

CÍRCULO REBOTANDO

Processing

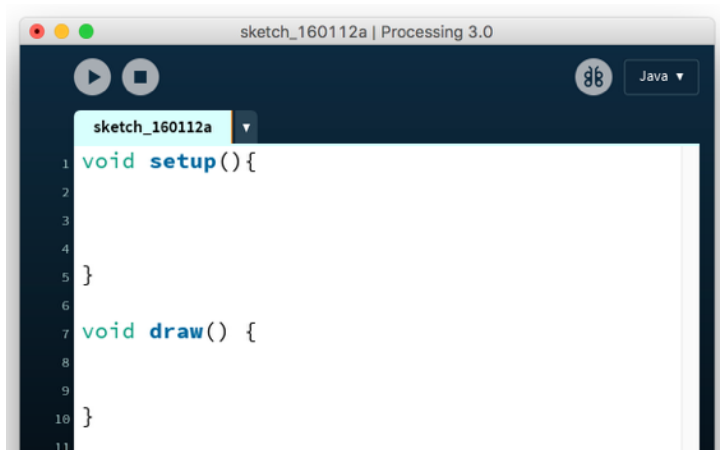
Lenguaje de programación basado en Java y C++ pensado para diseño digital e iniciación a la programación.

IDE

Son las iniciales de Entorno de Desarrollo Integrado, una aplicación informática para facilitarle al programador el desarrollo de software. Normalmente consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador. Algunos IDE contienen un compilador y/o un intérprete.

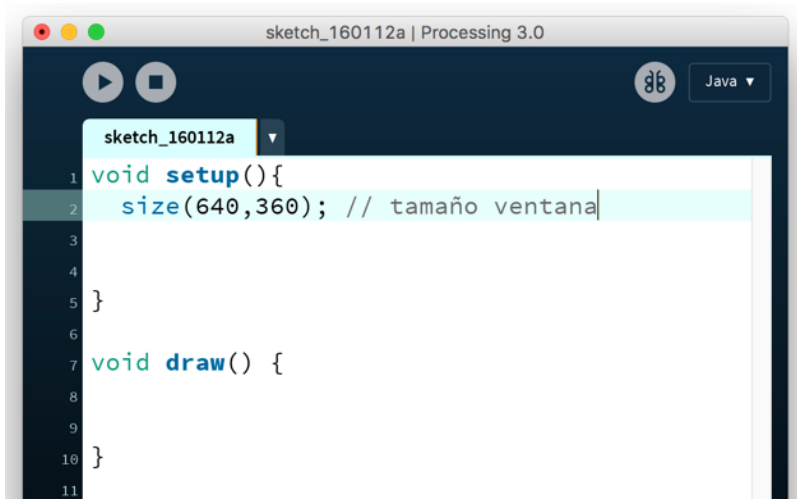
1) Empezamos

Ejecutamos el IDE de Processing y comenzamos nuestro programa con las funciones `setup` y `draw`:



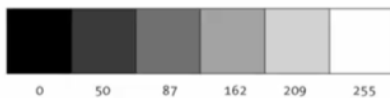
2) Ventana

Establecemos el tamaño en pixeles de nuestra ventana de trabajo con la orden: `size`(tamaño eje x, tamaño eje y);



Color

Modo Escala de grises



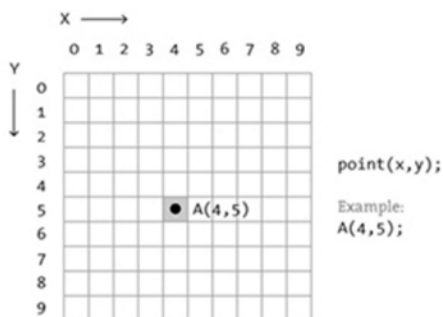
```
background(255); //color de fondo blanco
```

Modo RGB (Rojo, Verde, Azul)

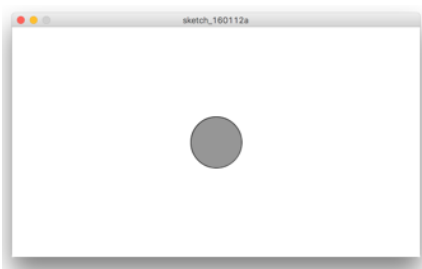


```
background(255,0,0); //color de fondo rojo
```

Coordenadas



- Ojo, las x aumentan de izquierda a derecha, mientras que las y lo hacen de arriba a abajo.



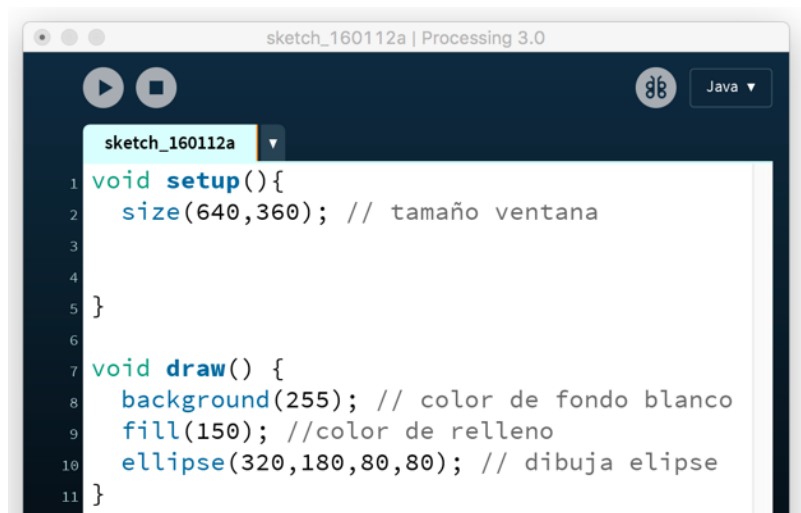
3) Color de fondo

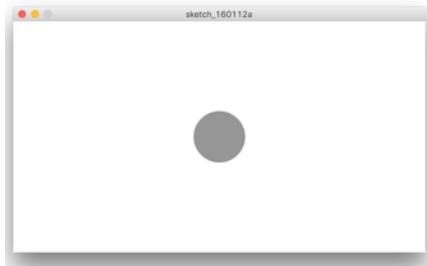
Con la orden [background](#)(código de color); establecemos el color de fondo de nuestra ventana. Si ponemos un valor entre 0 y 255 trabajaremos en escala de grises, si ponemos tres números trabajaremos en color modo RGB ([video: color en Processing](#)):



4) Círculo

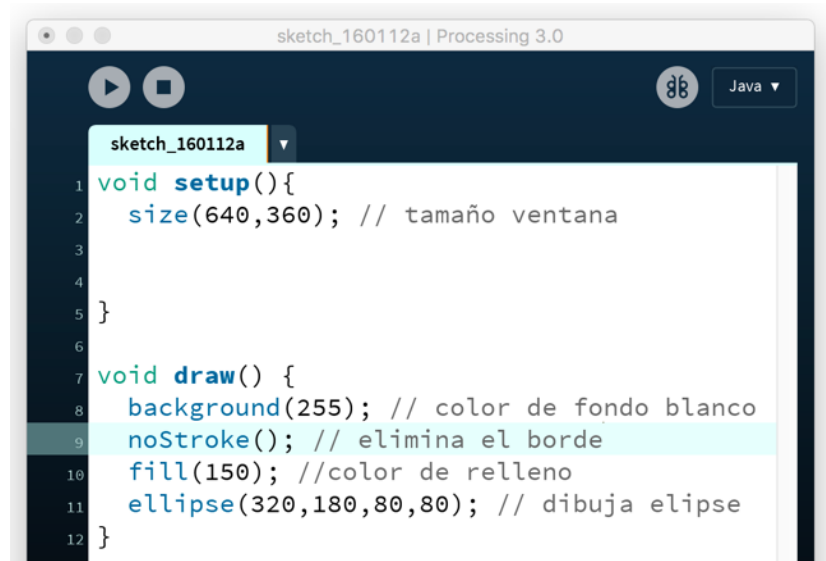
Dibujamos un círculo en la posición 320 en el eje X y 180 en el eje Y (en el medio de la pantalla) con un diámetro de 80 px con la orden [ellipse](#). Con la orden [fill](#) seleccionamos un color para el relleno de nuestro círculo:





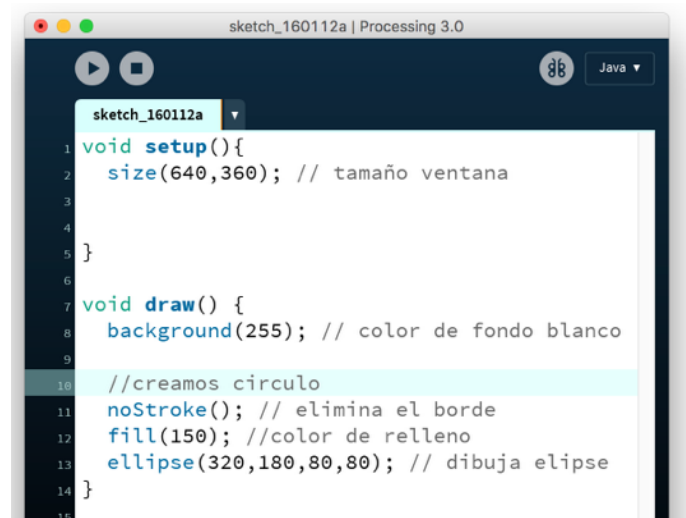
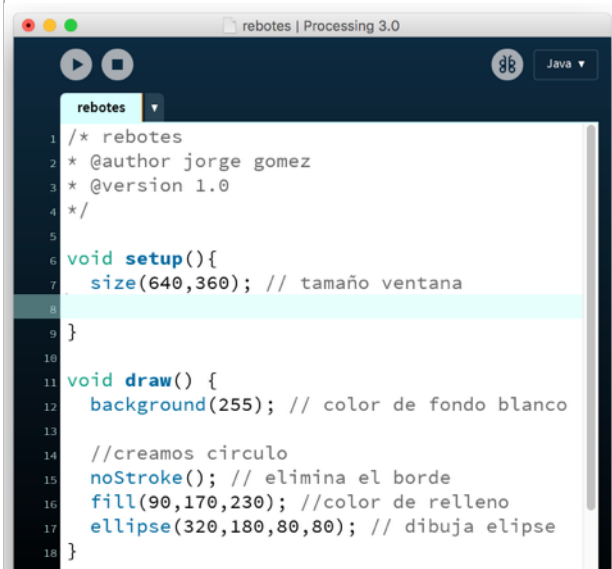
5) Sin borde

Como queremos eliminar el borde negro del círculo añadimos la orden `noStroke()`;



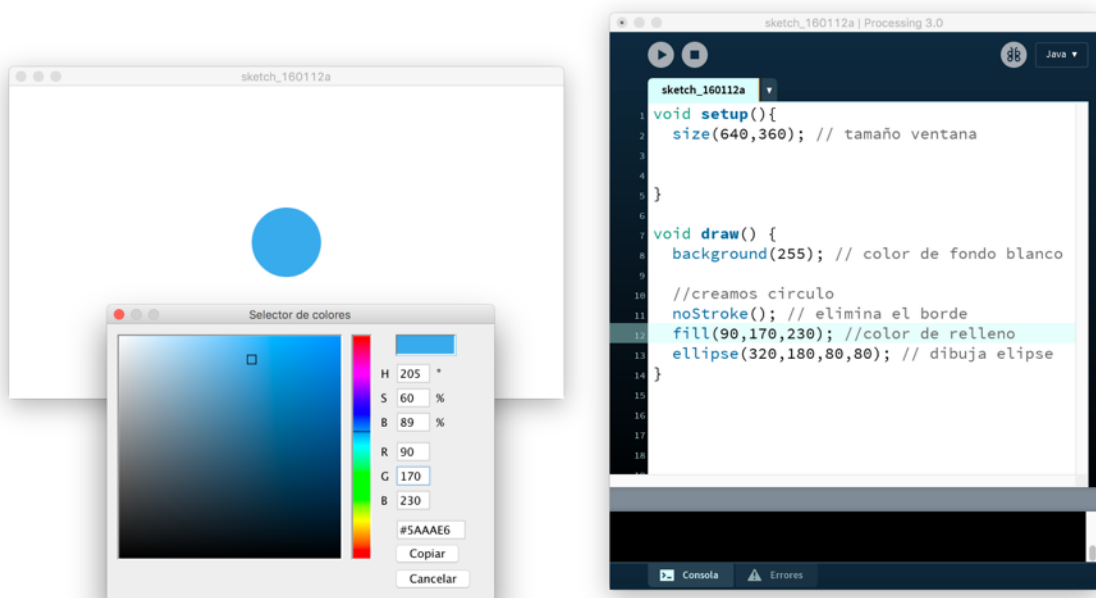
6) Con comentarios

Mejoramos nuestro programa añadiendo comentarios, tanto en una línea como en varias:



7) Selector de colores

Cambiamos el color de relleno de nuestro círculo usando la herramienta de selector de colores de Processing:



variables

Para declarar las variables debemos indicar su nombre y también el tipo de dato que tendrán. Los tipos de datos primitivos que admite Processing son:

[int](#) (números enteros)

[byte](#) (enteros pequeños)

[long](#) (enteros grandes)

[double](#) (decimal)

[float](#) (decimal)

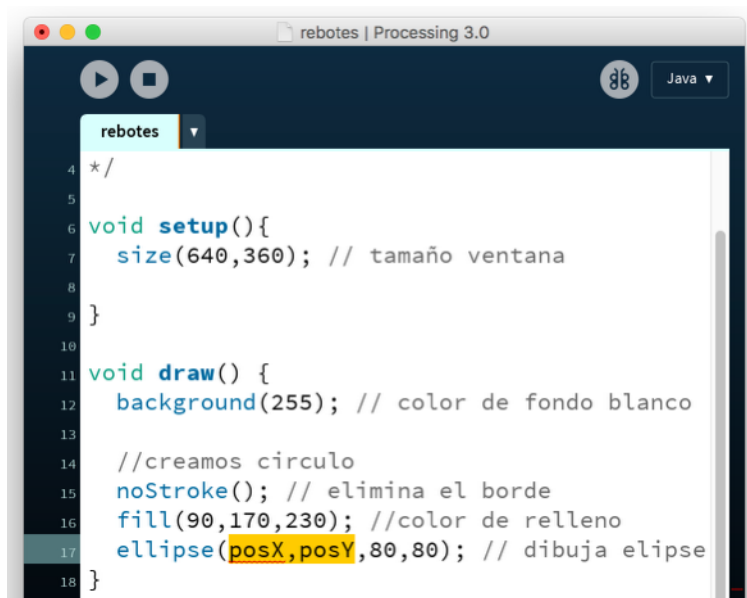
[char](#) (caracteres)

[boolean](#) (true / false)

[color](#) (valor de colores)

8) Variables

Ahora haremos un cambio, sustituimos los valores 320 y 180, la posición X y Y del círculo, por los caracteres posX y posY respectivamente. Estamos creando unas [variables](#).

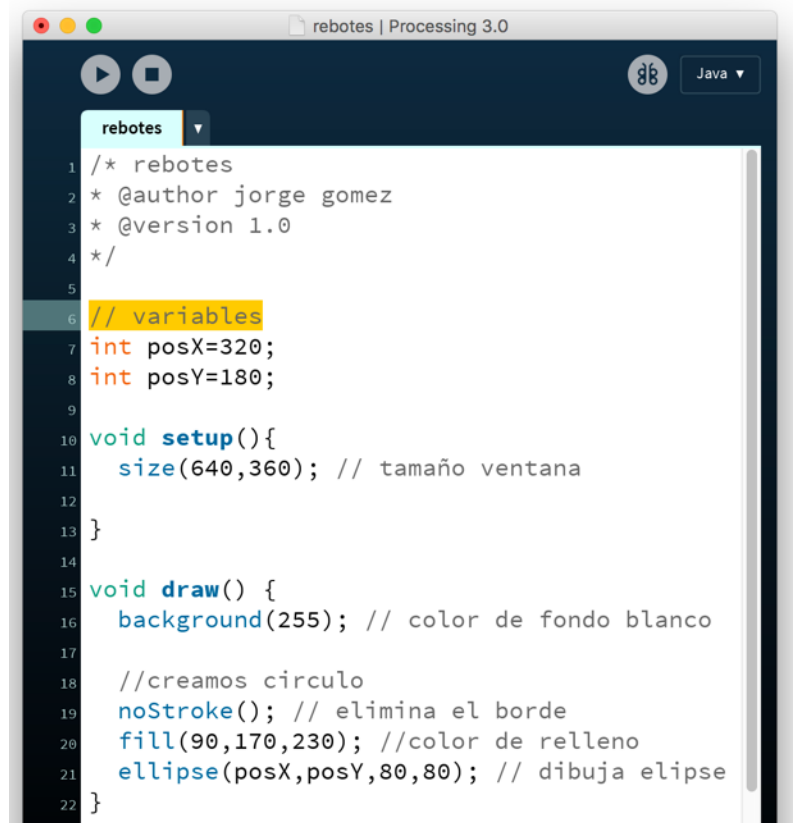


Nombre variables

Una recomendación que se usa en Java es que el nombre de la variable vaya en minúsculas y haga referencia al valor que almacena.

9) Declarar las variables

Pero para que las variables existan tenemos que crearlas y darles un valor inicial. Esto lo haremos al principio del programa, antes del setup.



```

rebotes | Processing 3.0
1  /* rebotes
2  * @author jorge gomez
3  * @version 1.0
4  */
5
6  // variables
7  int posX=320;
8  int posY=180;
9
10 void setup(){
11   size(640,360); // tamaño ventana
12 }
13
14
15 void draw() {
16   background(255); // color de fondo blanco
17
18   //creamos circulo
19   noStroke(); // elimina el borde
20   fill(90,170,230); //color de relleno
21   ellipse(posX,posY,80,80); // dibuja elipse
22 }
  
```

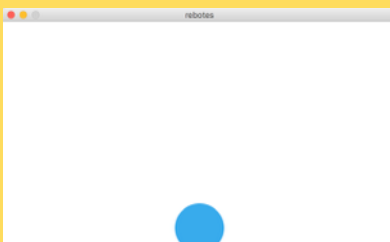
10) Declarar las variables

Si ejecutásemos el programa comprobaríamos que no ha cambiado nada, simplemente modificamos el modo de introducir los datos a través de variables. La ventaja es que ahora podemos modificar el valor de posX y posY de manera que nos da la sensación de movimiento, ya que el círculo se desplaza por la ventana del programa. En este caso añadiremos un incremento tipo **a=a+1**; o lo que es lo mismo **a++**; o incluso también **a+=1**;

Operadores matemáticos

Los operadores matemáticos en Processing son:

+	suma
-	resta
*	producto
/	cociente
%	resto cociente

Captura del círculo bajando

El código quedaría:

```
rebotes | Processing 3.0
3 * @version 1.0
4 */
5
6 // variables
7 int posX=320;
8 int posY=180;
9
10 void setup(){
11   size(640,360); // tamaño ventana
12 }
13
14
15 void draw() {
16   background(255); // color de fondo blanco
17
18   //creamos circulo
19   noStroke(); // elimina el borde
20   fill(90,170,230); //color de relleno
21   ellipse(posX,posY,80,80); // dibuja elipse
22   posY=posY+1; // incrementa el valor en 1
23 }
24
```

II) Declarar las variables

Vamos a crear otras variables para que así podamos variar la velocidad de caída, tanto en valor como en dirección. Las variables velX y velY serán las encargadas de esto.

```
rebotes | Processing 3.0
4 */
5
6 // variables
7 int posX=320;
8 int posY=180;
9 int velX=1; //velocidad eje x
10 int velY=1; //velocidad eje y
11
12 void setup(){
13   size(640,360); // tamaño ventana
14 }
15
16
17 void draw() {
18   background(255); // color de fondo blanco
19
20   //creamos circulo
21   noStroke(); // elimina el borde
22   fill(90,170,230); //color de relleno
23   ellipse(posX,posY,80,80); // dibuja elipse
24   posY=posY+velY; // incrementa el valor en 1
25 }
26
```

Operadores relacionales

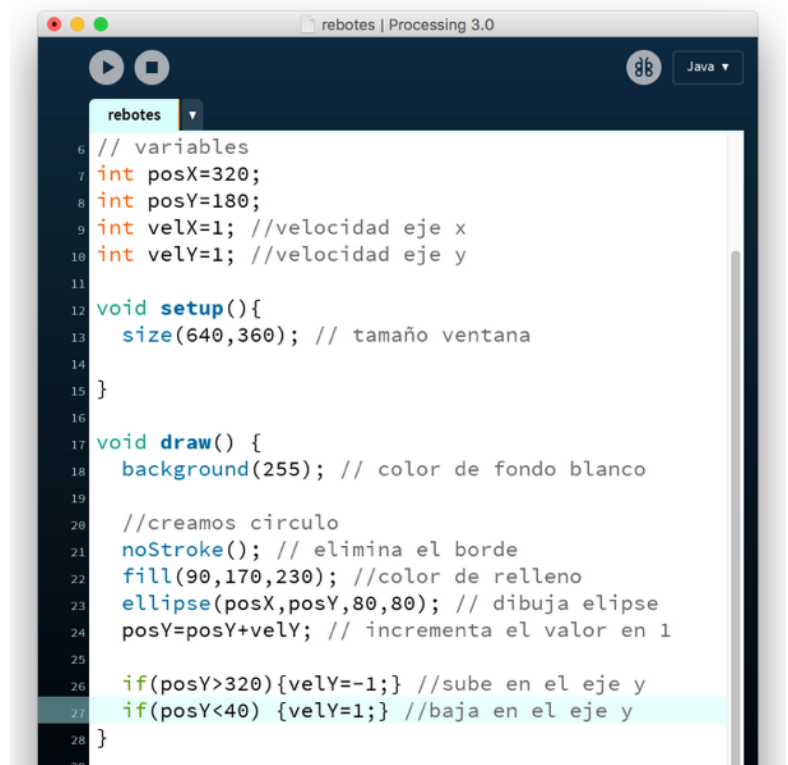
Al utilizar la orden if tenemos que usar [operadores relacionales](#), que en Processing son:

>	mayor que
>=	mayor o igual que
<	menor que
<=	menor o igual
==	exactamente igual
!=	diferente a

12) Condicional if

Si ejecutamos el programa vemos que todo funciona igual, pero ahora añadiremos una orden condicional [if](#) para que el círculo cambie de dirección cuando llegue a la parte baja de nuestra ventana. Esta sería la orden:

```
if(condición) {consecuencia;}
```



Si ejecutamos veremos como el círculo sube y baja sin salirse de la ventana del programa.

13) Más variables...

Ahora haremos lo mismo para el eje X y el círculo rebotará contra las cuatro paredes de nuestra ventana de programa.

En este caso nos falta añadir las instrucciones if para el movimiento en el eje x.

Operadores lógicos

Los operadores lógicos de Processing son:

	función O / OR
&&	función Y / AND
!	función NO / NOT

Por lo que quedaría:

```
rebotes | Processing 3.0

int posX=320;
int posY=180;
int velX=1; //velocidad eje x
int velY=1; //velocidad eje y

void setup(){
  size(640,360); // tamaño ventana
}

void draw() {
  background(255); // color de fondo blanco

  //creamos circulo
  noStroke(); // elimina el borde
  fill(90,170,230); //color de relleno
  ellipse(posX,posY,80,80); // dibuja elipse
  posY=posY+velY; // incrementa el valor en 1
  posX=posX+velX;

  if(posY>320){velY=-1;} //sube en el eje y
  if(posY<40) {velY=1;} //baja en el eje y
  if(posX>600){velX=-1;} //va a la izquierda
  if(posX<40) {velX=1;} //va a la derecha
}
```

14) Mejoras

Podemos mejorar el código anterior cambiándolo tal como aparece en la imagen de la izquierda. Por lo que ahora podemos usar un operador lógico “o” en la instrucción if que nos permite reducir el número de líneas del código:

```
20 //creamos circulo
21 noStroke(); // elimina el borde
22 fill(90,170,230); //color de relleno
23 ellipse(posX,posY,80,80); // dibuja elipse
24 posY=posY+velY; // incrementa el valor en 1
25 posX=posX+velX;
26
27 if(posY>320){velY=-velY;} //sube en el eje y
28 if(posY<40) {velY=-velY;} //baja en el eje y
29 if(posX>600){velX=-velX;} //va a la izquierda
30 if(posX<40) {velX=-velX;} //va a la derecha
31 }
```



```
17 void draw() {
18   background(255); // color de fondo blanco
19
20   //creamos circulo
21   noStroke(); // elimina el borde
22   fill(90,170,230); //color de relleno
23   ellipse(posX,posY,80,80); // dibuja elipse
24   posY=posY+velY; // incrementa el valor en 1
25   posX=posX+velX;
26
27   if(posY>320 || posY<40){velY=-velY;}
28   if(posX>600 || posX<40){velX=-velX;}
29
30 }
```

Para descargar el archivo de la práctica: [Pulsa aquí](#)