Academia Verão

Programação Criativa

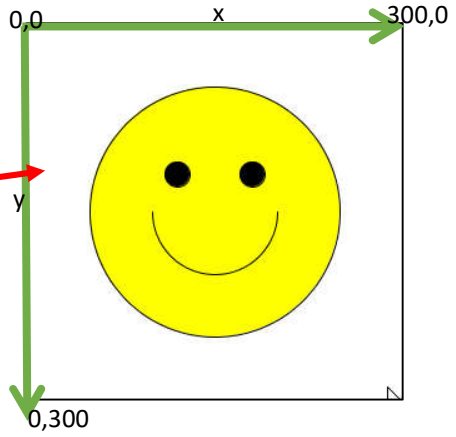
4 a 8 e 11 a 15 de julho

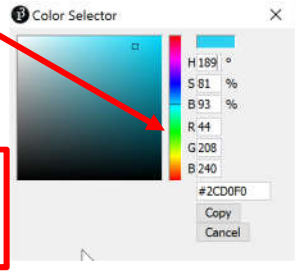
Arnaldo Martins / José Moreira

(jam@ua.pt, jose.moreira@ua.pt)

Processing.org

```
/* Primeiro programa - desenho de uma cara
   JAM 2016-6-6
 */
 // A função setup() é executada uma vez no início do programa
void setup() {
  size( 300, 300 );                    // tamanho da tela
}
// A função draw() é repetida várias vezes por segundo (25/30)
void draw() {
  background( 255 );                   // cor do fundo (0 – preto; 255 – branco)
  fill( 255, 255, 0 );                 // cor dos objectos (combinação de Red, Green, Blue)
  ellipse( 150, 150, 200, 200 );       // desenha cabeça

  fill(0);
  ellipse( 120, 120, 20, 20 );         // desenha olho esq
  ellipse( 180, 120, 20, 20 );         // desenha olho dir

  noFill();                                          // desenha só contorno
  arc( 150, 150, 100, 100, 0, PI );    // desenha boca
}
```

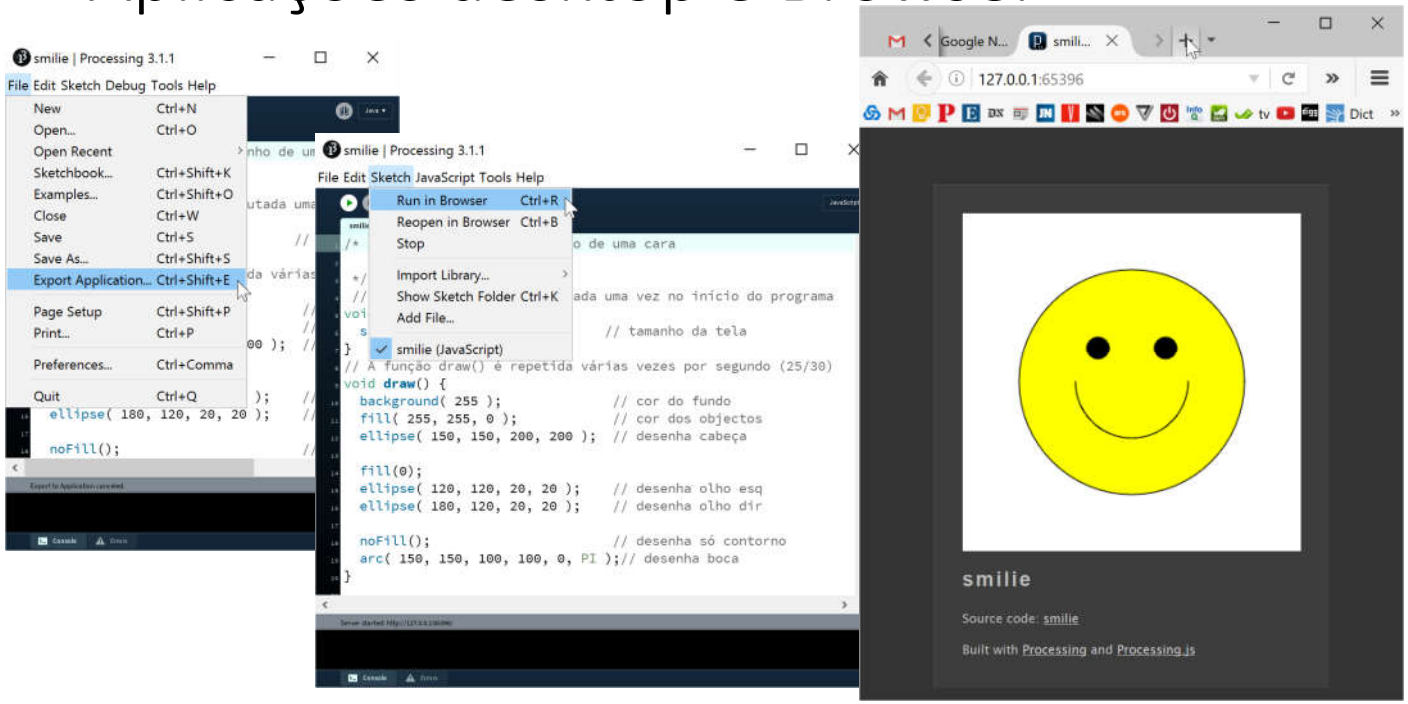




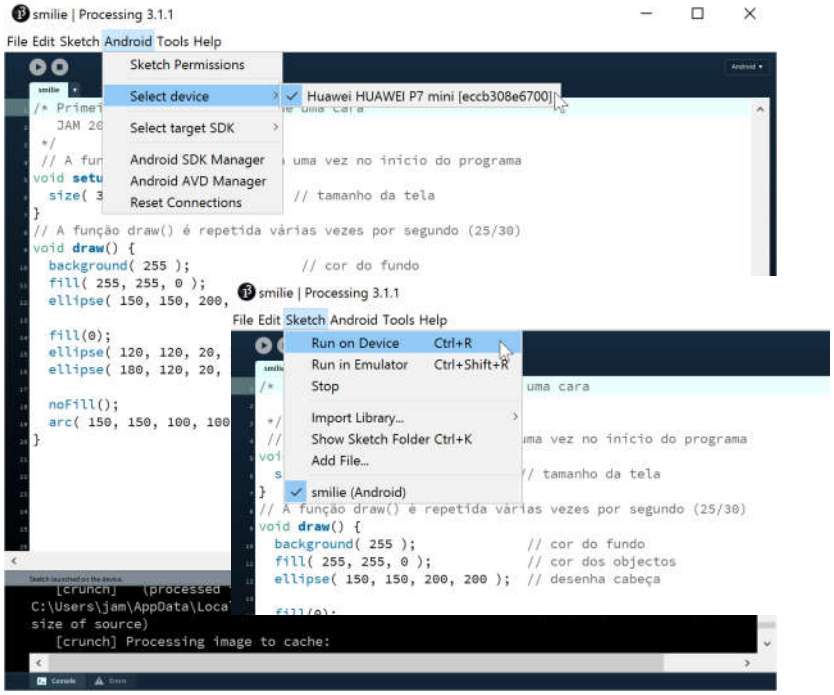
Algoritmo/programa = sequência de instruções/comandos

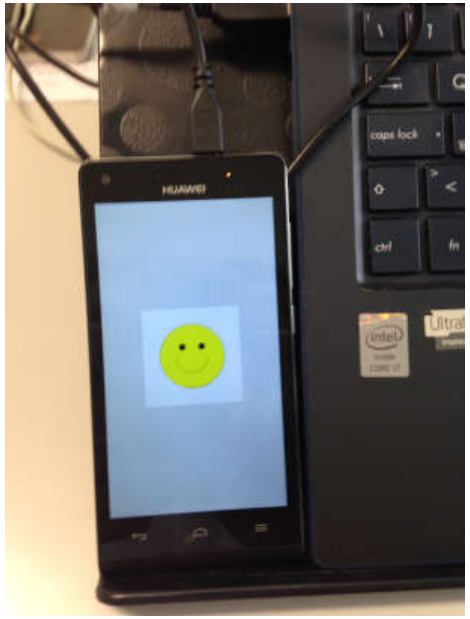
/* … */ ou // definem comentários de texto, não são instruções!

Aplicações desktop e Browser



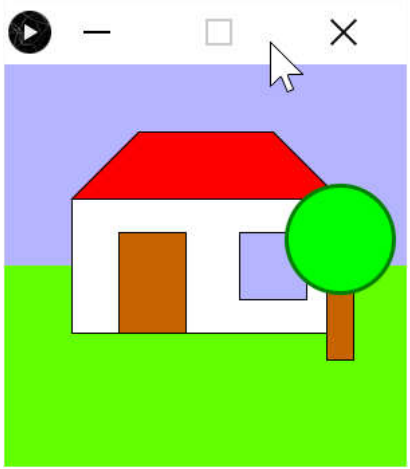
Aplicações Android





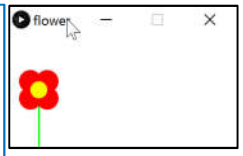
casa

```
void setup() {
  size(300,300);
}
void draw() {

  background(180,180,255);
  fill( 100, 255, 0 );
  noStroke();
  rect(0,150,width,150);
  stroke(0);
  strokeWeight(1);

  fill(255);
  rect( 50,100,200, 100);
  fill(200, 100, 0);
  rect( 85, 125, 50, 75 );
  fill( 180, 180, 255 );
  rect( 175, 125, 50, 50 );

  fill(255,0,0);
  beginShape();
  vertex(50,100);
  vertex(100,50);
  vertex(200,50);
  vertex(250,100);
  endShape(CLOSE);

  fill(200,100,0);
  rect( 240,130,20,90);

  fill(0,255,0);
  stroke(0,128,0);
  strokeWeight(3);
  ellipse(250,130,80,80);
}
```

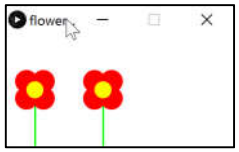


Flor(es)

```
void setup() {
  size(400, 200);
}

void draw() {
  background(255);
  strokeWeight(3);
  stroke(0, 255, 0);
  line( 50, 200, 50, 100 );
  noStroke();
  fill( 255, 0, 0 );
  ellipse( 35, 85, 40, 40 );
  ellipse( 65, 85, 40, 40 );
  ellipse( 65, 115, 40, 40);
  ellipse( 35, 115, 40, 40 );
  fill( 255, 255, 0 );
  ellipse( 50, 100, 30, 30 );
}
```




```
void draw() {
  background(255);
  strokeWeight(3);
              …
//desenho da 2ª flor
  strokeWeight(3);
  stroke(0, 255, 0);
  line( 50+120, 200, 50+120, 100 );
  noStroke();
  fill( 255, 0, 0 );
  ellipse( 35+120, 85, 40, 40 );
  ellipse( 65+120, 85, 40, 40 );
  ellipse( 65+120, 115, 40, 40);
  ellipse( 35+120, 115, 40, 40 );
  fill( 255, 255, 0 );
  ellipse( 50+120, 100, 30, 30 );
}
```



variáveis

```
void draw() {
  int dx = 0;
  background(255);
  //desenho 1ª flor
  strokeWeight(3);
  stroke(0, 255, 0);
  line( 50+dx, 200, 50+dx, 100 );
  noStroke();
  fill( 255, 0, 0 );
  ellipse( 35+dx, 85, 40, 40 );
  ellipse( 65+dx, 85, 40, 40 );
  ellipse( 65+dx, 115, 40, 40);
  ellipse( 35+dx, 115, 40, 40 );
  fill( 255, 255, 0 );
  ellipse( 50+dx, 100, 30, 30 );

  //desenho da 2ª flor
  dx = 120;
  strokeWeight(3);
  stroke(0, 255, 0);
  line( 50+dx, 200, 50+dx, 100 );
  noStroke();
  fill( 255, 0, 0 );
  ellipse( 35+dx, 85, 40, 40 );
  ellipse( 65+dx, 85, 40, 40 );
  ellipse( 65+dx, 115, 40, 40);
  ellipse( 35+dx, 115, 40, 40 );
  fill( 255, 255, 0 );
  ellipse( 50+dx, 100, 30, 30 );
}
```

Tipos de varáveis

boolean
byte
char
color
double
float
int
long

String

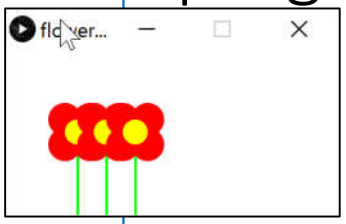
Operadores

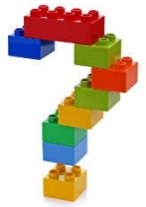
% (modulo, resto)
* (multiplicação)
/ (divisão)
+ (adição)
- (subtração)

++ (incremento)
-- (decremento)

Abstração, funções, programação modular

```
void setup() {
  size(400, 200);
}

void draw() {
  background(255);
  flower( 50 + 35 * 1, 100 );
  flower( 50 + 35 * 2, 100 );
  flower( 50 + 35 * 3, 100 );
}

void flower( int posx, int altura) {
  strokeWeight(3);
  stroke(0, 255, 0);
  line( posx, 200, posx, 200 - altura );
  noStroke();
  fill( 255, 0, 0 );
  ellipse( posx - 15, 200 - altura - 15, 40,
40);
  ellipse( posx + 15, 200 - altura - 15, 40, 40
);
  ellipse( posx + 15, 200 - altura + 15, 40,
40);
  ellipse( posx - 15, 200 - altura + 15, 40, 40
);
  fill( 255, 255, 0 );
  ellipse( posx, 200 - altura, 30, 30 );
}
```



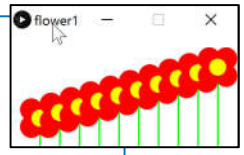


Tipo nome_função (tipo parâmetro 1, tipo parâmetro 2, …)

```
{
   …
   instruções
   …

}
```

Ciclos, repetição de instruções

```
void setup() {
  size(400,200);
}

void draw() {
  background(255);

  for( int i=0; i < 10; i++ ) {
    flower( 50 + 35 * i, 50 +10* i );
  }
}

void flower( int posx, int altura) {
…
}
```



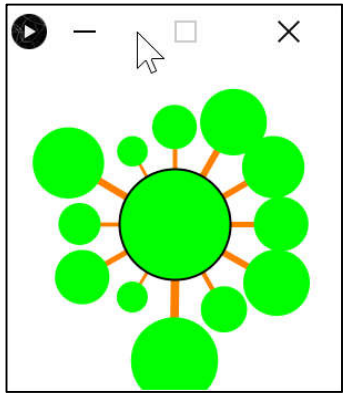

```
for (valor inicial; condição de paragem ; atualização) {
    instruções
}
```

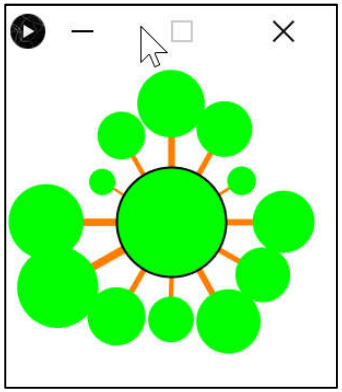
```
while (expressão verdadeira) {
    instruções
}
```

Rotação, escala, translação

```
void setup() {
  size(300,300);
  smooth();
  noLoop();
}

void draw() {
  background(255);

  for( int i=0; i < 12; i++ ) {
    pushMatrix();
    translate( 150,150 );
    rotate(radians(30 * i));
    translate(0, -50);
    float s = random( 0.3, 1 );
    scale(s,s);
    tree();
    popMatrix();
  }
  strokeWeight(2);
  stroke(0);
  fill(0, 255, 0);
  ellipse( 150,150,100,100 );
}
```

```
void tree() {
  strokeWeight(8);
  stroke(255, 128, 0);
  line( 0, 0, 0, -75 );
  noStroke();
  fill(0, 255, 0 );
  ellipse( 0, -75, 80, 80 );
}
```

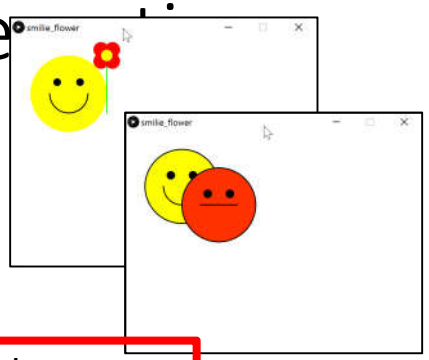




Questões, caminhos alte

```
int count=0;
 void setup() {
 size( 800, 600 );
}

void draw() {
 background( 255 );
 smile(150, 150, 255, 100);
 if ( count < 150 ) {
   smile(250, 200, 50, 0);
 } else {
   flower (250, 150);
 }
 if (++count >= 300) count =
0;
}
void flower( int pos, int hi) {
 strokeWeight(3);
 stroke(0, 255, 0);
 line( pos, 200, pos, 200 - hi
);
```

```
 noStroke();
 fill( 255, 0, 0 );
 ellipse( pos - 15, 200 - hi - 15, 40, 40 );
 ellipse( pos + 15, 200 - hi - 15, 40, 40 );
 ellipse( pos + 15, 200 - hi + 15, 40, 40);
 ellipse( pos - 15, 200 - hi + 15, 40, 40 );
 fill( 255, 255, 0 );
 ellipse( pos, 200 - hi, 30, 30 );
}
void smile(int x, int y, int cor, int sorriso) {
 fill( 255, cor, 0 );
 ellipse( x, y, 200, 200 );
 fill(0);
 ellipse( x-30, y-30, 20, 20 );
 ellipse( x+30, y-30, 20, 20 );
 noFill();
 stroke(0);
 arc( x, y, 100, sorriso, 0, PI );
}
```



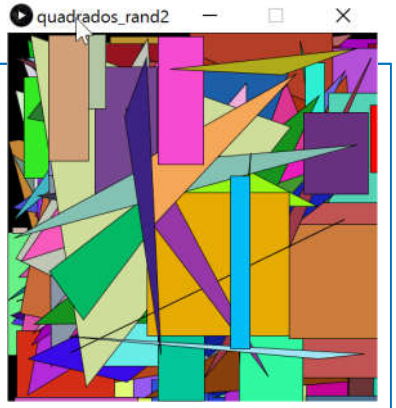
Operadores relacionais

!= (diferente)
< (menor)
<= (menor ou igual)
== (igual)
> (maior)
>= (maior ou igual)

Operadores Lógicos

! (logical NOT)
&& (logical AND)
|| (logical OR)

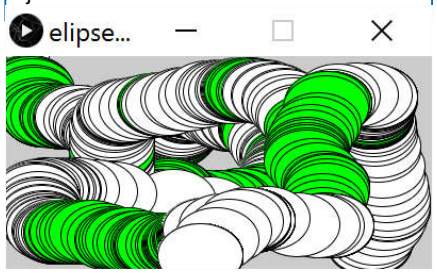
quadro abstrato …




```
int count=300;
void setup() {
  size (510, 510);
  background (0);
}

void draw () {
  fill (random (0, 255), random(0, 255), random (0, 255));
  if (random(0, 10) > 5 ) {
    rect (random(0, width), random(0, height), random(width/2), random(height/2));
  } else {
    triangle(random(0, width), random(0, height),random(0, width), random(0, height),random(0, width), random(0,
height));
  }
  if ( --count == 0 ) noLoop();
}
```

Interatividade

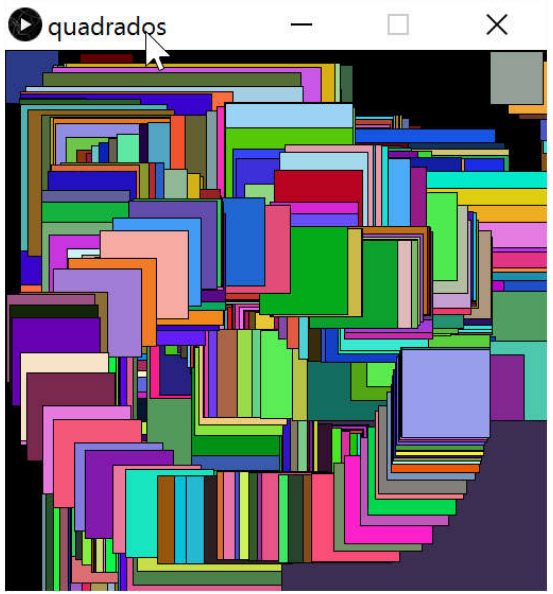
```
void setup() {
  size(400, 200);
}

void draw() {
   if (mousePressed) {
    fill(0,255,0);
  } else {
    fill(255);
  }
  ellipse( mouseX, mouseY, 80, 60);
}
```




```
void draw() {
  background(128);
  if (mousePressed) {
    fill(0,255,0);
  } else {
    fill(255);
  }
  ellipse( mouseX, mouseY, 80, 60);
}
```

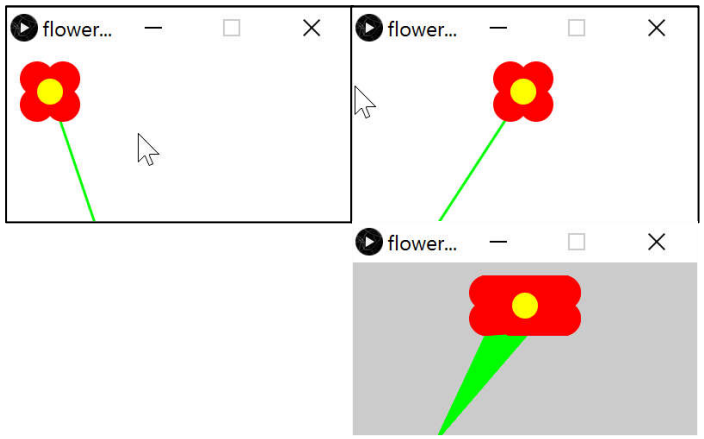
Quadro abstrato interativo …

```
int dx=0;
void setup() {
  size (510, 510);
  background (0);
}

void draw () {
  fill (random (0, 255), random(0, 255), random (0, 255));
  rect (mouseX, mouseY, 50 + dx, 50 + dx );
  if (keyPressed) {
    dx = (int)random (300);
  }
}

void keyPressed () {
  if (key == 'f') noLoop();
}
```



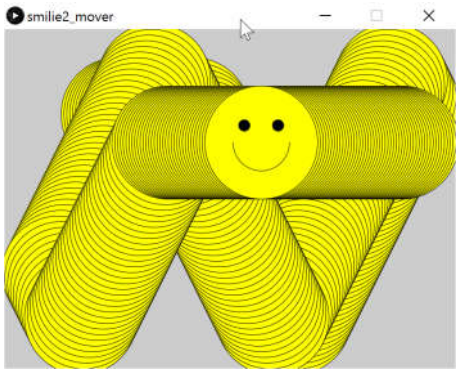
```
float vento;
void setup() {
 size(400, 200);
}

void draw() {
 background(255);
 vento = vento + PI/20;
 flower(100 + (100-mouseX) + (int)(20*sin(vento)),
50, 100, 200 );}

void flower( int posx, int posy, int basex, int basey)
{
 strokeWeight(3);
 stroke(0, 255, 0);
 line( posx, posy, basex, basey );
 noStroke();
 fill( 255, 0, 0 );
 ellipse( posx - 15, posy - 15, 40, 40 );
 ellipse( posx + 15, posy - 15, 40, 40 );
 ellipse( posx + 15, posy + 15, 40, 40);
 ellipse( posx - 15, posy + 15, 40, 40 );
 fill( 255, 255, 0 );
 ellipse( posx, posy, 30, 30 );
}
```

movimento



```
int x=200, y=150, velx = 5, dy=10;
void setup() {
  size( 800, 600 );
}

void draw() {
  background( 255 );
  smile(x, y, 255,100);
  x = x+velx;
  if (x > width-100 || x < 100) velx = -velx;
  if (y > height - 100 | y <100 ) dy= -dy;
  if (mousePressed) y = y + dy;

}
void smile(int x, int y, int cor, int sorriso) {
  fill( 255, cor, 0 );
  ellipse( x, y, 200, 200 );
  fill(0);
  ellipse( x-30, y-30, 20, 20 );
  ellipse( x+30, y-30, 20, 20 );
  noFill();
  arc( x, y, 100, sorriso, 0,PI );
}
```

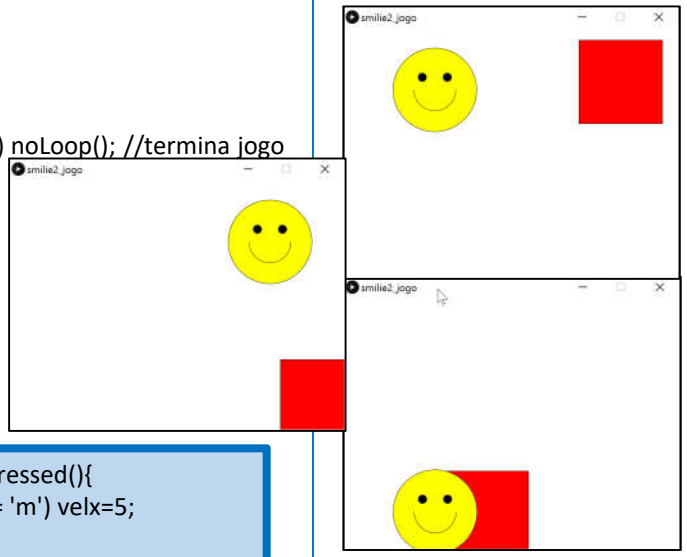
Movimento, limites



Jogo, smilie

```
int x=101, y=150, velx = 1, dy=10, alvox=600, alvoy=200, alvolado=200;
void setup() {
  size( 800, 600 );
}

void draw() {
  background( 255 );
  fill(255, 0, 0);
  rect(alvox, alvoy, alvolado, alvolado);
  smile(x, y, 255, 100);
  if (distancia(x, y, alvox+alvolado/2, alvoy+alvolado/2) < 200) noLoop(); //termina jogo
  x = x+velx;
  if (x > width-100 || x < 100) {
    velx = -velx;
    alvox=(int)random(0, width);
    alvoy=(int)random(0, height);
  }
  if (y > height - 100 | y <100 ) dy= -dy;
  if (mousePressed) y = y + dy;
}
void smile(int x, int y, int cor, int sorriso) {   …. }
float distancia(int x, int y, int x1, int y1) {
  return sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1));
}
```

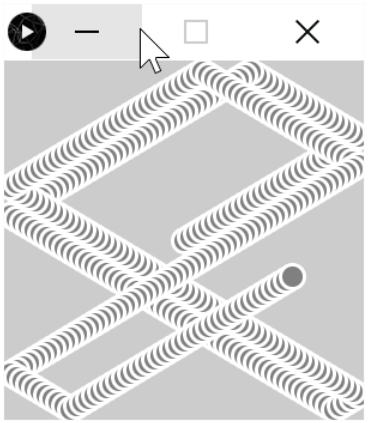
```
void keyPressed(){
  if (key == 'm') velx=5;
}
// aumenta nível de dificuldade
```



bounce

```
PVector pos;
PVector vel;

void setup() {
  size(300,300);
  smooth();
  strokeWeight(3);
  fill(128);
  stroke( 255 );
  frameRate( 25 );
  pos = new PVector( width/2, height/2 );
  vel = new PVector( 5, -3 );
}
```

```
void draw() {
  background(0);
  ellipse( pos.x, pos.y, 20, 20);
    pos.add( vel );
  if ( pos.x + 10 > width ||  pos.x - 10  < 0 ) {
    vel = new PVector( -vel.x, vel.y );
  }
    if ( pos.y + 10 > height || pos.y - 10 < 0 ) {
    vel = new PVector( vel.x, -vel.y );
  }
}
```



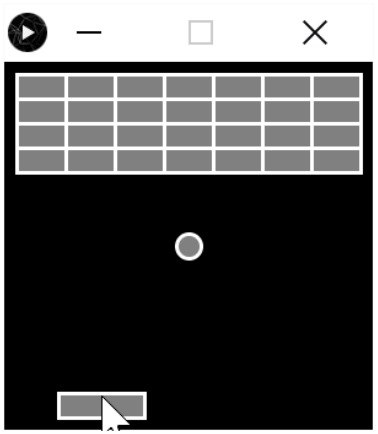
Matrizes, texto

```
String[] palavras = {"ria", "olá", "chuva", "verão", "brexit"};
int NPALS = palavras.length;
int[] pesos = new int[NPALS];
void setup() {
  size(800, 800);
  background(0);
  noLoop();  // se tirar sobrepõe texto
  pesos[0] = 150;
  for (int i=1; i < palavras.length; i++) {
    pesos[i] = (int)random(150);
  }
}
void draw() {
  for (int i=0; i < palavras.length; i++) {
    fill(random(255), random(255), random(255));
    textSize( pesos[i] );
    text(palavras[i], width/4 + random(width/2), height/4 + random(height/2));
  }
}
```

Matrizes, Jogo arkanoid

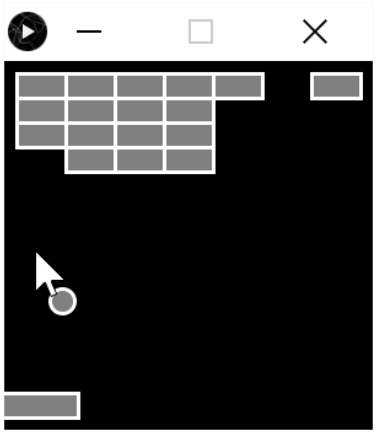
```
PVector pos;
PVector vel;
int[][] stones;
boolean paused = true;

void setup() {
  size(300, 300);
  smooth();
  strokeWeight(3);
  fill(128);
  stroke( 255 );
  frameRate( 25 );
  pos = new PVector( width/2, height/2 );
  vel = new PVector( 10, -5 );
// matriz com blocos (=1 existe)
  stones = new int[7][4];
  for ( int x = 0; x < 7; x++ ) {
    for ( int y = 0; y < 4; y++ ) {
      stones[x][y] = 1;
    }
  }
}
```

```
void draw() {
  background(0);
  if ( !paused ) {
    update();
  }
// desenha blocos que existam
  for ( int x = 0; x < 7; x++ ) {
    for ( int y = 0; y < 4; y++ ) {
      if ( stones[x][y] > 0 ) {
        fill( 128 );
        rect( 10 + x * 40, 10 + y * 20, 40, 20 );
      }
    }
  }
  ellipse( pos.x, pos.y, 20, 20);
  rect( mouseX - 35, 270, 70, 20 );
}

void mousePressed() {
  paused = !paused;
}
```



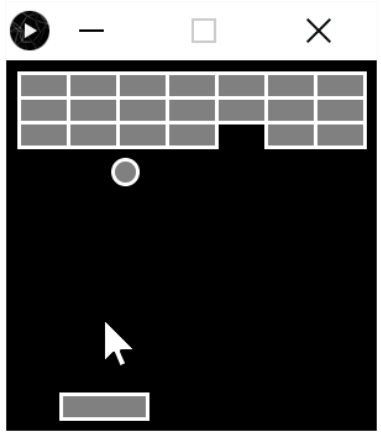
Jogo arkanoid (2)

```
void update() {
  pos.add( vel );
  if ( pos.x + 10 > width ||  pos.x - 10  < 0 ) {
    vel = new PVector( -vel.x, vel.y );
  }

  if ( pos.y - 10 < 0 ) {
    vel = new PVector( vel.x, -vel.y );
  }
// se bola passa a raquete acaba o jogo
  if ( pos.y + 10 > height ) {
    vel = new PVector( vel.x, -vel.y );
    pos = new PVector( width/2, height/2 );
    paused = true;
  }
  // testa se bola bateu na raquete
  if ( pos.y >= 260  && pos.x >= mouseX - 35 && pos.x <= mouseX +35 ) {
    vel = new PVector( int(map( pos.x - mouseX, -35, 35, -10, 10 )), -
vel.y);
  }
```



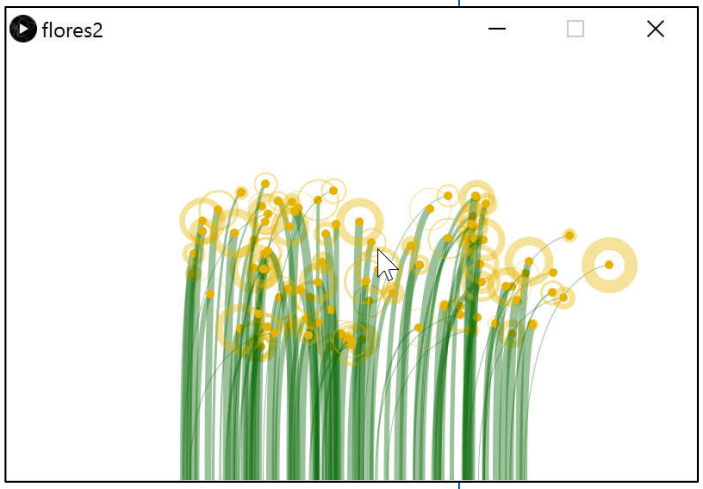
Jogo arkanoid (3)

```
// testa se a bola bateu num bloco
for ( int x = 0; x < 7; x++) {
  for ( int y = 0; y < 4; y++ ) {
    if ( stones[x][y] > 0 ) {
      if ( pos.x + 10 > 10 + x * 40 && pos.x - 10 < 10 + x * 40 + 40 &&
        pos.y + 10 + y * 20 > 10 && pos.y - 10 < 10 + y * 20 + 20 ) {
        stones[x][y] = 0; // apaga bloco
        // muda a velocidade na direção y se o bloco foi batido em baixo ou em cima
        if ( pos.x > 10 + x * 40 && pos.x < 10 + x * 40 + 40 ) {
          vel = new PVector( vel.x, -vel.y );
        }
        // muda velocidade na direção x se o bloco foi batido nos lados
        if ( pos.y > 10 + y * 20  && pos.y < 10 + y * 20 + 20 ) {
          vel = new PVector( -vel.x, vel.y );
        }
      }
    }
  }
}
```



Matrizes, campo flores

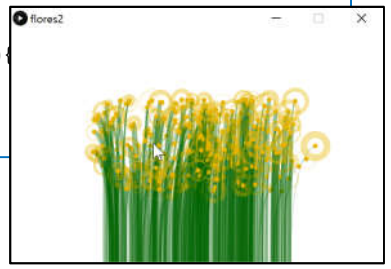
```
/* 2016-5-31, Desenha campo de flores */
int SW, SH;
int NFLORES;     //numero total de flores a desenhar
float vento, incvento, inc;  // Vento; angulo em radianos para fazer oscilacar; variavel inc incremento para fazer oscilar
int[][] flores;   // Declaracao da lista de flores a desenhar - Matrizes
void setup() {
  size(800, 500);
  background(255);
  SW = 800;
  SH = 500;
  NFLORES = int (random(1, 1250));
  //Inicializacao de flores MANUAL para teste da primeira
  //{locx, locy, altura, espessura, diametro petala, largura petala}
  flores = new int[NFLORES][6];
  flores[0][0] = SW/2+SW/4; //locx
  flores[0][1] = SH; //locy
  flores[0][2] = SH/2; // altura
  flores[0][3] = 1; // espessura
  flores[0][4] = 50; // diametro da petala
  flores[0][5] = 15; // largura da petala
  //inicializacao automatica - percorrer a matriz
  for (int n=1; n < NFLORES; n++) {
    flores[n][0] = SW/2+int(random(-SW/4, SW/4));
    flores[n][1] = SH;
    flores[n][2] = SH/2+int(random(-100, 100));
    flores[n][3] = int(random(10)+1);
    flores[n][4] = int(random(50)+1);
    flores[n][5] = int(random(10)+1);
  }
  vento = 0.0;  // inicializacao do vento
  incvento = PI/100; //PI radianos = 180 graus incremento a dar ao valor vento para "circular"
}
```



Matrizes, campo flores (2)

```
void draw() {
  background(255);
  //desenhar a primeira flor
  //{locx, locy, altura, espessura, diametro petala, largura petala}
  drawflower(flores[0][0], flores[0][1], flores[0][2], flores[0][3], flores[0][4], flores[0][5]);
  // invocar o metodo de flores personalizado em ciclo a percorrer a matriz
  for (int n = 1; n<flores.length; n++) {
     drawflower(flores[n][0], flores[n][1], flores[n][2], flores[n][3], flores[n][4], flores[n][5]);
  }
  vento=(vento+incvento)%TWO_PI;    // adicionar o incremento ao vento
  inc = sin(vento)*100; // incremento da oscilacao do vento
}
```

```
//{locx, locy, altura, espessura, dpetalas, largpetala}
void drawflower(int p0, int p1, int p2, int p3, int p4, int p5) {
  //localizacao da flor
  int intflocx = int(p0+inc/p3); // loc horizontal + vento mais espesssura/resistencia do caule
  int intflocy = p2;
  // verificar a posicao do rato e alterar a localizacao da flor
  float c = dist(mouseX, mouseY, intflocx, intflocy);
  if (c<100) {
    if (intflocx<mouseX) {
      intflocx = intflocx+(intflocx-mouseX)/2;
    } else if (intflocx>mouseX) {
      intflocx = intflocx-(mouseX-intflocx)/2;
    }
  }
  //desenho do Caule do chao para a flor
  noFill();
  stroke(0, 100, 0, 100);
  strokeWeight(p3);
  //{locx, locy, altura, espessura, dpetalas, largpetala}
  bezier(p0, p1, p0, p2, intflocx, intflocy, intflocx, intflocy);
  // desenho da flor
  fill(230, 180, 0);
  noStroke();
  ellipse(intflocx, intflocy, 10, 10);
  // desenho da petala
  noFill();
  stroke(230, 180, 0, 100);
  strokeWeight(p5);
  ellipse(intflocx, intflocy, p4, p4);
}

void mousePressed() {
  // restart
  setup();
}
```



Objetos (estruturas de dados), Starfield

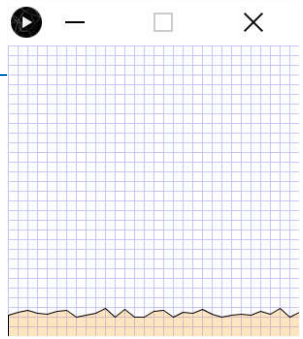
```
Star stars[];
int STARS = 100;
void setup() {
 size(640,480);
 stars = new Star[STARS];
 for ( int i =0; i < STARS; i++) {
  stars[i] = new Star( random( width ), random( height ), random( 10 ));
 }
 frameRate( 25 );
}
void draw() {
 background(0);
 starfield();
}
void starfield() {
for ( int i =0; i < STARS; i++) {
  strokeWeight( stars[i].z );
  stroke( stars[i].z * 25);
  fill(stars[i].z * 25);
  ellipse( stars[i].x, stars[i].y , 5,5
  stars[i].x = stars[i].x - stars[i].z;
  if (stars[i].x < 0) {
   stars[i] = new Star( width, random( height ), sqrt(random( 100 )));
  } } }
```

```
class Star {  // Classe é um molde para novos
objetos
 float x, y, z; // dados (podem ser de qualquer tipo)
 Star( float x, float y, float z ) { // função Star
  this.x = x;                  // é o Construtor
  this.y = y;                  // chamada quando
  this.z = z;                  // se cria o objeto
 }
}
```

Lua

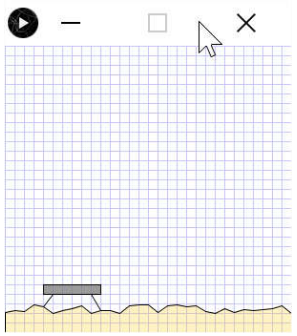
```
int[] moon;
void setup() {
  size( 300, 300 );
  moon = new int[width/10+1];
  for ( int i=0; i < moon.length; i++) {
    moon[i] = int( random( 10 ));
  }
}
void draw() {
  background(255);
  stroke(200, 200, 255);
  for ( int i=0; i<height/10; i++) {
    line( 0, i*10, width, i*10);
  }
  for ( int i=0; i<width/10; i++) {
    line( i*10, 0, i*10, height );
  }
  drawMoon();
}
```

```
void drawMoon() {
  stroke(0);
  fill(255, 150, 0, 60);
  beginShape();
  vertex(0, height);
  for ( int i=0; i < moon.length; i++) {
    vertex( i * 10, height - 20 - moon[i] );
  }
  vertex(width, height);
  endShape(CLOSE);
}
```



Lua, plataforma

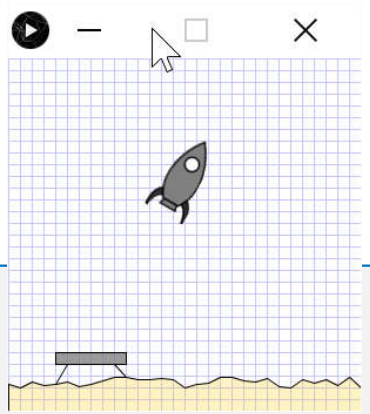
```
int[] moon;
int landingX = 0; // plataforma

void setup() {
  size( 300, 300 );
  moon = new int[width/10+1];
  for ( int i=0; i < moon.length; i++) {
    moon[i] = int( random( 10 ));
  }
  landingX = int( random(3, moon.length-4))*10; // plataforma
}
void draw() {
  background(255);
  stroke(200, 200, 255);
  for ( int i=0; i<height/10; i++) {
    line( 0, i*10, width, i*10);
  }
  for ( int i=0; i<width/10; i++) {
    line( i*10, 0, i*10, height );
  }
  drawMoon();
  drawLandingZone(); // plataforma
}
```



```
void drawMoon() {
    ....
}

//plataforma
void drawLandingZone() {
  fill(128, 200);
  rect( landingX - 30, height - 50, 60, 10);
  line( landingX - 30, height - 20 - moon[landingX/10-3], landing - 20, height - 40 );
  line( landingX + 30, height - 20 - moon[landingX/10 +3], landingX + 20, height - 40
);
}
```

Lua, plataforma, nave

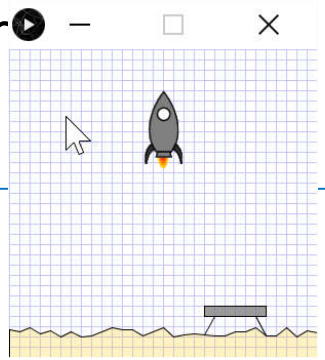
```
int[] moon;
int landingX = 0; // plataforma
PImage ship; // nave

void setup() {
  size( 300, 300 );
  moon = new int[width/10+1];
  for ( int i=0; i < moon.length; i++) {
    moon[i] = int( random( 10 ));
  }
  landingX = int( random(3, moon.length-4))*10; //plataf
  ship = loadImage( "nave2.png" ); // nave
}
void draw() {
  background(255);
  … //desenha grelha
  drawMoon();
  drawLandingZone(); //plataforma
  drawShip(); // nave
}
```



```
void drawMoon() {
  …
}
//plataforma
void drawLandingZone() {
   …
}
// nave
void drawShip() {
  pushMatrix(); // para afetar só imagem
  translate(150, 100);
  rotate(PI/6);
  image( ship, -ship.width/2, -ship.height/2, ship.width, ship.height
);
  popMatrix(); // para afetar só imagem
}
```

Lua, plataforma, nave, ater

```
int[] moon;
int landingX = 0; // plataforma
PImage ship; // nave
// aterrar
PVector pos = new PVector( 150, 20 );
PVector speed = new PVector( 0, 0 );
PVector g = new PVector( 0, 1.622 );
// aterrar, angulo e acelaração
float a = 0;
float acc = 0;

void setup() {
…
}
void draw() {
  …
  drawMoon();
  drawLandingZone(); // plataforma
  drawShip(); // nave
  update(); // aterrar
}
```



```
void drawMoon() { … }
void drawLandingZone() { … }

void drawShip() {
  pushMatrix();
  translate(pos.x, pos.y);  //translate(150, 100);
  rotate (a); //rotate(PI/6);
  // chamas
  noFill();
  for ( int i=4; i >= 0; i--) {
    stroke(255, i*50, 0);
    fill(255, i*50, 20);
    ellipse( 0, 30, min(1, acc*10) *i*4, min(1, acc*10)* i*10);
  } //chamas
  image( ship, -ship.width/2, -ship.height/2, ship.width, ship.height );
  popMatrix();}
```

Lua, plataforma, nave, aterrar (2)

```
// aterrar
void update() {
  // velocidade e posição da nave influenciadas pela aceleração e rotação
  PVector f = new PVector( cos( a+PI/2 ) * -acc, sin( a+PI/2 ) * -acc );
  if ( acc > 0 ) {
    acc = acc * 0.95;  // para reduzir a aceleração quando não se carrega na tecla
  }
// influência da gravidade
  PVector gDelta = new PVector( g.x / frameRate, g.y / frameRate);
  speed.add( gDelta );
  speed.add( f );  // influência da aceleração e rotação
  pos.add( speed );
// testa se chegou à superfície da lua e para a nave
  if ( pos.x > landingX - 40 && pos.x < landingX + 40 && pos.y > height-50 - ship.height/2 ) {
    pos.y = height - 50 - ship.height/2;
  } else if (pos.y > height - 20 - ship.height/2 ) {
    pos.y = height - 20 - ship.height/2;
  }
}
```

```
// aterrar, teclas de controlo
void keyPressed() {
  if ( keyCode == LEFT ) {
    a = a - 0.1;
  }
  if ( keyCode == RIGHT ) {
    a = a + 0.1;
  }
  if ( keyCode == UP ) {
    acc = acc + 0.04;
    acc = max( acc, 1/frameRate);
  }
}
```