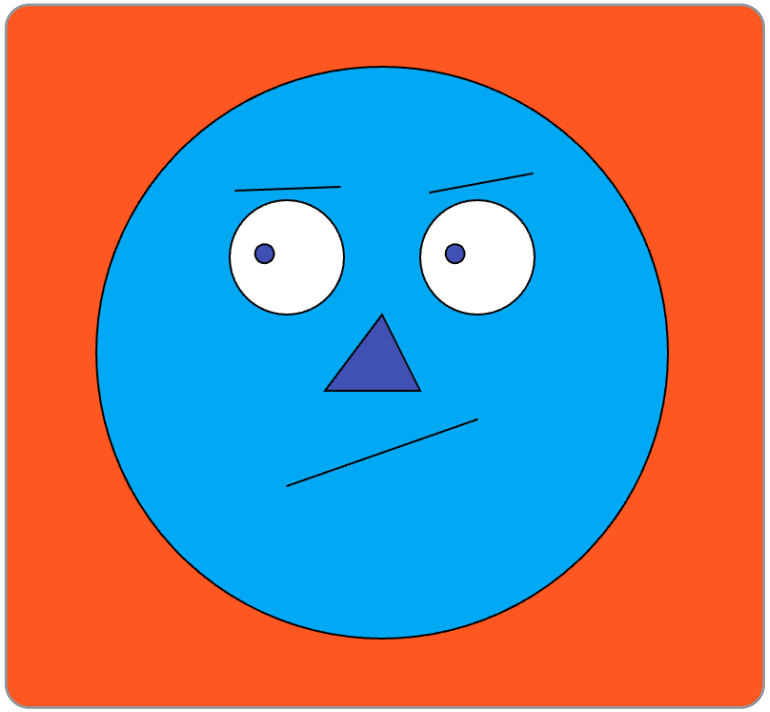
Lógica de programação: criando arte interativa com p5.js

O que vamos aprender?

- Aplicar conceitos de mapeamento para restringir a movimentação de elementos gráficos na tela.
- Analisar a lógica de posicionamento e ajuste de variáveis para criar interações visuais precisas.
- Criar uma animação interativa na qual elementos gráficos seguem a posição do cursor do mouse.

ACESSE A PLATAFORMA START

Como utilizamos o map()?



Na aula passada, adicionamos mais elementos e cores ao nosso projeto. Nesta aula, aprenderemos a adicionar alguns efeitos que deixarão nosso projeto interativo, tornando-o ainda mais divertido: programaremos nossa personagem para que ela siga o ponteiro do mouse com os olhos.

Até o momento, construímos o rosto da nossa personagem. Observem que poderíamos perfeitamente transpor essa imagem para um papel. Para isso, precisamos apenas usar tinta laranja no fundo, algum objeto para fazer círculos perfeitos, como um compasso, tintas azul-claro e azul--escuro para pintar o rosto e uma régua para fazer triângulos e linhas.

No entanto, seria muito interessante se pudéssemos criar algo que somente o computador é capaz de fazer. Assim, adicionaremos uma animação à personagem que criamos.

Imagine que as pupilas da personagem que criamos pudessem olhar para o cursor do mouse e segui-lo.

Ao criar essa animação, estamos fazendo uma analogia inspirada na obra Monalisa, que é uma das pinturas mais famosas do mundo.

Uma curiosidade dessa pintura é a ilusão de ótica que ela cria ao fazer com que a Monalisa pareça estar sempre olhando para nós, independente de nosso posicionamento diante do quadro.

Por exemplo, se observarmos a pintura do lado esquerdo ou do lado direito do quadro, temos a impressão de que ela está nos olhando. Essa é uma característica muito específica dessa pintura e que a torna cada vez mais misteriosa.

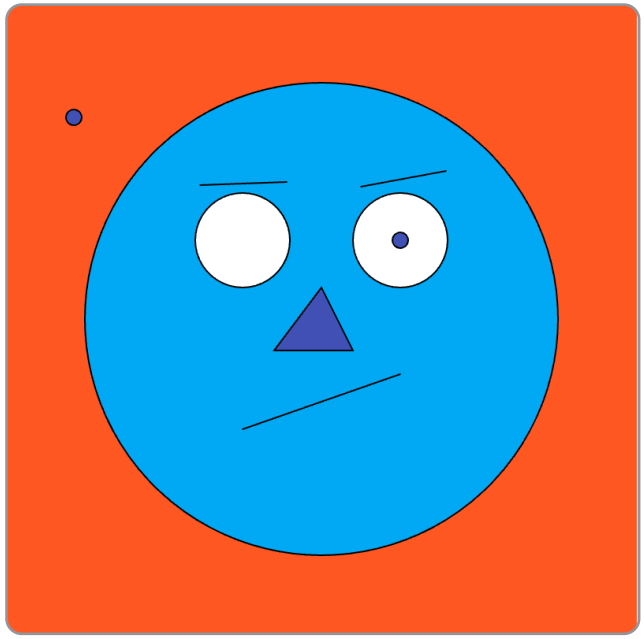
Para criarmos esse efeito, o primeiro passo será configurar a pupila da personagem de uma outra forma. Começaremos comentando a linha da pupila esquerda. Faremos isso abaixo de **`circle()`** da pupila direita.

Se a pupila deve seguir o cursor do mouse, adicionaremos um **`circle()`**, recebendo as coordenadas **`mouseX`** e **`mouseY`**, e manteremos um diâmetro de valor 10, conforme fizemos anteriormente. Observe:

```
function draw() {
  ...
  // circle(150, 150, 10); // pupila esquerda
  circle(250, 150, 10); // pupila direita
  circle(mouseX, mouseY, 10); // nova pupila esquerda

  if(mouseIsPressed){
    consele.log(mouseX,mouseY);
  }
}
```

Ao executar o código dessa forma, observe que a pupila sairá do olho, algo que não queremos. A ideia é que ela permaneça dentro do olho e apenas acompanhe o movimento do cursor.



Para conseguir esse efeito, precisamos trabalhar com alguns conceitos de programação, como as variáveis e a função **`.map()`**.

Logo abaixo da linha de código da pupila direita, informaremos ao programa que não podemos mais usar **mouseX** e **mouseY** como referência, pois não queremos que a pupila da personagem saia do lugar.

Assim, escreveremos **olhoX = map(mouseX, 0, 400);**. Isso significa que queremos guardar a informação resultante de uma fórmula matemática, a qual chamamos de variável (**olhoX**). Neste caso, estamos seguindo a ideia de que o cursor do mouse pode ir para qualquer lugar, mas a pupila precisa estar fixa dentro do olho.

```
function draw() {
  ...
  circle(250, 150, 10); // pupila direita
    olhoX = map(mouseX, 0, 400);
  circle(mouseX, mouseY, 10); // nova pupila esquerda
  if(mouseIsPressed){
    consele.log(mouseX,mouseY);
  }
}
```

Sendo assim, é necessário remapear o limite de valores nos quais o ponto da pupila pode se movimentar pela tela. Para isso, trabalharemos com a função **map()**.

O **`map()`** significa que precisamos redefinir alguma variável, limitando sua escala. Nesse caso, queremos limitar o **`mouseX`**, pois, no momento, estamos trabalhando com a variável **`olhoX`**.

A variável **`mouseX`** pode ir de 0, que é a extremidade esquerda, até 400, que corresponde à extremidade direita. Se adicionássemos um valor maior, como 500, por exemplo, a variável não funcionaria, pois o **`createCanvas()`** no início do código deixa nítido que as dimensões do canvas são **`400`** (em X) por **`400`** (em Y).

Se o canvas tiver, por exemplo, **`600`** no eixo X, podemos trabalhar com o limite de **`600`**.

Dito isso, vamos remapear o valor de **`mouseX`**, que pode ir de 0 até 400. Passaremos essa informação para o comando **`map()`**, atribuído à variável **`olhoX`**.

Porém, ainda precisamos remapear para que a pupila permaneça dentro do olho.

Nesse caso, qual limite de X ela pode atingir? Ela pode ir tanto para a esquerda, que começa no valor mínimo possível, quanto para o valor máximo à direita, sempre permanecendo dentro do olho.

Ao clicar nos limites desejados, recebemos no terminal os valores 132 e 166. Adicionaremos esses valores ao final da função **`map()`**, entre parênteses. Observe:

```
  olhoX = map(mouseX, 0, 400, 132, 166);
```

Ao executarmos o projeto, veremos que ele ainda está do mesmo jeito: a pupila segue presa ao ponteiro do mouse. Está tudo certo, mas ainda precisamos fazer mais algumas alterações para que tudo funcione corretamente.

Precisamos fazer o mesmo com o valor de **mouseY**. Dessa forma, logo abaixo de **olhoX**, vamos declarar **olhoY = map(mouseY, 0, 400, 130, 170);**, redefinindo **mouseY** de modo que a pupila permaneça dentro do olho direito.

Quanto ao valor máximo e mínimo, já identificamos as proporções de 400 por 400 no canvas, então, adicionamos as mesmas informações entre parênteses: **mouseY, 0, 400**.

Nesse caso, temos um novo parâmetro: a altura. O **0** (zero) corresponde à maior altura possível, enquanto o valor mais baixo possível é 400. Dessa forma, quanto maior a altura, menor o valor.

Assim, podemos trabalhar com os valores indicados no terminal após clicar sobre as posições desejadas no canvas: 133 na parte de cima do olho; e 169 na parte inferior. Podemos arrendondar os valores para 130 e 170. Observe:

```
  olhoX = map(mouseX, 0, 400, 130, 170);
  olhoY = map(mouseY, 0, 400, 130, 170);
```

Com isso, em vez de usar **mouseX** no último **circle()** que criamos, chamaremos a variável **olhoX**. Da mesma forma, em vez de **mouseY**, teremos a variável **olhoY**, pois já remapeamos esses valores.

```
olhoX = map(mouseX, 0, 400, 130, 170);
olhoY = map(mouseY, 0, 400, 130, 170);

circle(olhoX, olhoY, 10);
```

Para finalizar esse processo, vamos refazer o código da pupila para o olho direito. Na pupila esquerda, tínhamos os valores 150, 150, 10 para centralizá-la.

Na pupila direita, temos os valores 250, 150, 10. Portanto, os parâmetros de Y e do diâmetro serão os mesmos para as duas pupilas.

Sendo assim, podemos comentar o código antigo da pupila direita e criar um outro **`circle()`**, abaixo de **`circle(olhoX, olhoY, 10)`**, que receberá **`olhoY`**, tendo 10 como dois últimos valores. Observe:

```
  // circle(150, 150, 10); // pupila esquerda
  // circle(250, 150, 10); // pupila direita

  olhoX = map(mouseX, 0, 400, 130, 170);
  olhoY = map(mouseY, 0, 400, 130, 170);

  circle(olhoX, olhoY, 10); // nova pupila esquerda
  circle(olhoX + 100, olhoY, 10); //nova pupila direita

}
```

Agora, podemos executar o projeto e visualizar a pupila esquerda se movimentando dentro do limite do círculo que define o olho do desenho:

Para finalizar esta aula, precisamos reforçar um ponto muito importante. Quando executamos um código dessa forma, ele funciona normalmente. No entanto, ao criar uma variável, é extremamente importante seguir a boa prática de declará-la fora da função.

O que isso significa? Antes de executar o código, declaramos que determinada variável será utilizada em algum momento. Para isso, usamos o comando **`let olhoX;`** logo após a função **`setup()`**. Faremos o mesmo com a variável **`olhoY`** na linha abaixo.

```
let olhoX;
let olhoY;

function draw() {
  ...
}
```

Nosso projeto não terá alterações visíveis ao fazermos isso, mas, como dito, isso deixa o código adequado às boas práticas de programação, que funcionam como a norma culta de escrita de textos.

start by alura

Bons estudos!