DADOS DO RELATÓRIO

Matéria: Computação Gráfica **Código:** CMP1170

Relatório Atividade 13: Relatório sobre cores

Aluno: Vitor de Almeida Silva

Matrícula: 20161003305497

ENUNCIADO

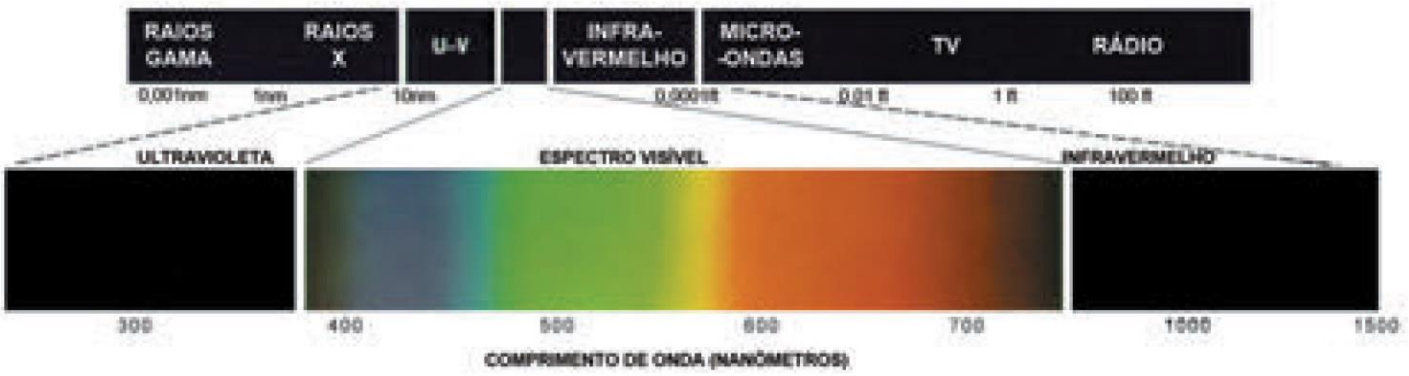
1) **Investigue quais são as primitivas do *Processing* utilizadas para configuração de cores. Crie no mínimo 6 objetos e utilize-os para mostrar as possibilidades de aplicação destas primitivas na definição de cores de objetos. Elabore um relatório e converta-o em PDF**.

DESENVOLVIMENTO

1 breve introdução sobre cores

As cores percebidas pelos humanos são determinadas pela natureza da luz refletida pelo objeto. A Luz visível é composta por uma banda de frequência estreita no espectro de energia eletromagnética, essa banda é mostrada na Figura 1.

Figura 1: banca de cor visível pelos humanos



Fonte: (GONZALEZ, WOODS, 2010)

A luz observada em aparelhos de televisão preto e branco é chamada de Luz acromática. Tendo em vista o espectro de luz visível, pode-se dizer que, os cones são os sensores dos olhos responsáveis pela visão das cores, sendo que, aproximadamente correspondem ao vermelho, ao verde e ao azul (RBG). Esses três componentes de cores, por sua vez, são as cores primárias, as outras cores são vistas como combinações dessas cores RGB (GONZALEZ, WOODS, 2010).

Dessa forma, as cores primárias podem ser combinadas para formarem as cores secundárias, as quais são:

- Magenta: vermelho mais azul;
- Ciano: verde mais azul;
- Amarelo: vermelho mais verde;

Por meio da combinação dessas cores, é possível produzir diversas outras cores no espectro de cores visível. A Figura 2, dá um exemplo da mistura dessas cores, para formar as cores branco e preto (GONZALEZ, WOODS, 2010).

Figura 2: Cores primárias e secundárias de luz e pigmentos

Fonte: (GONZALEZ, WOODS, 2010)

As cores quando representadas digitalmente, utilizam diversos modelos de cores, dentre esses modelos, são utilizados 2 com as primitivas do processing, sendo eles:

- **RGB:** modelo explicado anteriormente, consiste de componentes primárias sendo elas Vermelho, verde e azul. Cada componente, quando representado digitalmente, é normalizado entre um valor de intensidade que vai de 0 (preto) até 255 (branco). A combinação dessas cores forma diversas outras cores (GONZALEZ, WOODS, 2010);
- **HSI (Hue, Saturation, intensity (HSB)):** Esse modelo separa os componentes de cores em três tipos matiz, saturação e intensidade (*brigtiness*-brilho) (GONZALEZ, WOODS, 2010).

2 primitivas do *Processing* utilizadas para configuração de cores

Baseando-se nos conceitos de cores citados na seção anterior. O *processing* traz em suas bibliotecas, primitivas que podem ser utilizadas para atribuir e configurar cores a objetos e janelas. Tais primitivas, consistem das funções que serão citadas e testadas nessa seção.

Para os testes serão criados seis objetos distintos, onde serão aplicadas as funções de cores. o código do relatório comentado consta no Apêndice 1. As primitivas foram retiradas de Processing (2020).

2.1 colorMode()

Essa função, de forma resumida, muda o modo como o processing interpreta os dados de cores para RGB ou HSB. Como essa função somente muda o modo de cor, para testa-la, é preciso associar com outras funções que são capazes de atribuir estes modos de cores a objetos ou fundo. A sintaxe da função é a seguinte:

- ***colorMode(mode):*** *mode* pode ser RGB ou HSB;
- ***colorMode(mode, max):*** max é intervalo para todos os elementos de cores (0 a 255);
- ***colorMode(mode, max1, max2, max3):*** **max1** represente o primeiro componente de cor (red ou hue), max2 representa o segundo (Green ou *saturation*) e max3 o terceiro (bluo ou brightness);
- ***colorMode(mode, max1, max2, max3, maxA): maxA*** intervalo para um valor alpha.

Essa função será usada em exemplos mais adiante.

2.2 background()

Background se refere ao fundo da janela, essa função atribui a cor que será usada no fundo da janela. O background padrão é cinza. Essa função é tipicamente usada dentro da função ***draw()*** para limpar o display a cada mudança de quadro do objeto, porém, ela também pode ser usada na função ***setup()***.

Da mesma forma que as cores, uma imagem também pode ser usada como fundo da janela, porém, essa imagem deve ter as dimensões da janela. Assim como muitas funções de cores no processing, a função de background permite configurar dois tipos de parâmetros de cores, RGB e HSI. A sintaxe da função é mostrada a seguir:

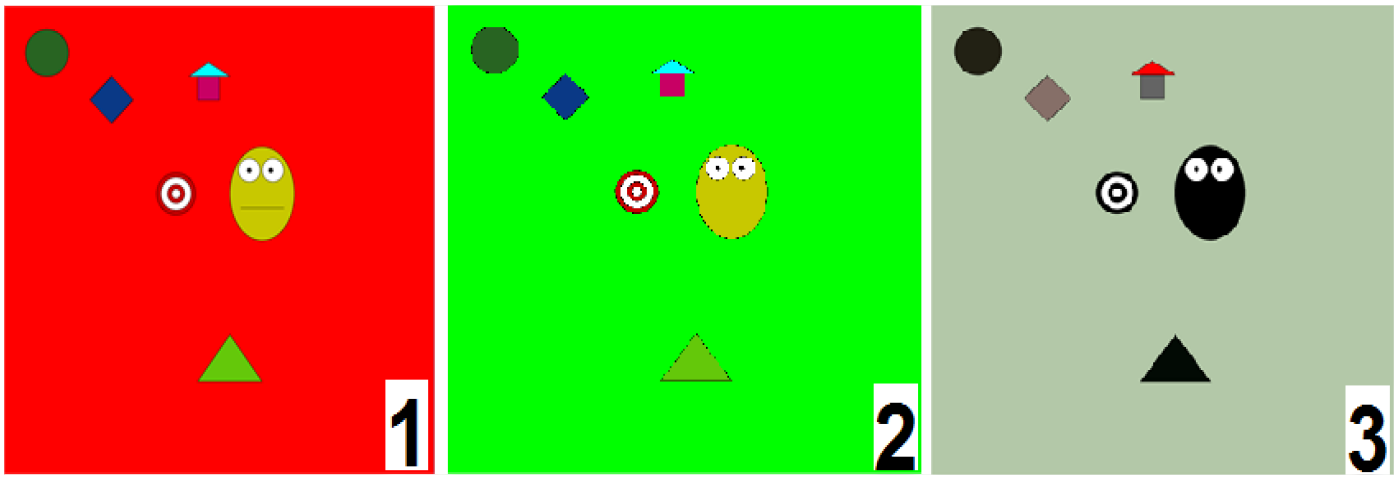
- **background(rgb);**
- **background(rgb, alpha);**
- **background(gray):** gray defino um tom de cinza entre 0 e 255;
- **background(gray, alpha);**
- **background(v1, v2, v3):** v1, v2 e v3 correspondem a R, G e B ou ao H, S e B;
- **background(v1, v2, v3, alpha);**
- **background(image):** image permite atribuir uma imagem ao fundo da janela;

Para testar a função background, foram selecionadas 3 variantes de cores para o fundo que são o seguinte:

1. modo de cor RGB: vermelho **background(255, 0, 0);**
2. modo de cor RGB: verde **background(0, 255, 0);**
3. modo de cor HSB: verde mais claro **background(70, 40, 200).**

A Figura 3 mostra dos testes para os tipos de cores supracitados.

Figura 1: teste modo 1, 2 e 3



Fonte: Autoral

2.3 fill() / noFill()

A função fill() é importante para aplicações em que se deseja colorir objetos, ela é capas de dar cor as formas definidas pelo desenvolvedor. Por exemplo, se for executado um fill(204, 102, 0), todos os formatos subsequentes irão ser preenchidos com laranja. Essa função também pode ser utilizada em dois modos de cores diferentes RGB e HSB.

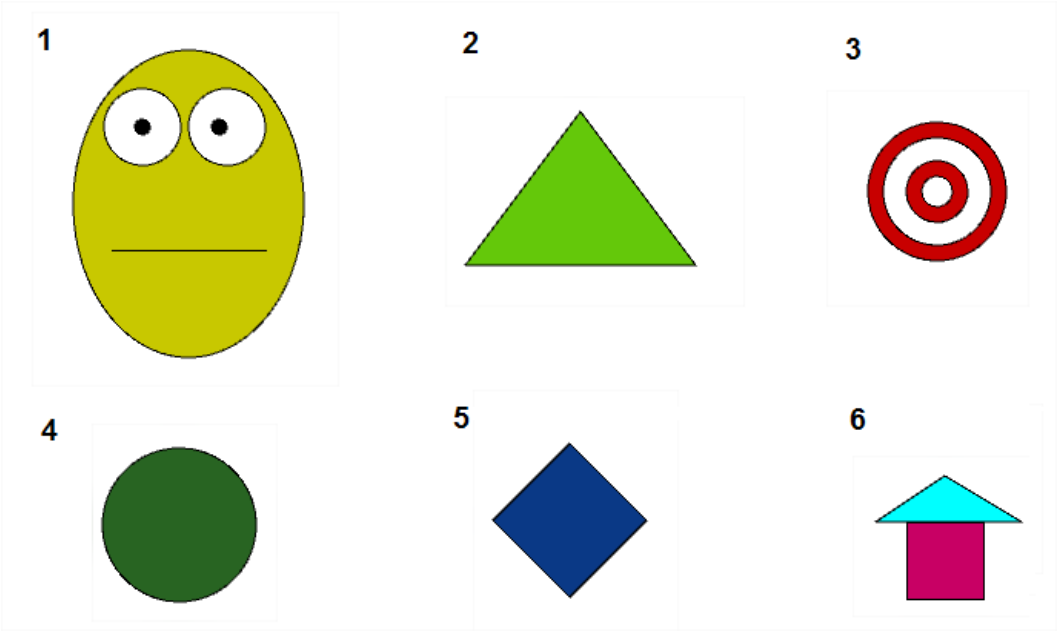
Por padrão o espaço de cor definido é RGB, o qual recebe valores inteiros de R, G e B que vão de 0 a 255. O valor para controlar os tons de cinza também vai de 0 a 255, bastando apenas utilizar somente o parâmetro de intensidade na função. A seguir são definidos a sintaxe da função e alguns exemplos de combinação de cores em RGB:

- **fill(G):** função para tons de cinza, G é a intensidade;
- **fill(R, G, B):** função para cores em RGB;
- **fill(H, S, B):** função para cores em HSB;

Com isso, a Figura 2 mostra o teste da função fill(R, G, B), para colorir os 6 objetos definidos na presente lista. Os testes foram feitos para os seguintes valores de fill():

1. **Boloniudo:** colorMode(RGB)
    a. **Pele:** fill(200,200, 0): um verde desbotado;
    b. **Parte branca dos olhos:** fill(255);
    c. **Iris:** fill(0): preto;
    d. **Boca:** stroke(0) (essa função será explicada mais adiante).
2. **Triangulo:** Fill(100, 200, 10) verde claro;
3. **Alvo:**
    a. **Circulo 1 e 3:** fill(255, 0, 0) vermelho;
    b. **Circulo 2 e 4:** fill(255) branco;
4. **Círculo:** fill(40, 100, 34) verde escuro;
5. **Losango:** fill(10, 57, 134) azul escuro;
6. **Casa:** colorMode(R,G,B)
    a. Telhado: fill(0, 255, 255) azul claro;
    b. Parede: fill(200, 0, 100).

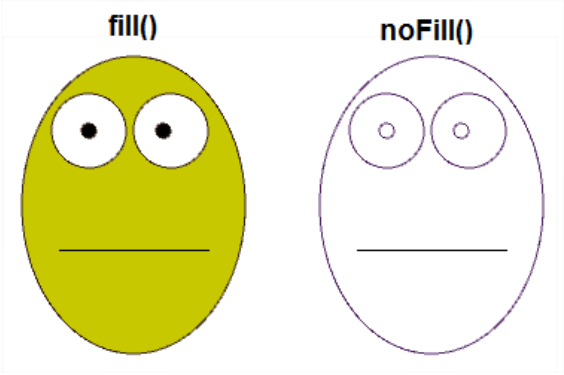
Figura 2: Triangulo (), boloniudo(), alvo(), circulo(), losango() e casa();



Fonte: Autoral

Em contrapartida a primitiva *fill(),* existe a primitiva *noFill(),* que deixa presente só os traços do desenho, esse comportamento pode ser visto na Figura 3. Cabe destacar que, o código desenvolvido para gerar e colorir os objetos está disponível no Apêndice 1 do relatório.

Figura 3: bolonildoNoFill()



Fonte: Autoral

2.4 stroke() / noStroke()

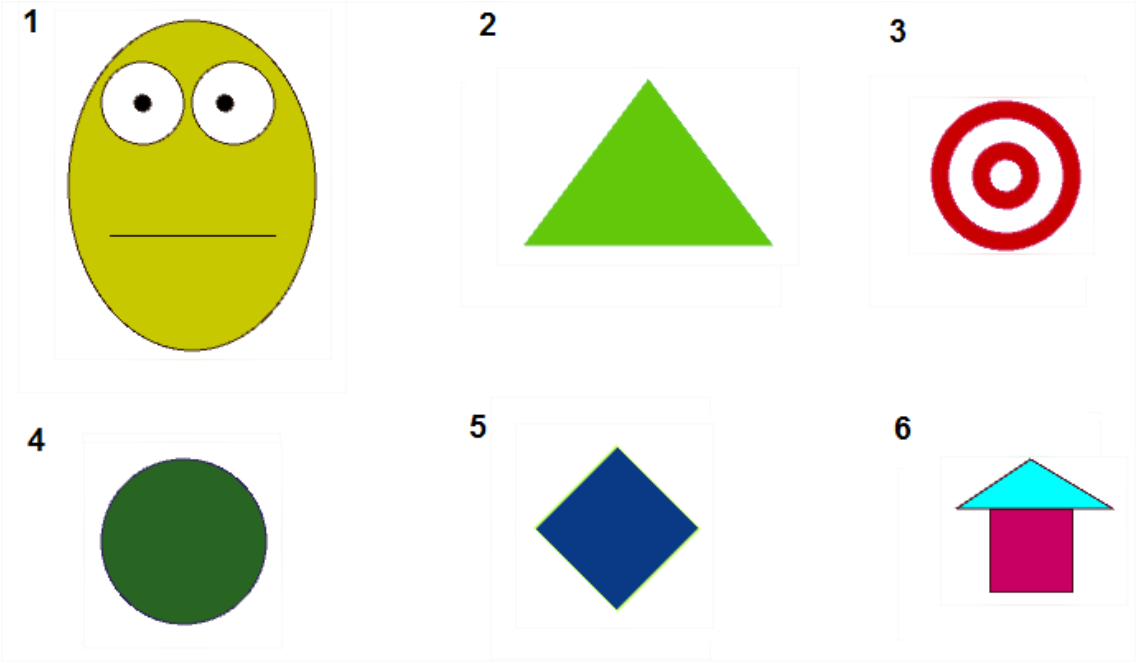
Diferente das funções anteriores que preenchiam fundo ou parte internas de formatos, a função *stroker()* tem a funcionalidade de desenhar traços no contorno dos formatos e de colorir esses traços em RGB ou HSB, por default os traços são pretos. A sintaxe da função pode ser listada da seguinte forma:

- ***stroke(RGB):*** valor de cor em notação hexadecimal;
- ***stroke(RGB, alpha):*** alpha é a opacidade do traço;
- ***stroke(gray):*** escala de cinza;
- ***stroke(gray, alpha):*** escala de cinza e opacidade do traço;
- ***stroke(v1, v2, v3)***: cores separas em RGB ou HSB;
- ***stroke(v1, v2, v3, alpha):*** cores separas e opacidade do traço;

Para ilustrar o exemplo de como funciona a função stroke(), cada objeto terá seu traço colorido de uma forma diferente, a Figura 4 mostra o resultado, as cores foram definidas como a seguir:

7. **Boloniudo:** stroke(56, 34, 24) rojo escuro;
8. **Triangulo:** stroke(100, 200, 50) verde claro;
9. **Alvo:** stroke(200, 34, 98) vermelho escuro;
10. **Circulo:** stroke(50, 34, 98) azul escuro;
11. **Losango:** stroke(200, 255, 98) verde;
12. **Casa:** stroke(80, 15, 30) rojo;

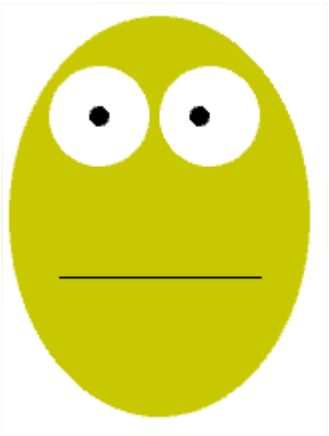
Figura 4: stroke()



Fonte: Autoral

Para deixar o desenho sem traços, basta utilizar a função ***noStroke(),*** está, por sua vez, irá deixar só o preenchimento interno do desenho. A Figura 5 mostra este comportamento.

Figura 5: bolonildo com noStroke()



Fonte: Autoral

2.5 color()

Em adicional, existe um tipo de objeto que pode ser criado que é da classe color(). Essa classe permite a manipulação de objetos que carregam atributos de cores em RGB ou HSB. Com o uso desse tipo de objeto é possível simplificar o código reduzindo algumas linhas e melhorando o entendimento do mesmo. A Figura 6 mostra duas maneiras de utilizar o objeto color.

Figura 6: objeto color em função

ex1:
```
color c1= color(200,200,200);
```

ex2:
```
void boloniudo(PVector centro, color c1, color c2, color c3, color c4)
boloniudo(new PVector(-100, 100), color(200, 200, 0), color(255), color(0), color(0));
```

Fonte: Autoral

REFERÊNCIAS

GONZALEZ, Rafael C., WOODS, Richard E. **Processamento Digtal de Imagem.** São Paulo: Sv, 2010.

PROCESSING. **Reference**. 2020. Disponível em: https://processing.org/reference/. Acesso em: 03 maio 2020.

APÊNDICE 1:

```
void setup()
{
size(1000, 1000, P2D);
colorMode(RGB);
//background(0, 255, 0);//modelo RGB
background(255);//modelo RGB

//alvo(R,G,B)
stroke(200,34,98); //traço do desenho
alvo(200, 0, 0);

//circulo(R, G, B)
stroke(50,34,98);
circulo(40, 100, 34);

//losango(R, G, B)
stroke(200,255,98);
Losango(10, 57, 134);

//boloniudo( centro, corPele, CorOlhoPbranca, corIris, corBoca);
stroke(56,34,24);
//noStroke();
boloniudo(new PVector(-100, 100), color(200, 200, 0), color(255), color(0), color(0));

// triangulo(R, G, B)
stroke(100,200,50);
triangulo(100, 200, 10);

//casa(centro, corTelhado, corParede);
stroke(80,34,98);
casa(new PVector(-200, 100), color(0, 255, 255), color(200, 0, 100));

//bolonildo sem fundo
boloniudoNoFill(new PVector(-300, 100));
}

//1) gera um objeto que é uma representação de um alvo
```

```
void alvo(int R, int G, int B)
{
fill(R, G, B);
circle(400, 400, 90);
fill(255);
circle(400, 400, 70);
fill(R, G, B);
circle(400, 400, 40);
fill(255);
circle(400, 400, 20);
}

//2) cria um circulo
void circulo(int R, int G, int B)
{
fill(R, G, B);
circle(100, 100, 100);
}

//3) cria um losangulo
void Losango(int R, int G, int B)
{
fill(R, G, B);
quad(200, 200, 250, 250, 300, 200, 250, 150);
}

//4) cria um objeto chamado Boloniudo,
//essa face pode ser colorida
//      *c1: pele;
//      *c2: olhos;
//      *c3: Cor do olho;
//      *c4: boca;
//PVector é um objeto que pode armazenar cordenas de um ponto
void boloniudo(PVector centro, color c1, color c2, color c3, color c4)
{
//rosto
fill(c1);
ellipse(500-centro.x, 500-centro.y, 150, 200);
```

```
//olhos
fill(c2);
circle(470-centro.x, 450-centro.y, 50); //circulo branco
fill(c3);
circle(470-centro.x, 450-centro.y, 10); //iris
fill(c2);
circle(525-centro.x, 450-centro.y, 50); //circulo
fill(c3);
circle(520-centro.x, 450-centro.y, 10); //iris
//boca
stroke(c4);
line(450-centro.x, 530-centro.y, 550-centro.x, 530-centro.y);
}

//5) define um triangulo
void triangulo(int R, int G, int B)
{
fill(R, G, B);
triangle(450, 800, 600, 800, 525, 700);
}

//6) define uma casa simples
void casa(PVector centro, color telhado, color parede)
{
//parede da casa
fill(parede);
square(250 - centro.x, 250 - centro.y, 50);
//telhado da casa
fill(telhado);
triangle(230 - centro.x, 250 - centro.y, 325 - centro.x, 250 - centro.y, 275 - centro.x, 220 - centro.y);
}


//bolonildo sem traço
void boloniudoNoFill(PVector centro)
{
noFill();
//rosto
```

```
ellipse(500-centro.x, 500-centro.y, 150, 200);
//olhos
circle(470-centro.x, 450-centro.y, 50); //circulo branco
circle(470-centro.x, 450-centro.y, 10); //iris
circle(525-centro.x, 450-centro.y, 50); //circulo
circle(520-centro.x, 450-centro.y, 10); //iris
//boca
stroke(0);
line(450-centro.x, 530-centro.y, 550-centro.x, 530-centro.y);
}
void draw()
{
}
```