

A large, stylized orange 'G1' logo. The 'G' is a thick, rounded shape, and the '1' is a simple vertical bar. The background is dark blue with faint binary code (0s and 1s) and glowing orange circuit lines.

24^{as} JORNADAS da COMPUTAÇÃO GRÁFICA e MULTIMÉDIA

Escola Superior de Tecnologia e Gestão | Instituto Politécnico de Viana do Castelo



23 e 24 de abril de 2026



ENGENHARIA DA COMPUTAÇÃO GRÁFICA E MULTIMÉDIA

Workshop em P5.js

Programação na Web com P5.js

Índice

1	Objetivo	1
2	Recursos Necessários	2
2.1	Hardware	2
2.2	Software	2
3	Workshop – Passo a Passo.....	3
3.1.	Configuração Inicial do Canvas.....	3
3.2.	Bola	4
3.2.1	Desenhar Bola.....	4
3.2.2	Mover Bola e Colisão com as Paredes	5
3.3.	Plataforma	7
3.3.1	Desenho e movimento da plataforma.....	7
3.3.2	Colisão da Bola com a Plataforma	10
3.4.	Tijolos	11
3.4.1	Desenhar os Tijolos e Tratar a Colisão com a Bola	11
3.4.2	Aplicar Imagem em Cima dos Tijolos	16
3.4.3	Mover Tijolos	22
3.5.	Vidas e Fim de Jogo	28
3.6.	Estado do Jogo e Ecrãs	34
4.	Conclusão.....	41
5.	Referências	42

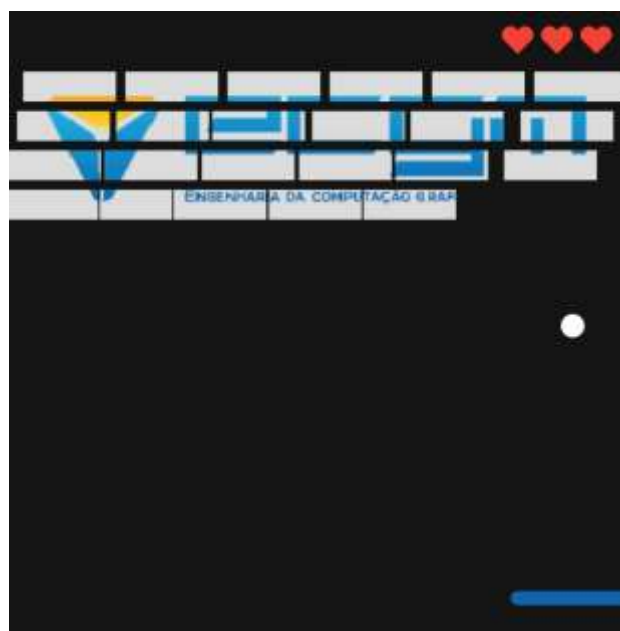
1 Objetivo

Este workshop tem como propósito introduzir-te ao mundo da programação criativa através da biblioteca p5.js. Através de uma metodologia prática de "aprender a fazer", serás guiado no desenvolvimento de um jogo clássico de destruição de tijolos (*Breakout*), que permite introduzir conceitos fundamentais de lógica de programação, computação gráfica e inteligência artificial.

Os principais objetivos a atingir são:

- **Compreender a estrutura p5.js:** Manipular o ciclo de vida de uma aplicação (setup() e draw()).
- **Desenvolver Mecânicas de Jogo:** Implementar vetores de movimento, deteção de colisões e gestão de variáveis globais (pontuação e vidas).
- **Introduzir Lógica de IA:** Criar comportamentos autónomos em elementos do jogo para aumentar o desafio e a interatividade.
- **Gestão de Estados:** Estruturar o código para alternar entre ecrãs de menu, jogabilidade, vitória e derrota.

No final do workshop, terás desenvolvido um jogo divertido que te permitirá ganhar autonomia para criar as tuas próprias experiências interativas na web.



2 Recursos Necessários

Para a realização deste Workshop são necessários os seguintes recursos:

2.1 Hardware

- **Computador:** Um computador (Portátil ou Desktop) com desempenho suficiente para correr um navegador web moderno.
- **Periféricos:** Rato (recomendado para facilitar a navegação no código) e teclado.
- **Acesso à Internet:** Ligação estável à Internet para aceder ao editor online do p5.js e carregar as bibliotecas e recursos necessários.

2.2 Software

- **Navegador Web (Browser):** Recomenda-se a utilização de uma versão atualizada do Google Chrome, Mozilla Firefox ou Microsoft Edge.
- **Editor de Código:** Utilizaremos o **p5.js Web Editor**, que permite programar diretamente no navegador sem necessidade de instalação local. Disponível em: <https://editor.p5js.org/>
- **Imagens:** Imagens para os fundos e sprites (disponibilizadas durante o workshop).
- **Conta p5.js:** É necessário que os participantes criem uma conta gratuita no site oficial do **p5.js Web Editor** para poderem guardar e partilhar os seus projetos.

3 Workshop – Passo a Passo

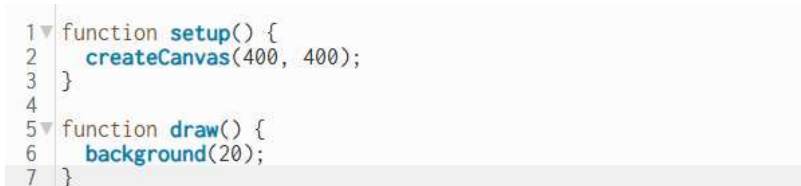
3.1. Configuração Inicial do Canvas

O primeiro passo deste projeto é criar a Estrutura Base do Programa em p5.js, constituído pelos métodos **setup()** e **draw()**.

O método **setup()** é executado apenas uma vez no início do programa. Cria este método e introduz a a função **createCanvas(400, 400)**, que cria uma área de desenho com 400 pixels de largura e 400 pixels de altura.

O método **draw()** é executado continuamente (aproximadamente 60 vezes por segundo) e é responsável por atualizar o conteúdo do ecrã. Dentro deste método, é chamada a função **background(20)**, que define a cor de fundo num tom preto/cinza escuro.

```
function setup() {  
  createCanvas(400, 400);  
}  
  
function draw() {  
  background(20);  
}
```



```
1 function setup() {  
2   createCanvas(400, 400);  
3 }  
4  
5 function draw() {  
6   background(20);  
7 }
```

Resultado: Ecrã preto/cinza escuro de dimensão 400 x 400 pixels.

3.2. Bola

3.2.1 Desenhar Bola

Antes do método `setup()`, define as variáveis da bola, que permitem determinar a sua posição nos eixos (**`let xPosBola = 200, yPosBola = 200`**), a sua velocidade (**`let xVel = 4, yVel = -4`**) e o seu tamanho (**`let BolaSize = 16`**):

```
let xPosBola = 200, yPosBola = 200;  
  
let xVel = 4, yVel = -4;  
  
let tamBola = 16;
```

Seguidamente, em baixo do método `draw()`, cria o método **`desenharBola()`**. Este método é responsável por desenhar a bola no ecrã. No seu interior, chama a função **`fill(255)`**, que permite preencher a bola com a cor branca, e **`circle(xPosBola, yPosBola, tamBola)`**, que desenha a bola com a posição e o tamanho definidos pelas variáveis anteriormente declaradas:

```
function desenharBola() {  
  
    fill(255);  
  
    circle(xPosBola, yPosBola, tamBola);  
  
}
```

Por fim, para conseguires ver a bola desenhada no ecrã, só tens de chamar a função `desenharBola()` dentro do método `draw()`.

```
desenharBola();
```

A este ponto, o teu código deve estar assim:

```

1  //---- Variáveis da bola ----
2  let xPosBola = 200, yPosBola = 200;
3  let xVel = 4, yVel = -4;
4  let tamBola = 16;
5
6  function setup() {
7    //---- Cria a área de jogo e as suas dimensões ----
8    createCanvas(400, 400);
9  }
10
11 function draw() {
12   background(20);
13
14   desenharBola();
15 }
16 //---- Desenha a bola ----
17 function desenharBola() {
18   fill(255);
19   circle(xPosBola, yPosBola, tamBola);
20 }

```

3.2.2 Mover Bola e Colisão com as Paredes

Após a bola estar desenhada, cria a função **moverBola()**. Esta função soma os valores da velocidade xVel e yVel às posições xPosBola e yPosBola respetivamente, de modo a que a bola altere a sua posição nos eixos cada vez que a função é executada:

```

function moverBola() {

  xPosBola += xVel;

  yPosBola += yVel;

}

```

O próximo passo é colocar dentro do método **moverBola()**, a condição que permite à bola rebater nas paredes laterais e na parte superior do canvas.

- Se **xPosBola < 0** ou **xPosBola > width**, significa que a bola ultrapassou, respetivamente, a lateral esquerda ou direita do canvas. Nesse caso, o valor de **xVel** é multiplicado por **-1**, o que inverte a direção da bola no eixo x.
- Se **yPosBola < 0**, significa que a bola ultrapassou a parte superior do canvas. Nesse caso, o valor de **yVel** é multiplicado por **-1** o que inverte a direção da bola no eixo y.

```
if (xPosBola < 0 || xPosBola > width) xVel *= -1;

if (yPosBola < 0) yVel *= -1;
```

O teu moverBola() deve estar assim:

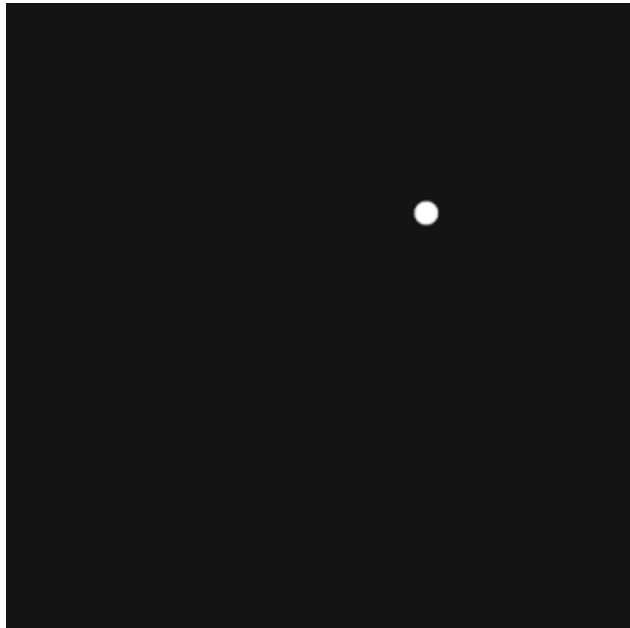
```
16 //---- Desenha a bola ----
17 function desenharBola() {
18     fill(255);
19     circle(xPosBola, yPosBola, tamBola);
20 }
21
22 //---- Atualiza a posição da bola e trata a colisão com as paredes
    e plataforma ----
23 function moverBola() {
24     // ---- Atualiza a posição ----
25     xPosBola += xVel;
26     yPosBola += yVel;
27
28     //---- Faz a bola rebater nas paredes ----
29     if (xPosBola < 0 || xPosBola > width) xVel *= -1;
30     if (yPosBola < 0) yVel *= -1;
31 }
32
```

Para que a função seja executada continuamente e consigas ver a bola a mover, chama moverBola(), antes de desenharBola(), no método draw().

```
moverBola();
```

```
10  
11 function draw() {  
12   background(20);  
13  
14   moverBola();  
15   desenharBola();  
16 }  
17
```

Resultado: Bola a mover-se no ecrã, rebate nas paredes superior e laterais.



3.3. Plataforma

3.3.1 Desenho e movimento da plataforma

No início do código, após a declaração das variáveis da bola, define as variáveis da plataforma, que permitem determinar a sua altura e largura (**let hBarra = 80, let wBarra = 10**) e a sua posição nos eixos (**let xPosBarra, yPosBarra**):

```
let wBarra = 80;

let hBarra = 10;

let xPosBarra, yPosBarra;
```

```
1 //---- Variáveis da bola ----
2 let xPosBola = 200, yPosBola = 200;
3 let xVel = 4, yVel = -4;
4 let tamBola = 16;
5
6 //---- Variáveis da plataforma ----
7 let wBarra = 80;
8 let hBarra = 10;
9 let xPosBarra, yPosBarra;
```

O próximo passo é criar a função `desenharPlataforma()`. A posição horizontal da plataforma é definida pela posição `mouseX` do jogador através de `mouseX - wBarra / 2` (subtrai-se metade da largura da plataforma para que o rato fique no centro da mesma). A função **`constrain(mouseX - wBarra / 2, 0, width - wBarra)`** garante que estes fiquem entre 0 e `width - wBarra`, para que não ultrapassem os limites laterais do canvas.

Já a posição vertical da plataforma é constante e definida por **`yPosBarra = height - 30`** (altura do canvas menos 30 pixels).

Após serem calculadas as posições nos eixos, **`rect(xPosBarra, yPosBarra, wBarra, hBarra, 5)`** permite desenhar a plataforma no ecrã, cuja cor é cinzento claro, definida por **`fill(15, 100, 173)`**, com largura e altura determinadas por `wBarra` e `hBarra` e raio igual a 5 nos cantos arredondados.

```
function desenharPlataforma() {
  xPosBarra = constrain(mouseX - wBarra / 2, 0, width - wBarra);
  yPosBarra = height - 30;
  fill(15, 100, 173);
  rect(xPosBarra, yPosBarra, wBarra, hBarra, 5);
}
```

```

29
30 //---- Atualiza a posição da bola e trata a colisão com as paredes e plataforma
31 function moverBola() {
32   // ---- Atualiza a posição ----
33   xPosBola += xVel;
34   yPosBola += yVel;
35
36   //---- Faz a bola rebater nas paredes ----
37   if (xPosBola < 0 || xPosBola > width) xVel *= -1;
38   if (yPosBola < 0) yVel *= -1;
39 }
40
41 //---- Desenha a plataforma de acordo com o movimento do rato do jogador ----
42 function desenharPlataforma() {
43
44   //----Calculo da posição no eixo x da plataforma, para que o mouse fique no
45   //centro da plataforma---
46   xPosBarra = constrain(mouseX - wBarra / 2, 0, width - wBarra);
47
48   //---- Posição no eixo y da plataforma ----
49   yPosBarra = height - 30;
50   fill(15, 100, 173);
51   rect(xPosBarra, yPosBarra, wBarra, hBarra, 5);
52 }

```

Para que consigas ver a plataforma no teu canvas, chama `desenharPlataforma()` na função `draw()`.

`desenharPlataforma();`

```

15
16 function draw() {
17   background(20);
18
19   desenharPlataforma();
20
21   moverBola();
22
23   desenharBola();
24 }
25

```

3.3.2 Colisão da Bola com a Plataforma

Coloca a condição abaixo dentro do método **moverBola()**. Esta condição permite que a bola ressalte na plataforma. Se **yPosBola > yPosBarra**, a bola encontra-se abaixo do topo da plataforma. Se **xPosBola > xPosBarra** e **xPosBola < xPosBarra + wBarra**, a bola encontra-se dentro dos limites horizontais da plataforma.

Quando estas três condições se verificam, o valor de **yVel** é multiplicado por -1, o que inverte a direção da bola no eixo y, e **yPosBola** recebe o valor **yPosBarra - 5**, para garantir que a bola fica imediatamente acima da plataforma após a colisão.

```
if (yPosBola > yPosBarra && xPosBola > xPosBarra && xPosBola < xPosBarra +
wBarra) {

    yVel *= -1;

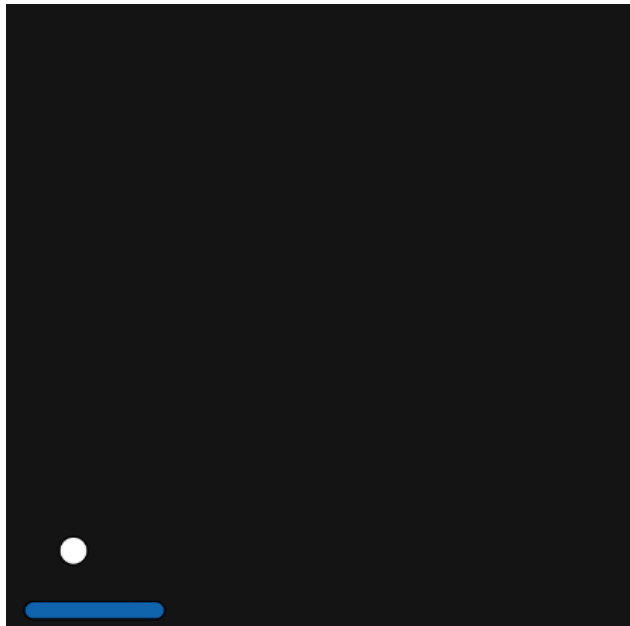
    yPosBola = yPosBarra - 5;

}
```

Este trecho de código deve ficar aqui:

```
27 function moverBola() {
28     // ---- Atualiza a posição ----
29     xPosBola += xVel;
30     yPosBola += yVel;
31
32     //---- Faz a bola rebater nas paredes ----
33     if (xPosBola < 0 || xPosBola > width) xVel *= -1;
34     if (yPosBola < 0) yVel *= -1;
35
36     //---- Faz a bola rebater na plataforma ----
37     if (yPosBola > yPosBarra && xPosBola > xPosBarra && xPosBola < xPosBarra +
+ wBarra) {
38         yVel *= -1;
39         yPosBola = yPosBarra - 5;
40     }
41 }
```

Resultado: Plataforma desenhada na parte inferior do ecrã que se move consoante o movimento do rato do jogador e faz a bola ressaltar.



3.4. Tijolos

3.4.1 Desenhar os Tijolos e Tratar a Colisão com a Bola

Após a declaração das variáveis da plataforma, acrescenta o array tijolos, que armazena todos os tijolos do jogo.

```
let tijolos = [];
```

```
5  
6 //---- Variáveis da plataforma ----  
7 let wBarra = 80;  
8 let hBarra = 10;  
9 let xPosBarra, yPosBarra;  
10  
11 //---- Array de tijolos ----  
12 let tijolos = [];  
13
```

Seguidamente, cria os tijolos do jogo com dois loops for dentro do setup(). O loop externo percorre as linhas (r) e o interno percorre as colunas (c). Para cada posição, cria-se um objeto t, que representa um tijolo, com a posição x e y espaçadas em 65 pixels horizontalmente e 25 pixels verticalmente dos restantes objetos, largura w de 60 pixels, altura h de 20 e estado ativo **active = true**. Este objeto é adicionado ao array tijolos através de **tijolos.push(t)**.

```
for (let r = 0; r < 4; r++) {
  for (let c = 0; c < 6; c++) {
    let t = {
      x: c * 65 + 10,
      y: r * 25 + 40,
      w: 60,
      h: 20,
      active: true
    };
    tijolos.push(t);
  }
}
```

```
15
16 function setup() {
17   //---- Cria a área de jogo e define as suas dimensões ----
18   createCanvas(400, 400);
19
20   // ---- Cria os tijolos ----
21   for (let r = 0; r < 4; r++) {
22     for (let c = 0; c < 6; c++) {
23       let t = {
24         x: c * 65 + 10,
25         y: r * 25 + 40,
26         w: 60,
27         h: 20,
28         active: true
29       };
30       tijolos.push(t);
31     }
32   }
33 }
34
```

Cria a função **desenharTijolos()**, responsável por desenhar apenas tijolos ativos no canvas, ou seja, aqueles que não colidiram com a bola. Esta função percorre o array tijolos e, para cada tijolo, verifica se ele está ativo. Se estiver, desenha um retângulo com a cor laranja, utilizando **fill(220)**, na posição definida por t.x e t.y, e com largura e altura definidas por t.w e t.h.

```
function desenharTijolos() {
    fill(220);
    for (let t of tijolos) {
        if (t.active) {
            rect(t.x, t.y, t.w, t.h);
        }
    }
}
```

```
85
86 //---- Desenha a plataforma de acordo com o movimento do rato do jogador ----
87 function desenharPlataforma() {
88
89     //----Calculo da posição no eixo x da plataforma, para que o rato fique no centro da
    plataforma----
90     xPosBarra = constrain(mouseX - wBarra / 2, 0, width - wBarra);
91
92     //---- Posição no eixo y da plataforma ----
93     yPosBarra = height - 30;
94     fill(15, 100, 173);
95     rect(xPosBarra, yPosBarra, wBarra, hBarra, 5);
96 }
97
98
99 //---- Dezenha os tijolos ----
100 function desenharTijolos() {
101     fill(220);
102     for (let t of tijolos) {
103         if (t.active) {
104             rect(t.x, t.y, t.w, t.h);
105         }
106     }
107 }
108
```

Chama **desenharTijolos()** dentro da função draw(), em baixo de moverBola().

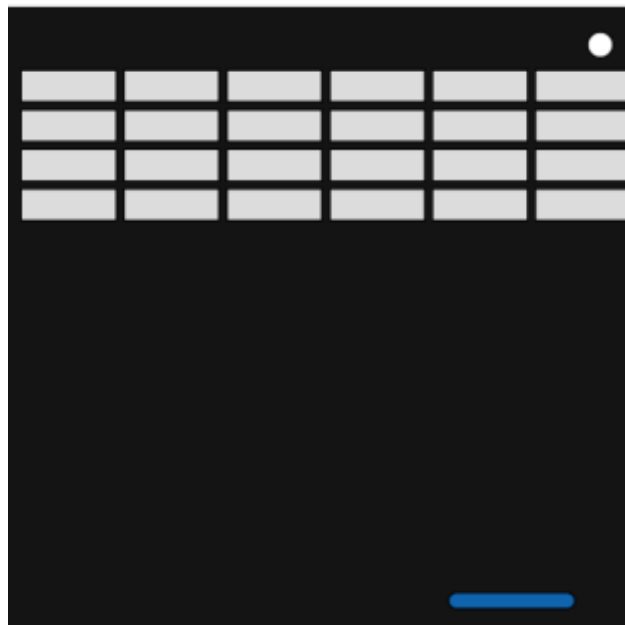
```
desenharTijolos()
```

```

50
51 function draw() {
52   background(20);
53
54   desenharaPlataforma();
55
56   moverBola();
57
58   desenharaTijolos();
59   desenharaBola();
60 }
61

```

Agora já deves poder ver os tijolos desenhados na tela, assim:



Dentro do loop, acrescenta a deteção de colisão entre a bola e o tijolo. A condição verifica se a área da bola se sobrepõe à área do tijolo, através da comparação dos limites esquerdo, direito, superior e inferior de ambos. Se houver sobreposição, ocorre uma colisão: desativa o tijolo (**t.active = false**) e inverte a velocidade vertical da bola (**yVel *= -1**), o que faz com que ela mude de direção.

```

if (
  xPosBola + tamBola / 2 > t.x &&
  xPosBola - tamBola / 2 < t.x + t.w &&
  yPosBola + tamBola / 2 > t.y &&
  yPosBola - tamBola / 2 < t.y + t.h
) {
  t.active = false;
  yVel *= -1;
}

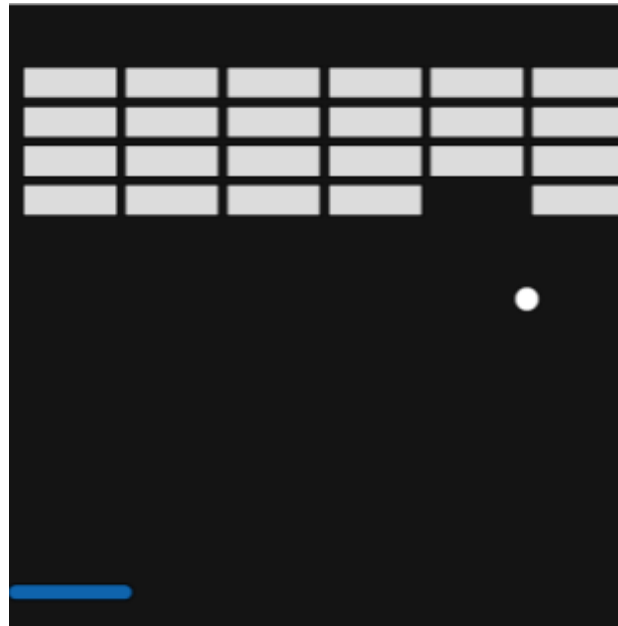
```

```

244
245 //---- Desenha os tijolos e trata colisão com a bola ----
246 function desenharTijolos() {
247   fill(220);
248
249   for (let t of tijolos) {
250     if (t.active) {
251       rect(t.x, t.y, t.w, t.h);
252
253       if (
254         xPosBola + tamBola / 2 > t.x &&
255         xPosBola - tamBola / 2 < t.x + t.w &&
256         yPosBola + tamBola / 2 > t.y &&
257         yPosBola - tamBola / 2 < t.y + t.h
258       ) {
259         t.active = false;
260         yVel *= -1;
261       }
262     }
263   }
264 }
265

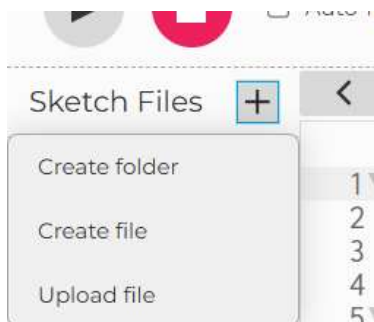
```

Resultado: 4 linhas e 6 colunas de tijolos dispostos na parte superior do canvas. Quando a bola colide com o tijolo, este desaparece. A este ponto, o teu jogo deve estar assim:



3.4.2 Aplicar Imagem em Cima dos Tijolos

Para completares o próximos passos terás que criar uma pasta no p5.js Web Editor. Clica no + que esta no lado superior esquerdo do ecrã, selecciona **Create Folder** e atribui o nome **img**.



Agora, dentro da pasta que criaste, clica em Carregar arquivo e seleciona as imagens logo.png, localizadas na pasta de recursos disponibilizada pelo workshop.



Seguidamente, declara as variáveis que vão guardar as imagens:

```
let imgEscola;
```

```
let imgCoracao;
```

```
17 //---- Estado do jogo (inicio, aJogar, fimJogo) ----
18 let estado = "inicio";
19
20 //---- Mensagem que é exibida quando o jogo termina ----
21 let msg;
22
23 //---- Imagens ----
24 let imgEscola;
25 let imgCoracao;
26
```

Antes do método draw(), cria uma função **preload()**. Esta função é responsável por carregar recursos (como imagens, sons ou ficheiros) antes da execução das funções setup() e draw().

```
function preload() {

  imgEscola = loadImage('/img/logo.png');

  imgCoracao = loadImage('/img/coracao.png');

}
```

```
24 let imgEscola;
25 let imgCoracao;
26
27 function preload() {
28   imgEscola = loadImage('/img/logo.png');
29   imgCoracao = loadImage('/img/coracao.png');
30 }
```

No início do método `desenharTijolos()`, remove `fill(220)` e define os parâmetros da grelha de tijolos através do número de colunas (6) e linhas (4). Com base nestes valores, é calculada largura total da grelha (**larguraGrade**) e a altura total (**alturaGrade**), tendo em conta o espaço ocupado por cada tijolo. Estes valores permitem determinar as dimensões da “parede” de tijolos no ecrã.

De seguida, são calculados os deslocamentos **deslocamentoX** e **deslocamentoY**, que servem para centrar corretamente a imagem de fundo (`imgEscola`) dentro da grelha, de forma a garantir que esta fique alinhada e visualmente equilibrada em relação ao conjunto de tijolos.

```
let colunas = 6;

let linhas = 4;

let larguraGrade = colunas * 65 - 5;

let alturaGrade = linhas * 25;

let deslocamentoX = (larguraGrade - imgEscola.width) / 2;

let deslocamentoY = (alturaGrade - imgEscola.height) / 2;
```

```
175
176 //---- Desenha os tijolos e trata colisão com a bola
177 function desenhartijolos() {
178     let colunas = 6;
179     let linhas = 4;
180
181     let larguraGrade = colunas * 65 - 5;
182     let alturaGrade = linhas * 25;
183
184     let deslocamentoX = (larguraGrade - imgEscola.width) / 2;
185     let deslocamentoY = (alturaGrade - imgEscola.height) / 2;
186
187     for (let t of tijolos) {
188         if (t.active) {
189             rect(t.x, t.y, t.w, t.h);
190
```

Na mesma função, dentro do ciclo for, define o aspeto visual de cada tijolo. Primeiro, remove o contorno com **noStroke()** e aplica uma cor cinzenta com **fill(220)**. Em seguida, desenha um retângulo nas coordenadas **t.x** e **t.y**, com largura **t.w** e altura **t.h**, que serve como base do tijolo.

Depois, desenha a imagem com a função **image()**, utilizando as mesmas coordenadas e dimensões do retângulo. Os valores **(t.x - 10) - deslocamentoX** e **(t.y - 40) - deslocamentoY** ajustam a área da imagem a recortar, o que garante que cada tijolo mostra a parte correta da imagem, e que esta fica alinhada com a grelha.

Por fim, o código volta a ativar o contorno com **stroke(20)** e desenha outro retângulo por cima do anterior, sem preenchimento (**noFill()**), para destacar visualmente os limites de cada tijolo.

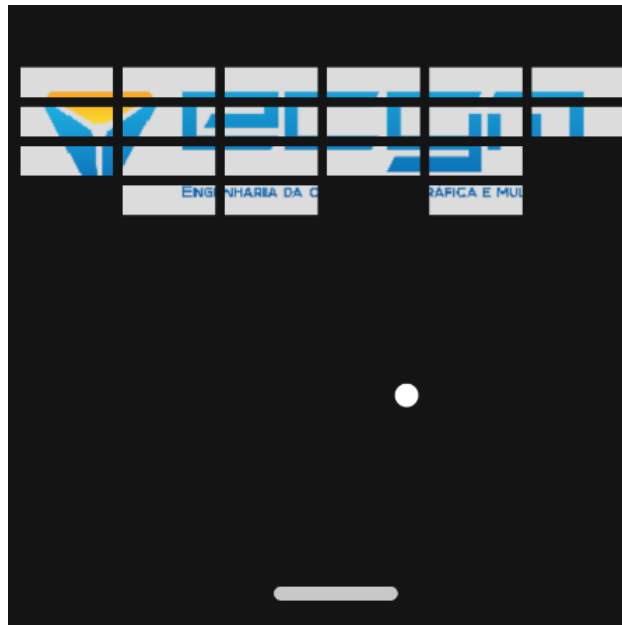
```
noStroke();  
  
fill(220);  
  
rect(t.x, t.y, t.w, t.h);  
  
  
image(  
    imgEscola,  
    t.x, t.y, t.w, t.h,  
    (t.x - 10) - deslocamentoX,  
    (t.y - 40) - deslocamentoY,  
    t.w, t.h  
);  
  
  
noFill();  
  
stroke(20);  
  
rect(t.x, t.y, t.w, t.h);
```

```

113
114▼ for (let t of tijolos) {
115▼   if (t.active) {
116     rect(t.x, t.y, t.w, t.h);
117
118     if (
119       xPosBola + tamBola / 2 > t.x &&
120       xPosBola - tamBola / 2 < t.x + t.w &&
121       yPosBola + tamBola / 2 > t.y &&
122       yPosBola - tamBola / 2 < t.y + t.h
123▼   ) {
124     t.active = false;
125     yVel *= -1;
126   }
127
128   noStroke();
129   fill(220);
130   rect(t.x, t.y, t.w, t.h);
131
132   image(
133     imgEscola,
134     t.x, t.y, t.w, t.h,
135     (t.x - 10) - deslocamentoX,
136     (t.y - 40) - deslocamentoY,
137     t.w, t.h
138   );
139
140   noFill();
141   stroke(20);
142   rect(t.x, t.y, t.w, t.h);
143 }
144 }
145 }

```

Resultado: Logo ECGM em cima do tijolo.



3.4.3 Mover Tijolos

Dentro da função **setup()**, define os valores **speed** e **reaction** dos tijolos. O **speed** controla a velocidade com que os tijolos se deslocam horizontalmente, enquanto o **reaction** define a probabilidade de o tijolo reagir à posição da bola e mudar de direção. A variável **vx** representa a velocidade horizontal inicial, que começa em zero.

```
speed: random(0.2, 0.6),
```

```
reaction: random(0.3, 0.7),
```

```
vx:0,
```

```
14 function setup() {
15   //---- Cria a área de jogo e as suas dimensões ----
16   createCanvas(400, 400);
17
18   for (let r = 0; r < 4; r++) {
19     for (let c = 0; c < 6; c++) {
20       let t = {
21         x: c * 65 + 10,
22         y: r * 25 + 40,
23         w: 60,
24         h: 20,
25         active: true,
26         speed: random(0.2, 0.6),
27         reaction: random(0.3, 0.7),
28         vx: 0,
29       };
30     };
31     tijolos.push(t);
32   }
33 }
34 }
```

Cria agora uma função chamada **moverTijolosComIA()**. A função cria um ciclo que começa por verificar se o tijolo está ativo. Se não estiver ativo, o código ignora esse tijolo e passa ao seguinte. Depois é calculada **distY**, que representa a distância vertical e permite perceber o quão perto a bola está em relação ao tijolo no eixo y.

Seguidamente, verifica se essa distância é menor que 70 e se a bola está a descer (**yVel > 0**). Quando estas condições são verdadeiras, a função calcula a direção em que o tijolo deve reagir, através da comparação entre a posição da bola e o centro do tijolo.

Por fim, aplica um fator de aleatoriedade com **t.reaction**, que determina se o tijolo reage naquele momento. Se reagir, aumenta a velocidade horizontal do tijolo (**t.vx**) na direção calculada, o que faz com que o tijolo se mova para tentar acompanhar a bola.

```
function moverTijolosComIA() {  
  for (let i = 0; i < tijolos.length; i++) {  
    let t = tijolos[i];  
    if (!t.active) continue;  
  
    let distY = abs(yPosBola - t.y);  
  
    if (distY < 70 && yVel > 0) {  
      let direction = (xPosBola < t.x + t.w / 2) ? 1 : -1;  
  
      if (random() < t.reaction) {  
        t.vx += direction * t.speed;  
      }  
    }  
  
    t.x += t.vx;  
    t.vx *= 0.95;  
  
  }  
}
```

```

356
357 //---- Movimenta os tijolos com IA ----
358 function moverTijolosComIA() {
359   for (let i = 0; i < tijolos.length; i++) {
360     let t = tijolos[i];
361     if (!t.active) continue;
362
363     // ---- IA: tijolo move-se horizontalmente para tentar alinhar com a bola ----
364     let distY = abs(yPosBola - t.y);
365
366     if (distY < 70 && yVel > 0) {
367       let direction = (xPosBola < t.x + t.w / 2) ? 1 : -1;
368
369       if (random() < t.reaction) {
370         t.vx += direction * t.speed;
371       }
372     }
373
374     // ---- Atualiza posição horizontal ----
375     t.x += t.vx;
376     t.vx *= 0.95;
377
378   }
379 }
380

```

Implementa a deteção de colisão dos tijolos com as paredes laterais do ecrã. Se a posição horizontal do tijolo for menor ou igual a 0, significa que atingiu a parede esquerda. Nesse caso, define a posição como 0 e inverte a velocidade horizontal (**t.vx *= -1**).

Se a posição **t.x + t.w** for maior ou igual à largura do ecrã (width), significa que o tijolo atingiu a parede direita. Nesse caso, define a posição como **width - t.w** e volta a inverter a velocidade horizontal, de modo a que o tijolo mude de direção.

```

if (t.x <= 0) {
    t.x = 0;
    t.vx *= -1;
}

if (t.x + t.w >= width) {
    t.x = width - t.w;
    t.vx *= -1;
}

```

```

356
357 //---- Movimenta os tijolos com IA e trata a colisão com as paredes ----
358 function moverTijolosComIA() {
359   for (let i = 0; i < tijolos.length; i++) {
360     let t = tijolos[i];
361     if (!t.active) continue;
362
363     // ---- IA: tijolo move-se horizontalmente para tentar alinhar com a bola ----
364     let distY = abs(yPosBola - t.y);
365
366     if (distY < 70 && yVel > 0) {
367       let direction = (xPosBola < t.x + t.w / 2) ? 1 : -1;
368
369       if (random() < t.reaction) {
370         t.vx += direction * t.speed;
371       }
372     }
373
374     // ---- Atualiza posição horizontal ----
375     t.x += t.vx;
376     t.vx *= 0.95;
377
378     // ---- Colisão com paredes ----
379     if (t.x <= 0) {
380       t.x = 0;
381       t.vx *= -1;
382     }
383     if (t.x + t.w >= width) {
384       t.x = width - t.w;
385       t.vx *= -1;
386     }
387   }
388 }
389

```

Para evitar a sobreposição entre os tijolos, cria um segundo ciclo dentro do primeiro. Nesse ciclo, percorre os restantes tijolos a partir de $j = i + 1$ e guarda cada um na variável **o**. De seguida, verifica se o tijolo **o** está ativo; caso não esteja, ignora-o e passa ao próximo (**if (!o.active)**).

Depois, compara as posições e dimensões dos tijolos **t** e **o** para verificar se existe sobreposição entre ambos.

Se ocorrer colisão, define um valor **push = 0.5** e verifica qual dos tijolos está mais à esquerda. Com base nessa posição, ajusta as coordenadas **t.x** e **o.x** em sentidos opostos, de forma a afastar os dois tijolos.

Por fim, inverte e reduz a velocidade horizontal de ambos (**t.vx** e **o.vx**) ao multiplicar por **-0.5**, simulando o efeito de impacto entre eles.

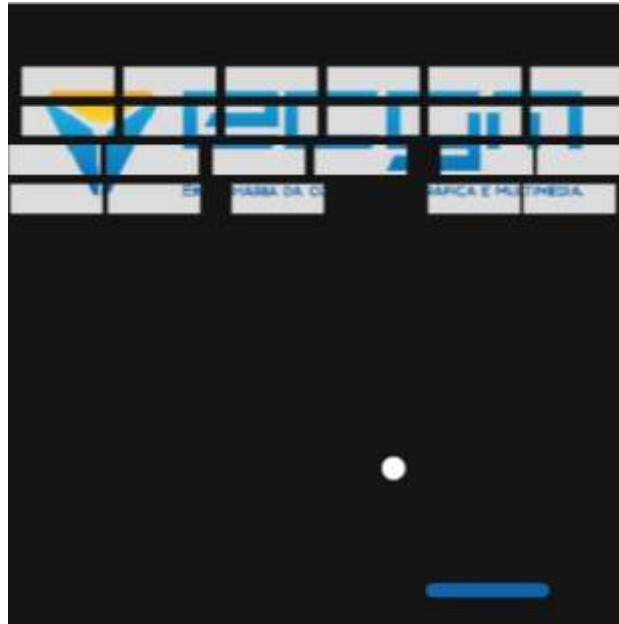
```
for (let j = i + 1; j < tijolos.length; j++) {  
  let o = tijolos[j];  
  if (!o.active) continue;  
  if (  
    t.x < o.x + o.w &&  
    t.x + t.w > o.x &&  
    t.y < o.y + o.h &&  
    t.y + t.h > o.y  
  ) {  
    let push = 0.5;  
    if (t.x < o.x) {  
      t.x -= push;  
      o.x += push;  
    } else {  
      t.x += push;  
      o.x -= push;  
    }  
    t.vx *= -0.5;  
    o.vx *= -0.5;  
  }  
}
```

```

355
356
357 //----- Movimenta os tijolos com IA e trata a colisão com as paredes -----
358 function moverTijolosComIA() {
359   for (let i = 0; i < tijolos.length; i++) {
360     let t = tijolos[i];
361     if (!t.active) continue;
362
363     // ----- IA: tijolo move-se horizontalmente para tentar alinhar com a bola -----
364     let distY = abs(yPosBola - t.y);
365
366     if (distY < 70 && yVel > 0) {
367       let direction = (xPosBola < t.x + t.w / 2) ? 1 : -1;
368
369       if (random() < t.reaction) {
370         t.vx += direction * t.speed;
371       }
372     }
373
374
375     // ----- Atualiza posição horizontal -----
376     t.x += t.vx;
377     t.vx *= 0.95;
378
379     // ----- Colisão com paredes -----
380     if (t.x <= 0) {
381       t.x = 0;
382       t.vx *= -1;
383     }
384     if (t.x + t.w >= width) {
385       t.x = width - t.w;
386       t.vx *= -1;
387     }
388
389     // ----- Evita colisão entre tijolos e aplica reação simples -----
390     for (let j = i + 1; j < tijolos.length; j++) {
391       let o = tijolos[j];
392       if (!o.active) continue;
393
394       if (
395         t.x < o.x + o.w &&
396         t.x + t.w > o.x &&
397         t.y < o.y + o.h &&
398         t.y + t.h > o.y
399       ) {
400         let push = 0.5;
401
402         if (t.x < o.x) {
403           t.x -= push;
404           o.x += push;
405         } else {
406           t.x += push;
407           o.x -= push;
408         }
409
410         t.vx *= -0.5;
411         o.vx *= -0.5;
412       }
413     }
414   }
415 }
416

```

Resultado: Os Tijolos movimentam se horizontalmente consoante a chegada da bola.



3.5. Vidas e Fim de Jogo

Declara a variável vidas e atribui o valor inicial 3:

```
let vidas = 3;
```

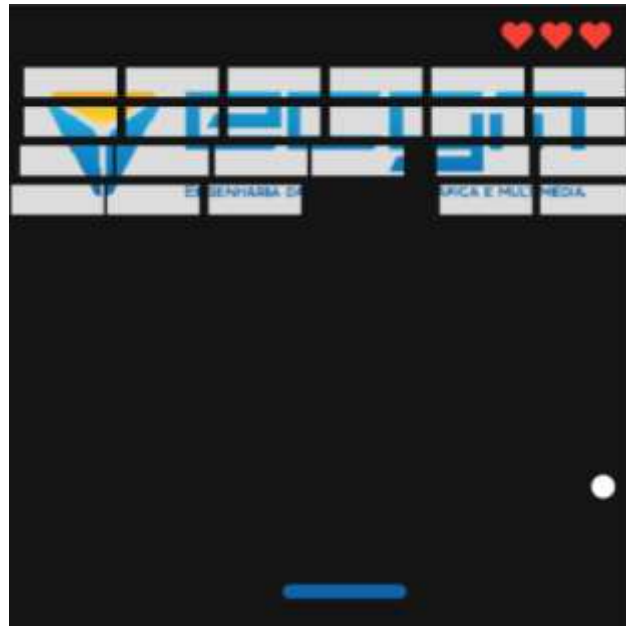
```
5
6 //---- Variáveis da plataforma ----
7 let wPaddle = 80;
8 let hPaddle = 10;
9 let xPosPaddle, yPosPaddle;
10
11 //---- Array de tijolos ----
12 let tijolos = [];
13
14 //---- Vidas ----
15 let vidas = 3;
16
```

Cria a função **desenharVidas()** para representar visualmente as vidas do jogador. Define o tamanho de cada ícone como 20 e uma margem de 5 entre eles. Calcula a posição inicial no eixo X (**startX**) com base na largura do ecrã e no número de vidas, de forma a alinhar os ícones à direita. Define também a posição vertical (y) como 10.

Em seguida, cria um ciclo que percorre o número de vidas e desenha a imagem do coração (**imgCoracao**) em cada posição calculada, espaçando-os com base no tamanho e na margem definidos.

```
function desenharVidas() {  
    let tamanho = 20;  
    let margem = 5;  
    let startX = width - (tamanho + margem) * vidas - 10;  
    let y = 10;  
  
    for (let i = 0; i < vidas; i++) {  
        image(imgCoracao, startX + i * (tamanho + margem), y, tamanho, tamanho);  
    }  
}
```

Neste momento, deves ter 3 corações a representar as vidas do jogador no canto superior direito da tela.



Seguidamente, cria a função **verificarPerdeu()**. Esta função verifica se a bola ultrapassa o limite inferior do canvas através da condição $yPosBola > height$. Quando isso acontece, o número de vidas diminui com **vidas--**. Se $vidas \leq 0$, o jogo termina com **noLoop()**.

Caso ainda existam vidas, a posição e a velocidade da bola são repostas. A bola volta para o centro na horizontal e para uma posição próxima da plataforma, e a velocidade é reiniciada para os valores iniciais.

```
function verificarPerdeu() {

    if (yPosBall > height) {

        vidas--;

        if (vidas <= 0) {

            noLoop();

        }

        xPosBall = width / 2;

        yPosBall = height - 70;

        xVel = 4;

        yVel = -4;

    }

}
```

```
134
135 //---- Diminui o número de vidas e verifica se o jogador perdeu ----
136 function verificarPerdeu() {
137     //---- Diminui o número de vidas se a posição em y da bola ultrapassa a altura do canvas ----
138     if (yPosBall > height) {
139         vidas--;
140
141         //---- Verifica se o utilizador perdeu todas as vidas e atualiza as variáveis estado e msg ----
142         if (vidas <= 0) {
143             estado = "fimJogo";
144             msg = "Não foi desta vez!";
145             return;
146         }
147
148         //---- Reset da posição e velocidade da bola ----
149         xPosBall = width / 2;
150         yPosBall = height - 70;
151         xVel = 4;
152         yVel = -4;
153     }
154 }
155
```

Implementa a função **verificarGanhou()** que percorre o array tijolos e conta quantos ainda estão ativos através da variável restantes. Sempre que encontra um tijolo ativo, incrementa o valor de restantes. No final, se restantes === 0, significa que todos os tijolos foram destruídos e o jogo termina com **noLoop()**.

```
function verificarGanhou() {
    let restantes = 0;

    for (let t of tijolos) {
        if (t.active) {
            restantes++;
        }
    }

    if (restantes === 0) {
        noLoop();
    }
}
```

```
136
137 //---- Verifica se todos os tijolos foram eliminados ----
138 function verificarGanhou() {
139
140     //---- Verifica se existem tijolos no array tijolos ---
141     let restantes = 0;
142
143     for (let t of tijolos) {
144         if (t.active) {
145             restantes++;
146         }
147     }
148     //---- Se não existirem, para o jogo ----
149     if (restantes === 0) {
150         noLoop();
151     }
152 }
153
```

Chama as funções **verificarPerdeu()** e **verificarGanhou()** dentro do método *draw()*. Estas funções permitem verificar, a cada frame, se o jogador perdeu todas as vidas ou se já destruiu todos os tijolos e terminam o jogo quando uma dessas condições se verifica.

verificarPerdeu();

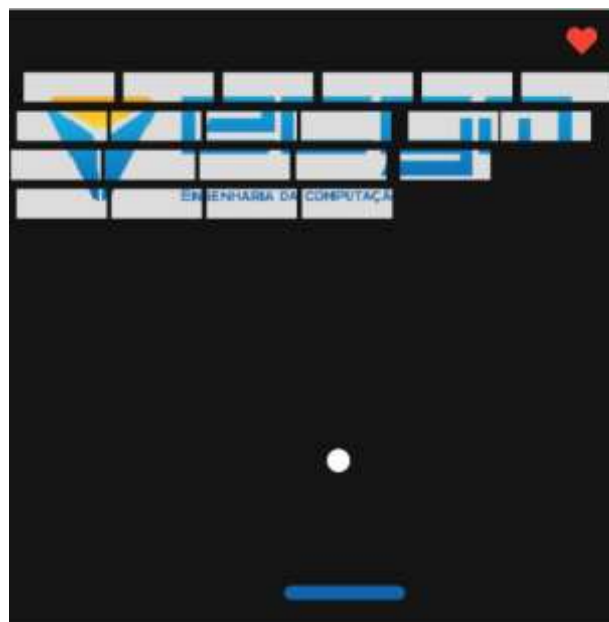
verificarGanhou();

```

38
39 function draw() {
40   background(20);
41   desenharaPlataforma();
42   moverBola(xPosPaddle, yPosPaddle);
43
44   verificarPerdeu();
45   verificarGanhou();
46
47   desenharaBola();
48   desenharaTijolos();
49
50   fill(255);
51   textSize(16);
52   textAlign(CENTER);
53
54   //---- Imprime o número de vidas do jogador na tela ----
55   text("Vidas: " + vidas, width - 35, 20);
56 }
57
58
59
60
61
62

```

Resultado: As vidas do jogador são exibidas no canto superior direito da tela. Quando o jogador perde todas as vidas ou elimina todos os tijolos, o jogo para.



3.6. Estado do Jogo e Ecrãs

Declara a variável estado, que armazena estado o do jogo (inicio, aJogar e fimJogo), e a variável msg, que guarda a mensagem que é exibida quando o jogo termina.

```
let estado = "inicio";
```

```
let msg;
```

```
10
11 //---- Array de tijolos ----
12 let tijolos = [];
13
14 //---- Vidas ----
15 let vidas = 3;
16
17 //---- Estado do jogo (inicio, aJogar, fimJogo) ----
18 let estado = "inicio";
19
20 //---- Mensagem que é exibida quando o jogo termina ----
21 let msg;
22
```

No método draw(), após a definição da cor do canvas em background(20), coloca as condições indicadas abaixo, responsáveis por verificar o valor da variável estado para controlar o que aparece no ecrã em diferentes momentos do jogo.

Quando o estado é "inicio", o programa mostra uma mensagem no centro do ecrã, que convida o jogador a começar o jogo.

Quando o estado é "fimJogo", o programa apresenta uma mensagem final guardada na variável msg, bem como uma instrução para reiniciar o jogo através do clique.

Em ambos os ecrãs, o texto está alinhado ao centro com **textAlign(CENTER)**, tem cor branca, definida por **fill(255)** e o tamanho da fonte é 16, através de **textSize(16)**.

```
if (estado === "inicio") {
    fill(255);
    textSize(16);
    textAlign(CENTER);
    text("Clica no ecrã para iniciar o jogo", width/2, height/2);
}
```

```

if (estado === "fimJogo") {

    fill(255);

    textAlign(CENTER);

    text(msg, width/2, height/2 - 10);

    text("Clica no ecrã para reiniciar o jogo", width/2, height/2 + 15);

}

```

```

58 function draw() {
59     background(20);
60
61     //---- Ecrã inicial ----
62     if (estado === "inicio") {
63         fill(255);
64         textSize(16);
65         textAlign(CENTER);
66         text("Clica no ecrã para iniciar o jogo", width/2, height/2);
67     }
68
69     //---- Ecrã fim de jogo ----
70     if (estado === "fimJogo") {
71         fill(255);
72         textAlign(CENTER);
73         text(msg, width/2, height/2 - 10);
74         text("Clica no ecrã para reiniciar o jogo", width/2, height/2 +
75         15);
76     }

```

Coloca o restante código do draw() dentro da condição **if (estado === "aJogar")**, que permite controlar o que aparece no ecrã de gameplay.

```

if (estado === "aJogar") {

    desenharPlataforma();

    ... //restante código

    desenharVidas();

}

```

O teu draw() deve estar assim:

```

43
44 function draw() {
45     background(20);
46
47     //---- Ecrã inicial ----
48     if (estado === "inicio") {
49         fill(255);
50         textSize(16);
51         textAlign(CENTER);
52         text("Clica no ecrã para iniciar o jogo", width/2, height/2);
53     }
54
55     //---- Ecrã fim de jogo ----
56     if (estado === "fimJogo") {
57         text(msg, width/2, height/2 - 15);
58         text("Clica no ecrã para reiniciar o jogo", width/2, height/2 + 15);
59     }
60
61     //---- Ecrã de gameplay ----
62     if (estado === "aJogar") {
63
64         desenharPlataforma();
65         desenharBola();
66         verificarPerdeu();
67         verificarGanhou();
68         desenharTijolos();
69         moverBola(xPosPaddle, yPosPaddle);
70
71
72         //---- Imprime o número de vidas do jogador na tela ----
73         text("Vidas: " + vidas, width - 35, 20);
74     }
75 }
76
77

```

A navegação entre os ecrãs é realizada através do clique. Para isso, utiliza-se a função **mousePressed()**, que é executada sempre que o jogador clica no ecrã. Se ocorrer um clique e o estado for "inicio", este passa a ser "aJogar", o que permite ao utilizador ver o ecrã de jogo. Quando o estado é "fimJogo", ou seja, quando o utilizador ganha ou perde, e é detetado um clique, a função **reiniciarJogo()** é chamada.

```

function mousePressed() {

    if (estado === "inicio") {

        estado = "aJogar";

    } else if (estado === "fimJogo") {

        reiniciarJogo();

    }

}

```

```
//---- Permite a transição entre ecrãs através da deteção o click do utilizador ----
function mousePressed() {
  if (estado === "inicio") {
    estado = "aJogar";
  } else if (estado === "fimJogo") {
    reiniciarJogo();
  }
}
```

Dentro do método verificarPerdeu(), altera a condição if (vidas <= 0) para que, quando esta for verdadeira, ou seja, quando o utilizador perder todas as vidas, o valor da variável estado passe a ser “fimJogo” e a variável msg armazene uma mensagem a informe o utilizador de que perdeu. A instrução return no final do bloco permite interromper a execução do método.

```
if (vidas <= 0) {

  estado = "fimJogo";

  msg = "Não foi desta vez!";

  return;

}
```

```
134
135 //---- Diminui o número de vidas e verifica se o jogador perdeu ----
136 function verificarPerdeu() {
137   //---- Diminui o número de vidas se a posição em y da bola ultrapassa a altura do canvas ----
138   if (yPosBall > height) {
139     vidas--;
140
141     //---- Verifica se o utilizador perdeu todas as vidas e atualiza as variáveis estado e msg ----
142     if (vidas <= 0) {
143       estado = "fimJogo";
144       msg = "Não foi desta vez!";
145       return;
146     }
147
148     //---- Reset da posição e velocidade da bola ----
149     xPosBall = width / 2;
150     yPosBall = height - 70;
151     xVel = 4;
152     yVel = -4;
153   }
154 }
155
```

Agora, na função `verificarGanhou()` altera a condição `if (restantes === 0)` para que, quando esta for verdadeira, ou seja quando o utilizador eliminou todos os tijolos, altere, também, valor da variável estado para “fimJogo” e a variável msg armazene uma mensagem a informar o utilizador de que ganhou.

```
if (restantes === 0) {
    estado = "fimJogo";
    msg = "Ganhaste!";
}
```

```
155
156 //---- Verifica se todos os tijolos foram eliminados ----
157* function verificarGanhou() {
158
159     //---- Verifica se existem tijolos no array tijolos ---
160     let restantes = 0;
161
162*   for (let t of tijolos) {
163*       if (t.active) {
164           restantes++;
165       }
166   }
167
168*   //---- Se não existirem, atualiza as variáveis estado e msg ----
169*   if (restantes === 0) {
170       estado = "fimJogo";
171       msg = "Ganhaste!";
172   }
173 }
```

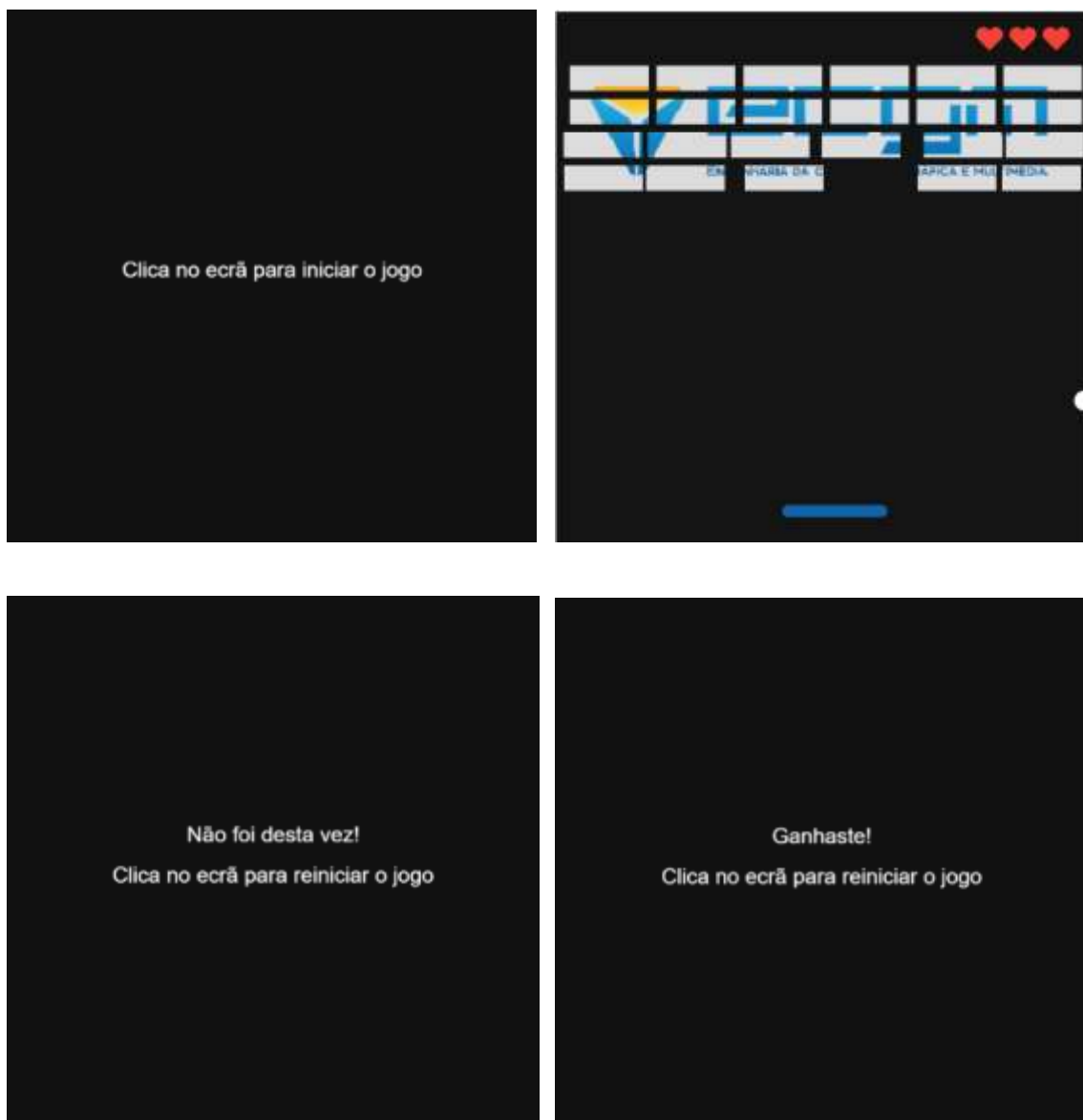
Cria a função **reiniciarJogo()**, que permitirá repor o jogo ao seu estado inicial após o jogador perder ou vencer: define as vidas para 3, coloca a bola no centro inferior e restaura a sua velocidade para os valores iniciais. Em seguida, percorre todos os tijolos com um loop for e reativa cada um deles. Por fim, muda o estado do jogo para “aJogar”, o que possibilita mudança do ecrã para o de gameplay.

```
function reiniciarJogo() {  
    vidas = 3;  
    xPosBola = width / 2;  
    yPosBola = height - 70;  
    xVel = 4;  
    yVel = -4;  
  
    for (let t of tijolos) {  
        t.active = true;  
        t.vx = 0;  
    }  
    //---- Muda o estado do jogo ---  
    estado = "aJogar";  
}
```

```
288 function reiniciarJogo() {  
289     vidas = 3;  
290     xPosBola = width / 2;  
291     yPosBola = height - 70;  
292     xVel = 4;  
293     yVel = -4;  
294  
295     for (let t of tijolos) {  
296         t.active = true;  
297         t.vx = 0;  
298     }  
299     //---- Muda o estado do jogo ---  
300     estado = "aJogar";  
301 }
```

Resultado: Ecrã inicial que informa o utilizador de que deve clicar na tela para iniciar o jogo. Após o clique, o jogo transita para o ecrã de gameplay. Quando o jogador perde todas as vidas, o canvas muda para um fundo escuro com duas mensagens: uma a indicar que perdeu o jogo e outra a informar o utilizador que deve clicar na tela para reiniciar. Caso todos os tijolos sejam eliminados, o ecrã apresentado é semelhante, mas com uma mensagem de vitória. Em ambos os casos, ao clicar na tela, o jogo é reiniciado.

Se completaste todos os passos, os teus ecrãs devem estar assim:



4. Conclusão

Neste workshop, exploraste as vastas capacidades da biblioteca **p5.js** para construir um jogo interativo completo, que alinha a lógica algorítmica à criatividade visual. Ao longo das sessões, foi possível converter conceitos teóricos de programação em elementos tangíveis, como o movimento físico da bola, a interação com a plataforma e a destruição dinâmica de objetos.

Este projeto serve agora como uma base sólida para que possas expandir o código e dar asas à tua criatividade na implementação de novas funcionalidades.

Sugestões para desenvolvimentos futuros:

1. **Novos Níveis e Dificuldade:** Implementa diferentes padrões de tijolos ou aumenta a velocidade da bola progressivamente.
2. **Efeitos Sonoros:** Utiliza a biblioteca `p5.sound` para adicionar áudio nas colisões e música de fundo.
3. **Power-ups:** Cria itens que caem dos tijolos e conferem vantagens temporárias, como o aumento da plataforma.
4. **Refinamento Estético:** Explora filtros visuais, sistemas de partículas e novos esquemas de cores para tornar o jogo visualmente único.

Parabéns pelo empenho e pelos resultados alcançados! Esperamos que este workshop tenha sido o teu primeiro passo de uma longa jornada na área da computação gráfica e multimédia.

5. Referências

Colocar as referências para os softwares, outros tutoriais, plugins, livros, vídeos, etc.

P5.js

<https://p5js.org/>

Licenciatura em Engenharia da Computação Gráfica e Multimédia (**ECGM**)

<https://www.ipvc.pt/estg/courses/engenharia-da-computacao-grafica-e-multimedia/?mp=213&mc=299>

Jornadas da Computação Gráfica e Multimédia (**JCGM**)

<http://jcgm.estg.ipvc.pt/>

