

BLOQUE **Código**

1

CONCEPTOS

PROYECTOS

En este primer bloque vas a hacer una serpiente roja interactiva, un reloj con post-it y un juego inspirado en Newton. Aprenderás lo básico de programación así como algo de la teoría detrás de los gráficos en las pantallas. Esta segunda parte es importante debido a que, aunque este curso trate sobre programación, el objetivo es siempre crear algo útil y divertido. Para hacer uso de la pantalla en tus proyectos, es bueno saber cómo funciona. Cuando este bloque esté terminado, estaremos encantados de ver qué otros juegos o proyectos interactivos serás capaz de crear.

Processing



Los códigos de programación son comandos comprensibles para los humanos. El código se transforma en los programas que los computadores entienden. Los computadores sólo entienden en realidad ceros y unos, por lo que se necesita una manera de traducir nuestro código humano al código de los computadores. Este es el motivo por el que escribimos el código en un editor de texto con propiedades especiales. Este editor es el software de desarrollo llamado Entorno de Desarrollo Integrado, abreviado por sus siglas del inglés IDE (Integrated Development Environment), el cual es capaz de traducir el código al lenguaje máquina. Cuando el IDE transforma nuestro código al lenguaje máquina, es lo que se llama compilar.

Cada ordenador, smartphone, servidor de internet, ascensor, horno – actualmente cualquier dispositivo con inteligencia digital – tiene diferentes capacidades y por lo tanto requiere de un lenguaje máquina algo diferente al resto. Existe un lenguaje de programación llamado Java que, una vez compilado, es capaz de utilizarse fácilmente en diferentes dispositivos. Hay muchas maneras diferentes de programar en Java IDE, pero nosotros nos centraremos en utilizar Processing.

Processing es código abierto y software libre, esto significa que se puede descargar de Internet, instalarlo sin coste alguno, y modificar su código. Processing funciona en cualquier tipo de ordenador de escritorio o portátil en los sistemas operativos: Windows, Mac OSX y Linux.

El IDE de Processing es muy sencillo, hay una barra de herramientas que puedes utilizar para:

-  Poner en marcha un programa
-  Detener un programa
-  Crear un nuevo programa
-  Abrir un programa
-  Almacenar el programa en el disco duro del ordenador
-  Exportar el programa



Processing crea programas que se ejecutan directamente dentro de tu ordenador. También puedes exportarlos y enviarlos a los ordenadores de otras personas. Los programas, una vez exportados en código máquina, se llaman aplicaciones. Puedes exportar el mismo programa para que se ejecute en un ordenador, en un teléfono, o en un sitio web.

Para hacer tus propios programas, lo único que necesitas saber es cómo utilizar un teclado y cómo funciona la pantalla del ordenador. Esto te será útil cuando escribas tus propios programas interactivos.

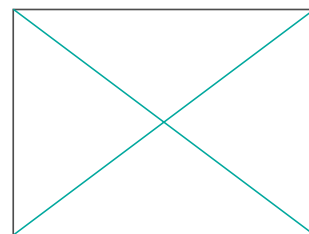
Pantallas y píxeles

Las pantallas se utilizan para representar imágenes. Esas imágenes pueden ser dinámicas, como en las películas, o estáticas, como en las fotografías. Se puede incluir texto, como en los documentos, o solo gráficos. La mayoría de las veces, las imágenes en la pantalla son una mezcla de muchas cosas; incluyen animaciones, textos, estadísticas, etc.

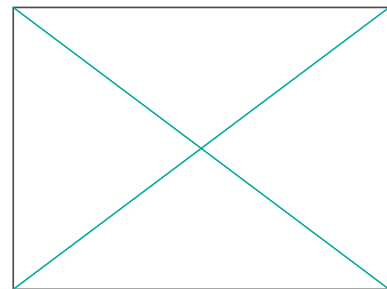
Píxeles – Resolución

Las pantallas están hechas de puntos que llamamos píxeles. Por lo general, cuantos más píxeles tenga una pantalla, mejor será. Expresamos la calidad de la pantalla en función de su cantidad de píxeles, esa cantidad es lo que llamamos resolución. Normalmente, hablamos de una pantalla en función de su anchura y altura. Por ejemplo, muchos proyectores tienen una resolución de 1024×768 píxeles

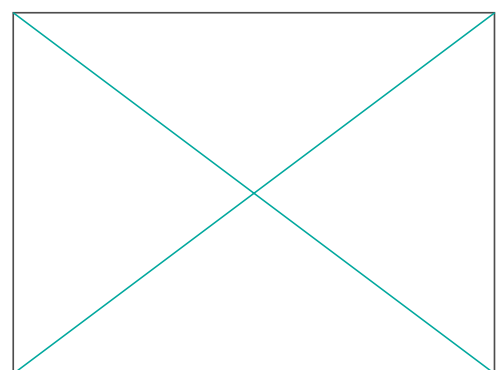
Esto significa que tiene 1024 píxeles de ancho y 768 píxeles de alto.



640 x 480



800 x 600



1024 x 768

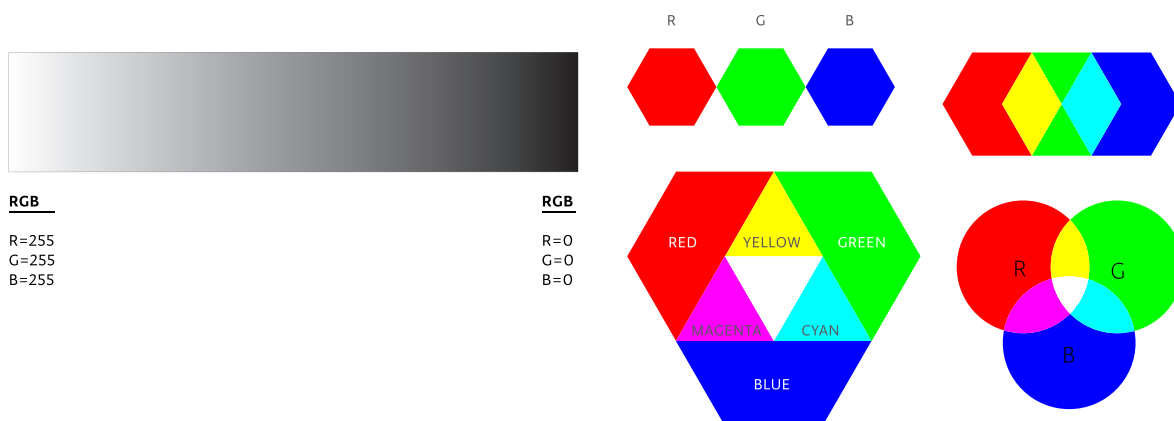
Colores

Cada píxel se puede iluminar de muchos colores diferentes. Una vez más, de cuantos más colores se pueda pintar el píxel, mejor será la pantalla. Los colores en la pantalla del ordenador se expresan como una combinación de tres componentes de color diferentes: rojo (R), verde (G) y azul (B). Cualquier color que representes en una pantalla, se puede hacer mezclando los componentes RGB en diferentes cantidades.

La cantidad de color para cada componente se expresa con un número entre 0 y 255. 0 representa el valor mínimo de un color, por ejemplo, si tiene 0 del componente rojo, significa que ese color no contiene rojo, sin embargo, si tiene 255 del componente rojo, significa que está saturado de color rojo.

Si las coordenadas son 255,255,255 – los píxeles producirán blanco. Si los valores R,G,B son 0,0,0 – los píxeles mostrarán negro. Si gradualmente cambias los valores R,G,B en la misma proporción desde 255,255,255 a 0,0,0, verás que el color cambia de blanco a negro.

La siguiente imagen muestra cómo el color de un píxel se funde de blanco a negro y cómo cambian los valores de R, G, y B:



Coordenadas

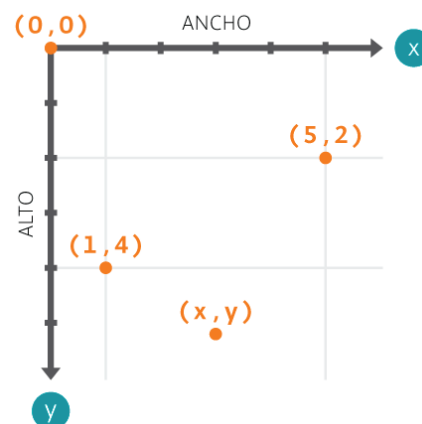
Puedes escribir programas que cambien los píxeles de la pantalla. Puedes incluso cambiar un solo píxel. La acción de acceso a un solo píxel en la pantalla es lo que llamamos direccionar un píxel. La ubicación de un píxel se determina mediante coordenadas.

Si recuerdas lo que aprendiste en clase de matemáticas y física, puedes dibujar gráficos, puntos, líneas y curvas usando sus coordenadas. Esta misma idea se aplica a la pantalla del ordenador, la única diferencia, es que el llamado origen de coordenadas de la pantalla del ordenador es la esquina superior izquierda.

Cada píxel tiene 2 coordenadas:

La coordenada horizontal es la que llamamos coordenada X (anchura, o en inglés width).

La coordenada vertical, es la que llamamos coordenada Y (altura, o en inglés height).



Línea

La mejor manera de aprender acerca de la programación es escribiendo tus propios programas. Vamos a empezar por hacer el programa lo más corto posible usando Processing. Será un programa para dibujar una línea. Esto te va a ayudar a entender cómo funciona.

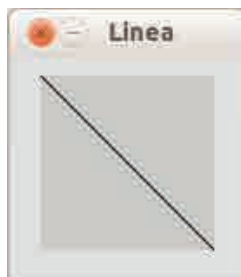


Abre el IDE y haz click en el icono para crear un nuevo programa, esto te mostrará un nuevo programa vacío. Escribe el siguiente código:

```
1 line(0, 0, 100, 100);
```



Ejecuta el programa apretando el botón *ejecutar*, que es el primero en la barra de herramientas. Esto es lo que deberías ver ahora por pantalla:



Sólo utilizamos un único comando aquí:

line(x1, y1, x2, y2): Dibuja una línea desde la coordenada x1,y1 hasta la x2,y2. En este ejemplo, desde 0,0 – esquina superior izquierda, hasta 100,100 – esquina inferior derecha. Esto es debido a que la ventana es 100 por 100 píxeles.

Lo que también necesitas saber es:

- Debes terminar cada línea de código con un punto y coma, ';'. Si te olvidas, el código no podrá ser compilado ni ejecutado.
- Por defecto, las ventanas de Processing son de 100×100 píxeles con un fondo gris. El color por defecto para dibujar es el negro. Pero por supuesto, todo esto se puede cambiar.

Vamos a escribir un programa que dibuje dos líneas de diferentes colores sobre un fondo blanco:

```
1 background(255);  
2 line(0, 0, 100, 100);  
3 stroke(0, 0, 255);  
4 line(0, 100, 100, 0);
```

Los nuevos comandos utilizados aquí son:

- **background(gray)**: Establece el color de fondo desde 0 – negro, hasta 255 – blanco. También puedes utilizar **background(red, green, blue)** para establecer el color que tú quieras.
- **stroke(red, green, blue)**: Establece el color de la línea. Cada valor de color puede ser desde 0 hasta 255. En este caso, la línea es de color azul puesto que red=0, green=0 y blue=255.



Cuando ejecutes el programa, lo que deberías ver por pantalla es:

Sólo utilizamos un único comando aquí:

- **line(x1, y1, x2, y2)**: Dibuja una línea desde la coordenada x1,y1 hasta la x2,y2. En este ejemplo, desde 0,0 – esquina superior izquierda, hasta 100,100 – esquina inferior derecha. Esto es debido a que la ventana es 100 por 100 píxeles.

Lo que también necesitas saber es:

- Debes terminar cada línea de código con un punto y coma, ';'. Si te olvidas, el código no podrá ser compilado ni ejecutado.
- Por defecto, las ventanas de Processing son de 100×100 píxeles con un fondo gris. El color por defecto para dibujar es el negro. Pero por supuesto, todo esto se puede cambiar.

Tus programas se ejecutarán siempre de arriba a abajo línea por línea. Este programa empieza estableciendo el color de fondo en blanco. Luego dibuja la línea negra igual que en el ejemplo anterior, desde la coordenada 0,0 hasta la 100,100. En tercer lugar, establece el color de línea a azul, justo antes de dibujar esta segunda línea desde la coordenada 0,100 – esquina inferior izquierda, hasta la 100,0 – esquina superior derecha.



No olvides guardar tu programa en el disco duro o en una memoria USB si quieres llevártelo a casa.

¡Sigue experimentando!

- Añade más líneas de diferentes colores y cambia el color de fondo.
- Cambia el orden del programa. ¿Qué ocurre si, por ejemplo, cambias el color de fondo al final?

Variables

Las variables son algo que usas todo el tiempo en programación. Son como un contenedor para diferentes tipos de datos. Para cada variable, necesitas especificar qué tipo de datos contendrán, cuál es el nombre de la variable y qué valor se le asigna.

Piensa en ellas como en botes. Digamos que tienes dos botes, uno para galletas y otro para palabras, estos son los tipos de datos. Ahora hay que darles un nombre a cada bote; cookieJar (*bote de galletas*) y jarOfWord (*bote de palabras*). Ahora tú decides que poner en cada bote. En el cookieJar pones una galleta de doble chocolate y en el jarOfWord decides poner "Arduino". Ahora cada bote tiene un valor.

Puedes cambiar el contenido de los botes, su valor, en cualquier momento, pero siempre y cuando sea del mismo tipo. Por ejemplo, puedes cambiar la galleta de doble chocolate por una oreo y "Arduino" por "spaceinvader".



Para hacerlo más claro, vamos a escribir un programa utilizando variables. Escribiremos un programa que dibuje, de nuevo, dos líneas, pero esta vez utilizaremos estas variables:

```
1  int value1 = 0;
2  int value2 = 100;
3  line(value1, value1, value2, value2);
4  line(value1, value2, value2, value1);
```

Nuevos comandos:

- **int variableName = value:** Crea una variable del tipo integer, un número entero. Puedes llamarla con cualquier nombre de tu elección, pero para hacerlo más sencillo para el futuro (probablemente tú o alguien más leerán el código y querrán entender lo que se hizo en el pasado) asegúrate que el nombre va acorde con el contexto en el que se usa la variable. **value** es el valor que decides darle al contenido de tu variable.

Lo que hace el programa es dibujar una línea desde la coordenada (0,0) hasta la (100,100) y luego dibuja una segunda línea desde la coordenada (0,100) hasta la (100,0). Puesto que **value1** contiene el valor 0, en cualquier lugar del código donde escribamos **value1** significará 0. Esto significa que si quieres cambiar la cruz que hemos dibujado, basta con cambiar el valor de la variable en lugar de cambiar el valor en los ocho lugares diferentes dentro de **line()**. Pruébalo tu mismo y verás.

Tipos de datos

Los tipos de datos más comunes que utilizarás son los siguientes:

- **int:** Número entero, p.ej., 2, 99 o 532.
- **float:** Número decimal, p.ej., 2.76, 8.211 o 900.3.
- **boolean:** Puede ser verdadero o falso.
- **char:** Un carácter, p.ej. 'r', '2' o '%'.
- **String:** Una secuencia de caracteres, p.ej. "hola", "¡Me encanta programar!" o "&%!@α".

Processing incluye algunas variables de sistema para hacerlas más accesibles dentro de tus programas. Un ejemplo es **width** y **height**. Estas variables devuelven la anchura (width) y altura (height) de la ventana de tu programa. Éste es un pequeño ejemplo:

```
1  size(400, 200);
2  ellipse(width/2, height/2, width, height);
```

Comandos nuevos:

- `size(width, height)`: Establece el tamaño de la ventana en píxeles.
- `ellipse(x, y, x diameter, y diameter)`: dibuja un elipse centrado en las coordenadas `x` e `y`. El tamaño se establece con `x diameter` e `y diameter`. Cuando estos dos parámetros son iguales, el resultado es un círculo.
- `width`: la anchura de la ventana de programa.
- `height`: la altura de la ventana de programa.



Este programa empieza estableciendo el tamaño de ventana. Luego dibuja la elipse en el medio de la ventana. No importa el tamaño de la venta, el elipse la llenará siempre por completo. Intenta cambiar el tamaño de la ventana.

Setup y draw

Los tipos de programas que hemos hecho hasta ahora son los llamados programas estáticos. Esto significa que nunca cambian. Se ejecutan una única vez y cuando llegan a la última línea de código, se paran. Si queremos que un programa sea interactivo, tenemos que habilitar la entrada de datos continuamente mientras el programa se ejecuta. Esto sólo es posible si la ejecución es continua.

Con Processing, puedes crear programas que se ejecuten continuamente utilizando la función `draw()`. Esta función repetirá el bloque de código una y otra vez hasta que el programa se pare. Sin embargo, no todo el código escrito necesita ser repetido continuamente. Para el código que sólo necesita ser ejecutado una única vez, debes usar la función llamada `setup()`.

Vamos a escribir de nuevo un programa que dibuje una línea, pero esta vez utilizando las nuevas funciones:

```
1 void setup() {
2   size(300, 300);
3 }
4
5 void draw() {
6   line(0, 0, width, height);
7 }
```

Nuevos comandos:

setup(): El código dentro de las llaves se ejecuta una única vez cuando el programa empieza. Sólo necesitamos establecer el tamaño de la ventana una vez, al principio del programa.

draw(): El código entre llaves se ejecuta una y otra vez. Se ejecuta línea por línea, de arriba a abajo hasta la última línea, donde vuelve a empezar desde el principio.

Este programa establece el tamaño de ventana a 300 x 300 píxeles y luego dibuja una línea sobre toda la ventana una y otra vez. En este momento, todavía no se puede apreciar que esto está pasando una y otra vez, así que vamos a escribir otro programa para hacerlo más visible.

```
1 void setup() {  
2   size(300,300);  
3 }  
4  
5 void draw() {  
6   line(0, 0, mouseX, mouseY);  
7 }
```

Nuevos comandos:

- **mouseX**: La coordenada X del puntero del ratón.
- **mouseY**: La coordenada Y del puntero del ratón.

Este programa te permite interactuar con la pantalla mientras mueves el ratón, como muestra la siguiente imagen. Esto es porque cada vez que el código dentro de **draw()** se ejecuta, una nueva línea se dibuja. El segundo punto de la línea cambia dependiendo de la posición del puntero del ratón.



Como puedes ver, se deja un rastro de líneas por donde vas moviendo el ratón. Esto es porque cada línea dibujada no es eliminada ni se dibuja nada más por encima. ¿Qué crees que ocurrirá cuando ejecutes el siguiente programa?

```
1 void setup() {  
2   size(300,300);  
3 }  
4  
5 void draw() {  
6   background(255);  
7   line(0, 0, mouseX, mouseY);  
8 }
```

Hemos añadido una línea de código al principio de `draw()`, esto sirve para establecer el color de fondo a blanco. Como puedes ver, este programa solo muestra una línea todo el tiempo. Esto se debe a que, como hemos explicado antes, el código dentro de `draw()` se ejecuta una y otra vez, línea por línea de arriba a abajo. Por tanto, la primera cosa que pasa cada vez que `draw()` se ejecuta es que se crea el fondo blanco, sobre escribiendo todo lo anterior. Después una nueva línea es dibujada, y luego el fondo se vuelve a poner en blanco... Eso se repetirá una y otra vez por siempre.

Serpiente roja

En este miniproyecto vas a aprender a programar un objeto que se mueve en la pantalla, pero que además deja el trazado del movimiento del ratón. Iremos escribiendo el código paso por paso, añadiendo nuevas funcionalidades en cada uno.

Dibuja un círculo

Empezaremos dibujando un círculo rojo.

```
1  /*
2   * Red Snake (Serpiente Roja)
3   *
4   * Programa para crear una serpiente roja moviendose por la pantalla
5   *
6   * Paso 1:
7   *   - crea un programa en una ventana de 400x400 píxeles
8   *   - dibuja un círculo rojo de 30 píxeles de diametro
9   *   - haz que el dibujo no tenga borde, usa el comando 'noStroke()'
10  *
11  * (c) 2013 D. Cuartielles, Arduino Verktstad, Suecia
12  */
13
14 void setup() {
15   size(400, 400);
16 }
17
18 void draw() {
19   noStroke();           // Dibuja formas sin 'borde'
20   fill(255, 0, 0);      // Haz que las figuras sean de color rojo
21   ellipse(100, 100, 30, 30); // Círculo de 30 píxeles de diametro
22 }
```

Nuevos comandos:

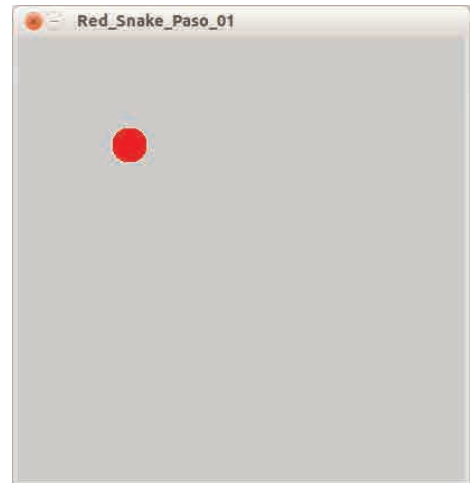
- **noStroke()**: Este comando es usado para no dibujar el contorno de las siguientes figuras
- **fill(red, green, blue)**: Establece el color utilizado para rellenar las siguientes figuras. En este ejemplo

rellenará el círculo de color rojo.

Veamos los otros comandos utilizados:

- `size(ancho, alto)`: establece el tamaño de la ventana del programa en píxeles.
- `ellipse(x, y, diametro x, diametro y)`: dibuja un elipse con centro en x,y. El tamaño se establece con diametro x y diametro y. Cuando estos dos parámetros son iguales, el resultado es un círculo.

Lo que el programa hace es primero establecer el tamaño de la ventana a 400 x 400 píxeles. Nosotros decidimos no utilizar contornos y utilizar el color rojo para el relleno. Por último, dibuja un círculo en las coordenadas 100,100 con un diámetro de 30 píxeles.

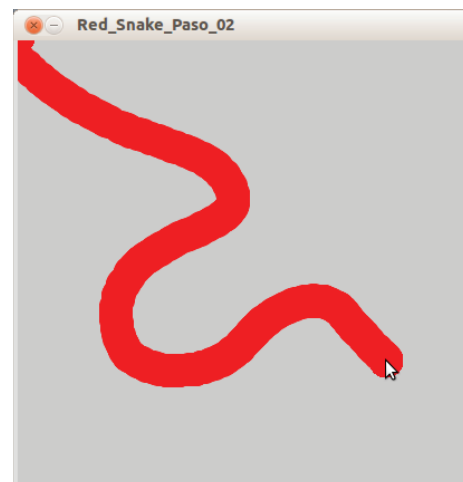


Haz que el círculo se mueva

En este paso haremos que el círculo se mueva con el ratón, dejando un rastro en la pantalla.

```
1  /*
2   * Red Snake (Serpiente Roja)
3   *
4   * Programa para crear una serpiente roja moviendose por la pantalla
5   *
6   * Paso 2:
7   * - haz que el circulo se mueva en funcion de las coordenadas del
8   *   raton (usa 'mouseX' y 'mouseY')
9   *
10  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
11  */
12
13 void setup() {
14   size(400, 400);
15 }
16
17 void draw() {
18   noStroke();
19   fill(255, 0, 0);
20   ellipse(mouseX, mouseY, 30, 30); // Circulo de en funcion de las coordenadas del raton
21 }
```

Todo lo que hemos hecho aquí ha sido reemplazar las coordenadas de la elipse con mouseX y mouseY. Esto hace que el círculo siempre siga al puntero del ratón. Al no redibujar el fondo de la pantalla, hacemos que el círculo deje su trazo.

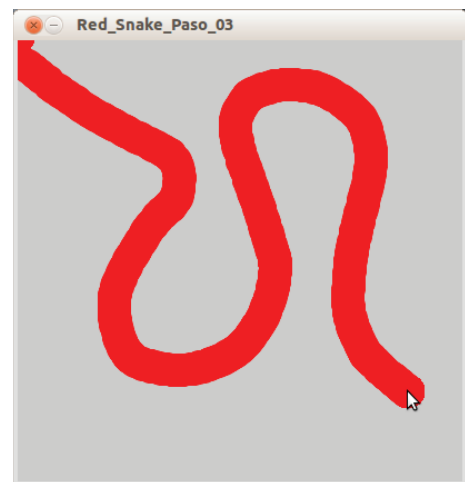


Cambia gradualmente el color

Queremos ahora que el color cambie durante el programa, y para hacerlo es más conveniente utilizar una variable para el color en lugar de un número constante. Simplemente declara la variable rojo con la que modificar el valor del color.

```
1  /*
2   * Red Snake (Serpiente Roja)
3   *
4   * Programa para crear una serpiente roja moviendose por la pantalla
5   *
6   * Paso 3:
7   *   - almacena la intensidad de rojo en una variable
8   *
9   * (veras como este programa no cambia en nada frente al Paso 2)
10  *
11  * (c) 2013 D. Cuartielles, Arduino Verktstad, Suecia
12  */
13
14  int rojo = 255;
15
16  void setup() {
17    size(400, 400);
18  }
19
20  void draw() {
21    noStroke();
22    fill(rojo, 0, 0);
23    ellipse(mouseX, mouseY, 30, 30); // Circulo de en funcion de las coordenadas del raton
24  }
```

Fijate que esto no causará ningún cambio en la aplicación, como puedes ver en la siguiente imagen.



Cambia gradualmente el color II

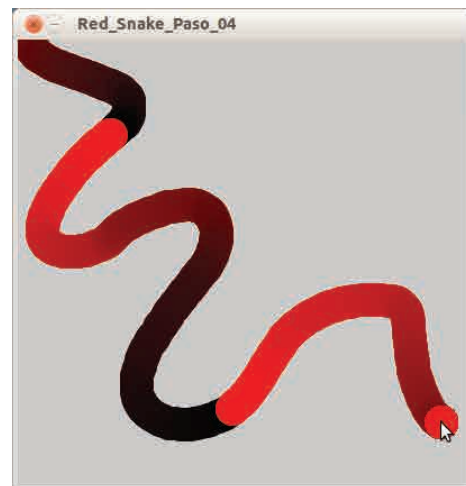
Ahora vamos a cambiar el color. El cambio de color dependiendo del tiempo puede hacerse de diferentes maneras. El color puede cambiar entre 0 y 255 (negro y rojo respectivamente). El siguiente ejemplo muestra como reducir la cantidad de rojo en cada instante que se ejecuta draw().

```
1  /*
2   * Red Snake
3   *
4   * Program to create a red snake moving through the screen
5   *
6   * Step 4:
7   * - chage the level or red with the time
8   *
9   * (c) 2013 D. Cuartielles, Arduino Verktstad, Sweden
10  */
11
12  int rojo = 255;
13
14  void setup() {
15    size(400, 400);
16  }
17
18  void draw() {
19    rojo = rojo - 1;          // Make the red a little darker
20    if (rojo < 0) rojo = 255; // Once it reaches black, make it light red again
21
22    noStroke();
23    fill(rojo, 0, 0);
24    ellipse(mouseX, mouseY, 30, 30);
25  }
```

Nuevos comandos:

- `if( test ){ statements }`: Comprueba si `test` es cierto. En este ejemplo, si rojo es menor que 0. Si es así, los `statements` entre llaves son ejecutados. En este ejemplo, establece `red` a 255 de nuevo. Si por el contrario, `test` es falso, el programa procede a ejecutar el código después de las llaves. Utilizando declaraciones "if" te permite decir al programa que código ejecutar.

Al principio de este programa `red` es 255, pero cada vez que `draw()` se ejecuta, `red` se reduce en una unidad. Esto hace que el color del círculo se haga más y más negro en cada iteración. Cuando `red` ha descendido mucho, tal que es menor que 0, se reinicia a 255. Este es el motivo por el que el color cambia de repente de negro a rojo.



Usa la función `sin()`

Para cambiar los colores más gradualmente, utilizaremos una función sinusoidal que oscile continuamente entre negro y rojo.

```
1  /*
2   * Red Snake (Serpiente Roja)
3   *
4   * Programa para crear una serpiente roja moviendose por la pantalla
5   *
6   * Paso 5:
7   *   - cambia la intensidad del rojo en funcion del tiempo
8   *   - para un mejor efecto visual y evitar que el rojo cambie
9   *     de forma brusca, utiliza una funcion matematica (por ejemplo
10  *     el seno 'sin()')
11  *   - necesitaras una variable para contar el paso del tiempo
12  *   - el '20' en la ecuacion de calculo del rojo determina cada cuanto
13  *     se repite un color
14  *
15  * Nota: la funcion 'sin()' devuelve un numero decimal ('float' o en coma flotante)
16  *       y por eso hay que emplear la funcion 'int()' para poder asignarla a la
17  *       variable que almacena el color rojo
18  *
19  * (c) 2013 D. Cuartielles, Arduino Verktstad, Suecia
20  */
21
```

```

22 int rojo = 255;
23 int tiempo = 0;
24
25 void setup() {
26   size(400, 400);
27 }
28
29 void draw() {
30   tiempo = tiempo + 1; // Incrementa el tiempo en una unidad
31   rojo = int(128 * (1 + sin(tiempo * 2 * PI / frameRate / 20))); // Cambia el rojo,
    repite el color cada 20s
32
33   noStroke();
34   fill(rojo, 0, 0);
35   ellipse(mouseX, mouseY, 30, 30);
36 }

```

Nuevos comandos:

- **sin(angle)**: Esta función se utiliza para calcular el seno de un ángulo. En nuestro caso, no tiene nada que ver con ángulos pero, ¿te acuerdas de la gráfica del seno de la clase de matemáticas? Bien, esa gráfica es la que es útil para nosotros. Siempre que quieras crear una oscilación suave y continua de un movimiento o, en este caso, un cambio de color, `sin()` es muy práctico de usar.
- **PI**: Esto es una variable constante que contiene el valor del número pi.
- **frameRate**: Esto es otra variable que te da el *framerate* (frecuencia de ejecución) del programa.

En este ejemplo, puedes ver como el color cambia gradualmente de forma continua. Esto se debe a que el color se establece por una función senoidal dependiendo de la frecuencia de ejecución del sketch (`frameRate`) y la variable `time` que cuenta las veces que `draw()` se ha ejecutado.



Cambiando de forma

El siguiente paso es cambiar la forma de la serpiente. Esto se consigue fácilmente cambiando el tamaño del círculo, utilizando también una función senoidal.

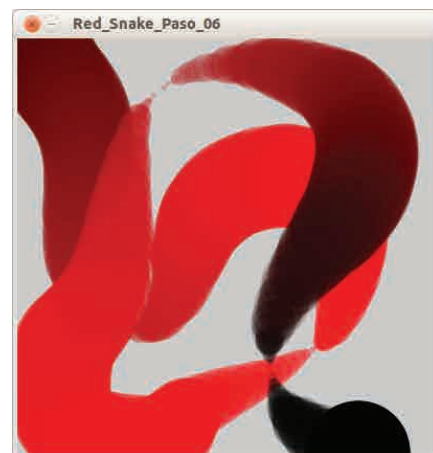
```

1  *
2  * Red Snake (Serpiente Roja)
3  *
4  * Programa para crear una serpiente roja moviendose por la pantalla
5  *
6  * Paso 6:
7  * - cambia el tamaño del círculo en función del tiempo
8  * - al igual que antes, necesitaras crear una variable
9  *   para almacenar el valor del tamaño
10 * - puedes usar la misma función que para el color, pero
11 *   cambiando la frecuencia
12 * - para mejor efecto visual, añade transparencia al color
13 *
14 * (c) 2013 D. Cuartielles, Arduino Verktstad, Suecia
15 */
16
17 int rojo = 255;
18 int tiempo = 0;
19 int diametro = 50;
20
21 void setup() {
22   size(400, 400);
23 }
24
25 void draw() {
26   tiempo = tiempo + 1;
27   rojo = int(128 * (1 + sin(tiempo * 2 * PI / frameRate / 20)));
28   diametro = int(50 * (1 + sin( tiempo * 2 * PI / frameRate / 5))); // Modifica el
   diametro del círculo con el paso del tiempo
29
30   noStroke();
31   fill(rojo, 0, 0, 50); // Añade un 50% de transparencia al color
32   ellipse(mouseX, mouseY, diametro, diametro);
33 }

```

Nuevos comandos:

- `fill(red, green, blue, alpha)`: Hemos añadido un cuarto parámetro `alpha` a la función `fill()`. Esto establece la transparencia del color y su rango va de 0 a 255.



Sigue experimentando

- La modificación más sencilla que puedes hacer es cambiar el color de la serpiente. Puede ser tan fácil como mover la variable `red` al segundo o tercer parámetro de la función `fill()`.
- Puedes también añadir otras variables para cambiar los otros parámetros del color de forma independiente.

Reloj de Post-it

En este miniproyecto vas a crear un reloj donde los números estarán hechos a partir de fotografías de notas adhesivas o Post-it. Para ello, aprenderás a utilizar imágenes en tus programas y almacenarlas en arrays (explicado luego). De nuevo, iremos avanzando paso por paso

Inserta una imagen en tu programa

En primer lugar, vamos a aprender a crear un programa capaz de mostrar una imagen. En Processing hay un tipo de variable (también conocido como clase) llamada `PImage`. Las variables de este tipo almacenan una imagen que podrás cargar de un fichero usando la función `loadImage()`.

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 1:
8   * - crea un programa que muestre una imagen
9   * - para cargar la imagen en tu programa, selecciona una imagen y arrastrala
10  *   sobre el IDE, asi se almacenara en una carpeta accesible a tu programa
11  * - puedes comprobar que se ha creado una carpeta 'data' dentro de la carpeta
12  *   de tu programa, abre la carpeta desde el menu 'Sketch --> Show Sketch Folder'
13  * - las imagenes:
14  *   + son variables del tipo 'PImage'
15  *   + se muestran con la funcion 'image()'
16  *   + se cargan con 'loadImage()'
17  *
18  * Nota: antes de introducir una imagen, asegurate de que no es demasiado grande,
19  *       reducela de tamaño a 400x400 o similar para que encaje en tu pantalla
20  *
21  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
22  */
23
24  PImage im;
25
26  void setup() {
27    size(400, 400); // Hacemos el programa del tamaño de la imagen
```

```

28  im = loadImage("foto.jpg"); // Nombre de la imagen
29  }
30
31  void draw() {
32    image(im, 0, 0); // Para mostrar la imagen en la pantalla, coordenadas 0, 0
33  }

```

Nuevos comandos:

- **PImage**: Un tipo de variable que puede contener una imagen.
- **loadImage(image file)**: Carga la imagen **image file** localizada dentro del directorio actual del sketch. **image file** debe escribirse exactamente igual que con el nombre del fichero. Mira más abajo cómo colocar la imagen en la carpeta correcta.
- **Image(image name, x, y)**: Muestra la PImage **image name** en las coordenadas **x, y**. La esquina superior izquierda es el eje de coordenadas.

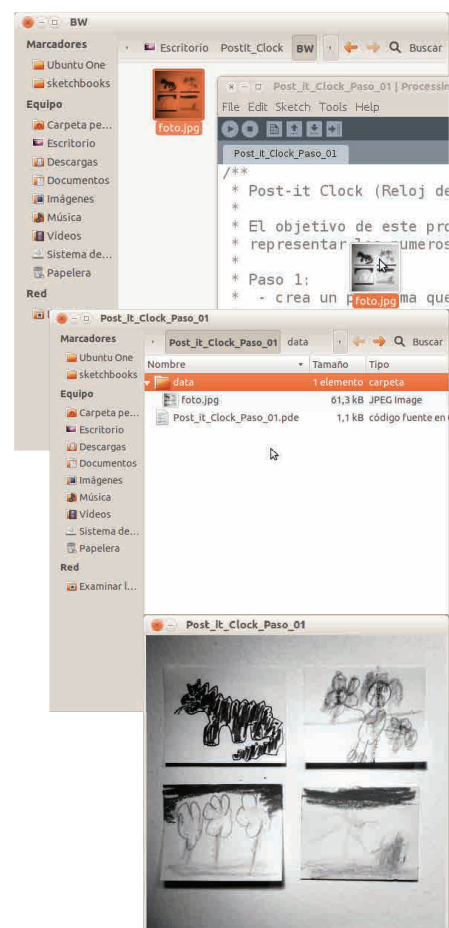
Añadir imágenes

Para que el programa tenga acceso a la imagen, debes arrastrar el fichero de la imagen en el IDE de Processing, como muestra la siguiente captura:

La imagen se almacena en una subcarpeta del programa llamada **data**. Esta captura de pantalla muestra la estructura de los ficheros donde puedes encontrar el archivo de la imagen. También puedes encontrarlo abriendo la carpeta desde el menú 'Sketch -> Mostrar carpeta del Sketch'.

Cuando ejecutes el programa, la imagen de abajo es lo que deberías ver, un collage con cuatro Post-its como imagen de fondo para tu ventana de programa:

En este programa vamos a crear una variable **image** llamada **im**. En **setup()** almacenamos una imagen llamada **foto.jpg** en nuestra variable, utilizando **loadImage()**. En **draw()** mostramos la imagen usando **image()**. Puesto que nuestra ventana de programa se establece al mismo tamaño que la imagen, dicha imagen encajará perfectamente con la ventana. Puedes cambiar la imagen con cualquier otra que hayas hecho tu mismo o descargado de Internet. Asegúrate de que el tamaño de ventana y de la imagen sean iguales para que el programa tenga una buena apariencia.



Mostrando dos imágenes

Para poner dos imágenes una al lado de la otra, debes declarar una variable con distinto nombre pero del mismo tipo `PImage` y cargar la segunda imagen en el programa. Deberás ajustar las coordenadas de esta segunda imagen.

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 2:
8   * - modifica el programa para que muestre dos imagenes
9   * - necesitaras dos variables, una para cada imagen
10  * - la funcion 'image()' usa coordenadas, muestra las imagenes una junto a otra
11  * - incluye la segunda imagen en el progama arrastrandola sobre el IDE
12  *
13  * Nota: si tus imagenes son de 400x400, necesitaras un programa de 800x400 pixeles
14  *
15  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
16  */
17
18 PImage im;
19 PImage im2;
20
21 void setup() {
22   size(800, 400); // Hacemos el programa dos veces el tamaño de la imagen en ancho
23   im = loadImage("foto.jpg");
24   im2 = loadImage("foto2.jpg"); // Nombre de la segunda imagen
25 }
26
27 void draw() {
28   image(im, 0, 0);
29   image(im2, 400, 0); // Para mostrar la segunda imagen en la pantalla, coordenadas
30   400, 0
31 }
```

Simplemente hemos añadido otra `PImage` llamada `im2`. Esta imagen es del mismo tamaño que la primera, por lo que la ventana debe ser el doble de grande en el eje x para que quepan las dos imágenes.



Arrays

Cuando trabajamos con muchas imágenes podemos utilizar algo llamado arrays que permiten un acceso más fácil a ellas. Un array es como una caja con muchos compartimentos. Cada compartimento puede tener diferentes objetos y se pueden encontrar referenciando dicho compartimento.

Por ejemplo, un array llamado `CajaDeFruta` tiene 3 compartimentos y puede contener 3 objetos. El compartimento 1 tiene un plátano, el compartimento 2 tiene una fresa y el compartimento 3 tiene una cereza. Si quieres encontrar la cereza, mirarás en el compartimento 3 de la `CajaDeFruta`. El orden de los arrays siempre empieza en 0, por lo que `CajaDeFruta[0]` contiene el plátano, `CajaDeFruta[1]` contiene la fresa y `CajaDeFruta[2]` la cereza.

Los arrays pueden ser de un cierto tipo como entero o `int` o una cadena como `String`. El máximo número de variables que pueden almacenar debe ser declarado con el array – 3 en el ejemplo de arriba. El siguiente programa, que es una modificación del anterior, muestra como usar un array del tipo `PImage` para almacenar dos imágenes.

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 3:
8   * - modifica el programa para que use arrays
9   * - los arrays:
10  *   + son variables que pueden almacenar mas de un dato
11  *   + tienen un solo nombre, por ejemplo 'img'
12  *   + los datos se distinguen porque "van numerados", es lo que llamamos "indexados"
13  *   + el primer dato es el 0, el segundo el 1, el tercero el 2 ...
14  *   + un dato se pide con el nombre del array seguido del numero entre
15  *     corchetes 'img[3]'
16  *   + se declaran usando el tipo de variable con corchetes (mira el codigo)
17  *   + al declararlos hay que decir cuantos datos van a contener
18  *
19  * Nota: este programa hace lo mismo que el anterior, solo que de forma mas eficiente
20  *
21  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
22  */
23
24  PImage im[] = new PImage[2]; // Declaracion de un array de 2 imagenes
25
26  void setup() {
27    size(800, 400);
28    // La primera imagen va a la primera posicion del array img[]
29    im[0] = loadImage("foto.jpg");
30    // La segunda imagen va a la segunda posicion del array
31    im[1] = loadImage("foto2.jpg");
32  }
33
34  void draw() {
35    image(im[0], 0, 0);
36    image(im[1], 400, 0);
37  }
```

Nuevos comandos:

- La declaración de las imágenes es un poco diferente: `PImage im[] = new PImage[2]`, esto significa que declaramos un array del tipo `PImage` con dos bloques (o compartimentos) de datos.
- Cuando mostramos y cargamos las imágenes, usamos `img[0]`, para la primera posición del array, y `img[1]`, para la segunda.

A pesar de que el programa ha sido modificado, no hay ninguna diferencia en el resultado.

Usando dos Arrays

Para en adelante simplificar la asignación de las variables imagen, el truco es almacenar los nombres de todas las imágenes en un array de strings (cadenas de texto). Aquí un ejemplo:

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 4:
8   * - modifica el programa de modo que los nombres de imagenes se almacenen
9   *   en un array, de este modo tendras dos arrays, uno para los nombres de
10  *   imagenes y otro para las imagenes
11  * - toma ventaja del hecho de que se puede hacer un array declarando sus
12  *   contenidos entre llaves '{dato, dato2 ...}' con los datos separados por comas
13  * - en el programa ejemplo, "foto.jpg" es la posicion 0 del array,
14  *   "foto2.jpg" es la 1
15  *
16  * Nota: este programa hace lo mismo que el anterior
17  *
18  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
19  */
20
21 PImage im[] = new PImage[2];
22 String imFile[] = {"foto.jpg", "foto2.jpg"}; // Array de nombres de fichero
23
24 void setup() {
25   size(800, 400);
26   im[0] = loadImage(imFile[0]); // Primer fichero a la funcion 'loadImage()'
27   im[1] = loadImage(imFile[1]); // Segundo fichero
28 }
29
30 void draw() {
31   image(im[0], 0, 0);
32   image(im[1], 400, 0);
33 }
```

Como podrás observar cuando ejecutes este programa, el resultado es el mismo que para los otros. La gran pregunta es, ¿Por qué utilizar esto para hacer lo mismo? Y la respuesta es simple: lo hace mucho más fácil para seguir trabajando desde aquí. Si necesitas cambiar algo, por ejemplo, sólo lo tendrás que hacerlo en un único lugar y no buscar por todo el programa. Cuando añadas más imágenes y para programas muy complejos, esto es importante.

El bucle for()

En este paso aumentaremos el número de imágenes que se muestran de 2 a 4. Todavía utilizaremos los 2 arrays de los pasos anteriores, pero para hacer el código más eficiente incluso, introduciremos una cosa nueva: el bucle for(). Esta es una de las funciones más prácticas y usadas en programación. Hace exactamente lo que parece, iterar, al igual que draw(). Veamos un ejemplo para seguir explicando como funciona.

Descarga las imágenes banana.jpg, peninsula.jpg, postit.jpg y tortilla.jpg

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 5:
8   * - añade mas imagenes a tu programa (aqui te pasamos un par mas)
9   * - recuerda que tienes que arrastrarlas sobre el IDE o copiarlas directamente
10  *   en la carpeta 'data' de tu programa
11  * - usa el bucle 'for()' para cargar todas tus imagenes en el array de forma
12  *   mas eficiente
13  * - el bucle 'for()':
14  *   + se usa en programacion para ejecutar acciones que se repiten varias veces
15  *   + es util para recorrer arrays
16  *   + tiene tres parametros separados por ';' (punto y coma)
17  *   + for( inicializacion; comprobacion; incremento )
18  *
19  * Nota: busca mas informacion sobre for en la referencia de Processing
20  *
21  * (c) 2013 D. Cuartielles, Arduino Verktstad, Suecia
22  */
23
24 PImage im[] = new PImage[4]; // Haz tu array mas grande para que le quepan 4 imagenes
25 String imFile[] = {"postit.jpg", "peninsula.jpg", "tortilla.jpg", "banana.jpg"};
26
27 void setup() {
28   size(800, 800); // Haz tu programa mas grande para acomodar todas las imagenes
29   for (int i = 0; i < 4; i = i + 1) {
30     im[i] = loadImage(imFile[i]); // Carga la imagen segun el contador 'i'
31   }
32 }
33
34 void draw() {
35   image(im[0], 0, 0); // Muestra las imagenes
36   image(im[1], 400, 0);
```

```

37  image(im[2], 0, 400);
38  image(im[3], 400, 400);
39  }

```

Nuevos comandos:

- `for(initiator; test; update){ statements }`: Esta función permite repetir una pieza de código tantas veces como necesites. La primera cosa que sucede dentro del `for()` es la inicialización, `initiator`. En este ejemplo esta declaración es un entero, `i`. Luego comprueba si `test` es verdadero o falso, tal y como hicimos con el `if` en el proyecto anterior, ¿recuerdas?. En este ejemplo `test` comprueba si `i` es menor que 4. Si así es, el código entre llaves se ejecutará. A continuación, lo que ocurre es el `update`, en este caso se trata de incrementar `i` una unidad. Después, `test` se comprueba de nuevo y si, nuevamente es cierto, se repite lo anterior. Puesto que siempre actualizamos añadiendo 1 a `i`, en algún momento `i` será igual que 4, lo que significa que `test` será falsa (4 no es menor que 4). En ese momento, se abandonará el bucle.

Aquí está el resultado:

Lo que hemos hecho es crear dos arrays con 4 posiciones cada uno. Uno para las imágenes y otro para el nombre de las mismas. En `setup()` establecemos el tamaño de la ventana para que quepan las 4 imágenes. A continuación, utilizamos el bucle `for()` para que cargue las imágenes.

El bucle se ejecuta 4 veces, tantas como imágenes queremos cargar. Esto se establece en el `test` con la condición `i<4` y en el `update` escribiendo `i=i+1`. La mejor parte de esto es que podemos usar la variable `i` para acceder a las posiciones de nuestros arrays. Por tanto, la primera vez que entramos en el `for`, `i` es igual a 0, lo que significa que cargamos la imagen `imFile[0]` ("postit.jpg") y la guardamos en `im[0]`. La siguiente vez, `i` es 1, etc. De esta manera, no tenemos que asignar todas las imágenes una por una con variables separadas, simplemente dejamos al bucle `for()` hacer todo el trabajo. Con esto conseguimos un código más compacto.



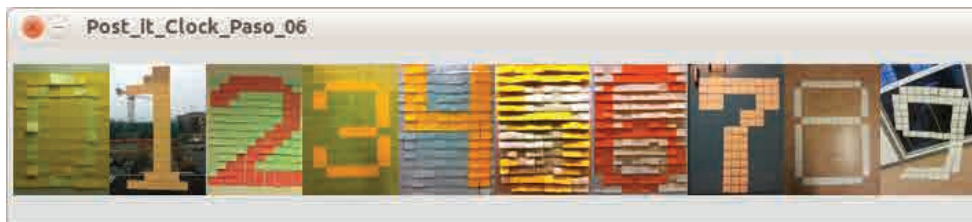
Múltiples imágenes

Puesto que el objetivo de este programa es crear un reloj, el siguiente paso es utilizar diferentes imágenes que representen los números y usarlas en nuestro programa. Para tu conveniencia, te damos una serie de imágenes hechas con notas de Post-it.

Descarga las imágenes que representan los números: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 6:
8   * - crea imagenes que representen numeros (por si acaso, aqui tienes unas
9   *   hechas con post-it)
10  * - modifica el programa para que muestre todos los numeros del 0 al 9
11  * - puedes usar 'for()' para mostrar las imagenes
12  * - este programa te servira como base para cualquier programa donde quieras
13  *   representar numeros, si por ejemplo quieres mostrar el 5 en la pantalla,
14  *   solo tienes que invocar 'image(img[5], coorX, coorY)'
15  *
16  * Nota: como quieres mostrar todos los numeros, tendras que hacer imagenes mas
17  *       pequeñas, por ejemplo de 70 píxeles de ancho
18  *
19  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
20  */
21
22  PImage im[] = new PImage[10]; // Array para 10 imagenes
23  String imFile[] = {"0.jpg", "1.jpg", "2.jpg", "3.jpg", "4.jpg", "5.jpg", "6.jpg",
24  "7.jpg", "8.jpg", "9.jpg"};
25
26  void setup() {
27    size(700, 95); // 10 imagenes de 70 píxeles de ancho y 95 de alto
28    for (int i = 0; i < 10; i = i + 1) {
29      im[i] = loadImage(imFile[i]);
30    }
31  }
32
33  void draw() {
34    for (int i = 0; i < 10; i = i + 1) {
35      image(im[i], 70 * i, 0); // Muestra las imagenes en secuencia
```

No hemos introducido ningún comando nuevo aquí. Lo que hemos hecho ha sido crear 10 posiciones en cada array. Hemos establecido el tamaño de ventana a 700x95 para que quepan las 10 imágenes de 70x95. Tenemos un bucle **for()** que itera 10 veces para cargar cada una de las imágenes, y otro bucle **for** en **draw**. Este bucle itera diez veces y simplemente se usa para mostrar las imágenes. Puesto que las imágenes son de 70 píxeles de ancho, podemos establecer la coordenada x para cada imagen como **70*i**.



El Tiempo

En este paso te enseñaremos como mostrar el tiempo. Ya has aprendido como acceder a diferentes variables del sistema con Processing. Por ejemplo, ya hemos usado `mouseX`, `mouseY` y `frameRate`. Hay variables que podemos utilizar para obtener el tiempo actual; `hour()`, `minute()` y `second()`. Empecemos con cómo mostrar el tiempo por sólo las horas:

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 7:
8   * - toma la hora del ordenador y representala con las imagenes
9   * - la hora del sistema se obtiene con la funcion 'hour()'
10  * - ten en cuenta que la hora tiene dos digitos y por lo tanto
11  *   tienes que buscar la forma de separarlos
12  *
13  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
14  */
15
16  PImage im[] = new PImage[10]; // Array para 10 imagenes
17  String imFile[] = {"0.jpg", "1.jpg", "2.jpg", "3.jpg", "4.jpg", "5.jpg", "6.jpg",
18  "7.jpg", "8.jpg", "9.jpg"};
19
20  void setup() {
21    size(140, 95); // 2 digitos de 70 píxeles de ancho y 95 de alto
22    for (int i = 0; i < 10; i = i + 1) {
23      im[i] = loadImage(imFile[i]);
24    }
25  }
26
27  void draw() {
28    int h = hour(); // Toma la hora del reloj del ordenador y almacenalo en
29    // una variable
30    int h_dec = int(h / 10); // Extrae el digito de mayor peso de la hora (decenas)
31    int h_uni = h - h_dec * 10; // Extrae el digito de menor peso de la hora (unidades)
32
33    image(im[h_dec], 0, 0); // Muestra el digito de las decenas
34    image(im[h_uni], 70, 0); // Muestra el digito de las unidades
35  }
```

Nuevos comandos:

- `hour()`: Devuelve la hora actual como un número de 0 a 23.
- `int(data)`: Convierte `data` a un entero. P.ej. `int(2.545)` devuelve 2, `int(233.9999)` devuelve 233 etc.



La primera cosa que se hace en `draw()` es almacenar el número de horas en la variable `h`. Ahora este número tiene dos dígitos pero nuestras imágenes sólo contienen uno, por lo que tendremos que calcular qué dígitos mostrar. El primer dígito se almacena en `h_dec` y se calcula como `hour()`, `minute()` and `second()`. El segundo dígito se almacena en `h_uni` y se calcula como `h - h_dec * 10`.

Digamos que la hora actual son las 14. Esto significa que `h` es igual a 14, `h_dec` es 1 y `h_uni` vale 4. Estas variables ahora pueden ser usadas para decidir qué imagen mostrar.

El reloj final

Este es el último paso del proyecto. En este paso vamos a añadir los minutos y los segundos a nuestro reloj utilizando el mismo método que en el paso anterior con las horas.

```
/**
 * Post-it Clock (Reloj de Post-it)
 *
 * El objetivo de este proyecto es crear un reloj usando imagenes para
 * representar los numeros.
 *
 * Paso 8:
 * - añade los minutos y los segundos a tu reloj
 * - los minutos del sistema se obtienen con la funcion 'minute()'
 * - los segundos se obtienen con la funcion 'seconds()'
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
 */

PImage im[] = new PImage[10]; // Array para 10 imagenes
String imFile[] = {"0.jpg", "1.jpg", "2.jpg", "3.jpg", "4.jpg", "5.jpg", "6.jpg", "7.jpg",
"8.jpg", "9.jpg"};

void setup() {
  size(140, 285); // 6 digitos en tres filas y dos columnas
  for (int i = 0; i < 10; i = i + 1) {
```

```

    im[i] = loadImage(imFile[i]);
}
}

void draw() {
    int h = hour();
    int h_dec = int(h / 10);
    int h_uni = h - h_dec * 10;

    image(im[h_dec], 0, 0);
    image(im[h_uni], 70, 0);

    int m = minute();          // Toma los minutos del reloj del ordenador y almacenalos
    en una variable
    int m_dec = int(m / 10);    // Extrae el dígito de mayor peso de los minutos (decenas)
    int m_uni = m - m_dec * 10; // Extrae el dígito de menor peso de los minutos (unidades)

    image(im[m_dec], 0, 95);    // Muestra el dígito de las decenas
    image(im[m_uni], 70, 95);   // Muestra el dígito de las unidades

    int s = second();          // Toma los segundos del reloj del ordenador y almacenalos
    en una variable
    int s_dec = int(s / 10);    // Extrae el dígito de mayor peso de los segundos (decenas)
    int s_uni = s - s_dec * 10; // Extrae el dígito de menor peso de los segundos (unidades)

    image(im[s_dec], 0, 190);   // Muestra el dígito de las decenas
    image(im[s_uni], 70, 190);  // Muestra el dígito de las unidades
}

```

Nuevos comandos:

- `minute()`: devuelve los minutos actuales como un valor de 0 a 59.
- `second()`: devuelve los segundos actuales como un valor entre 0 y 59.

Declaramos las variables para los minutos y segundos y calculamos los dígitos a mostrar de la misma manera que en el paso anterior.

Sigue experimentando

Usando las mismas variables de sistema, crea un temporizador para la cocina, por ejemplo un reloj que cuente hacia atrás con imágenes.



Caza la manzana

En este mini proyecto programarás un juego de ordenador. El objetivo de este ejercicio es llevar la programación un poco más lejos y crear un pequeño videojuego donde nuestro aguerrido héroe, el famoso científico Newton, intenta no perder la oportunidad de que la manzana le caiga en la cabeza.

Vamos a crear, paso a paso, un programa en el que Newton colecciona puntos durante medio minuto, al recibir tantos manzanazos en la cabeza como sea posible.

Crea una manzana y un científico

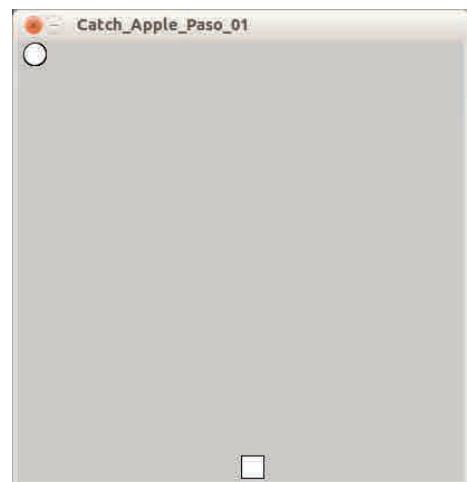
Crea la pantalla del juego. Las manzanas, por ahora, serán círculos que caen del cielo, mientras que Newton será un cuadrado al fondo de la pantalla.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Note: if you don't know about Newton's myth with the apple, you should
10  *       check it out on an encyclopedia, internet, or the like
11  *
12  * Step 1:
13  *   - create the game screen, the apples will be circles falling from the sky, while
14  *     Newton will be represented by a square. We will change those later
15  *
16  * (c) 2013 D. Cuartielles, Arduino Verksstad, Sweden
17  */
18
19 void setup() {
20   size(400, 400); // Make the screen of an average size
21 }
22
23 void draw() {
24   ellipse(15, 15, 20, 20); // Draw the apple at the top of the screen
25   rect(width / 2, height - 25, 20, 20); // Draw Newton at the bottom of the screen
26 }
```

Nuevos comandos:

`rect(x, y, width, height)`: Dibuja un rectángulo. `x` e `y` establecen la posición de la esquina superior izquierda, `width` y `height` establecen el tamaño.

De momento, Newton no se puede mover del centro de la ventana para coger la manzana, que está situada a su izquierda.



Toma el control de Newton con el teclado

Haz que Newton (el cuadrado) se mueva con las flechas del teclado. Para acceder al teclado desde Processing, considera lo siguiente: cada tecla del teclado tiene un `keyCode`. El `keyCode` es el valor o símbolo que representa una tecla específica. `keyCode` es una variable de sistema interna que puede ser usada para detectar qué tecla ha sido pulsada. Para ver que tecla ha sido apretada, vamos a usar una función llamada `keyPressed()`. Veremos que también necesitamos una variable para guardar las coordenadas del eje X (horizontal) del cuadrado, de forma que lo podamos ajustar cada vez que una tecla sea pulsada.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 2:
10  * - make Newton (square) move using the keys in the keyboard
11  * - each key in a keyboard has a 'keyCode' representing a value or
12  *   symbol code
13  * - to check which key was pressed, create a function called 'keyPressed()'
14  * - to move the square you will need a variable to store its X coordinate
15  *   (horizontal movement)
16  *
17  * (c) 2013 D. Cuartielles, Arduino Verktstad, Sweden
18  */
19
20 int nX = 0;
21
22 void setup() {
23   size(400, 400); // Draw the sketch at a not-too-small size
24 }
```

```

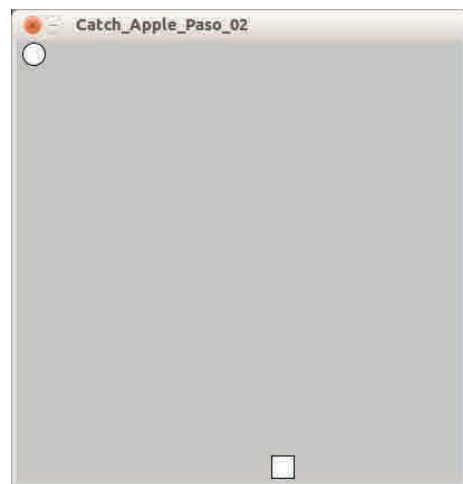
25
26 void draw() {
27   background(200); // Clear the screen
28   ellipse(15, 15, 20, 20);
29   rect(nX, height - 25, 20, 20); // Draw Newton with a variable X coordinate
30 }
31
32 void keyPressed() {
33   // If the right key was pressed, increase the X coordinate in 3 units
34   if (keyCode == RIGHT) {
35     nX = nX + 3;
36   }
37   // If the left key was pressed, decrease the X coordinate in 3 units
38   if (keyCode == LEFT) {
39     nX = nX - 3;
40   }
41 }

```

Nuevos comandos:

- **keyPressed()**: esta función se llama cada vez que una tecla es pulsada. Esto significa que cualquier código escrito dentro de esta función será ejecutado al pulsar una tecla.
- **keyCode**: devuelve el valor de la tecla que has pulsado.

La variable **nX** se usa para establecer la posición x de Newton. Cada vez que se pulsa el cursor derecho, **nX** se incrementa con 3 y cada vez que se pulsa el cursor izquierdo, **nX** se decrementa en 3. Ahora ya puedes mover el cuadrado por la pantalla de izquierda a derecha, pero ten cuidado, porque puede salirse de los límites de la pantalla.



Limita los movimientos del cuadrado

Limita el movimiento de Newton dentro de la ventana. Para esto vas a usar las funciones condicionales '**if-else**', con las que comprobarás que la coordenada X esté siempre dentro de la ventana del programa. Es decir, tendrá que ser mayor que 0 y menor que la anchura de la ventana o **width**.

```

/*
 * Catch the Apple (Caza la Manzana)
 *
 * Creacion de un videojuego con Processing. El objetivo de este ejercicio es
 * llevar la programacion un poco mas lejos y crear un pequeño videojuego donde
 * nuestro aguerrido heroe, el famoso científico Newton, intenta no perder la

```

```

* oportunidad de que la manzana le caiga en la cabeza.
*
* Paso 3:
* - limita el movimiento de Newton dentro de la ventana
* - para esto vas a usar las funciones condicionales 'if-else'
* - comprobaras que la coordenada nX esta siempre dentro de la
*   ventana del programa, es decir, habra de ser mayor que 0 y
*   menor que la anchura de la ventana o 'width'
*
* (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
*/

int nX = 0;

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

void draw() {
  background(200);
  ellipse(15, 15, 20, 20);
  rect(nX, height - 25, 20, 20);
}

void keyPressed() {
  // Incrementa las coordenadas en 3 unidades
  if (keyCode == RIGHT) {
    nX = nX + 3;
  }
  // Decrementa las coordenadas en 3 unidades
  if (keyCode == LEFT) {
    nX = nX - 3;
  }
  // Limita el valor de la coordenada X
  if (nX < 0) {
    nX = 0;
  }
  if (nX > width - 20) { // Por la derecha tienes que tener en cuenta el tamaño del cuadrado
    nX = width - 20;
  }
}

```

En este caso, no hay una gran diferencia en lo que se verá en la pantalla del programa, con la excepción de que el cuadrado ya no se saldrá de los límites.

Manzanas que caen

Modifica el programa para hacer que la manzana (círculo) caiga de lo alto de la pantalla. Para esto, necesitarás una variable que contenga la coordenada Y del círculo y, cuando la manzana toque el suelo, que aparezca otra arriba.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 4:
10  *   - modify the program to make the apple fall
11  *   - you will need a variable to store the Y coordinate for the apple
12  *   - when the apple touches the ground, you will need to "hang it" from the tree again
13  *
14  * (c) 2013 D. Cuartielles, Arduino Verktstad, Sweden
15  */
16
17  int nX = 0;
18  int mY = 0; // Apple's Y coordinate
19
20  void setup() {
21    size(400, 400);
22  }
23
24  void draw() {
25    background(200);
26
27    mY = mY + 1; // Increase apple's coordinate
28    if (mY > height) {
29      mY = 15; // If the apple touches the ground, lift it again
30    }
31    ellipse(15, mY, 20, 20); // Make the Y coordinate into a variable
32    rect(nX, height - 25, 20, 20);
33  }
34
35  void keyPressed() {
36    // Increase the coordinates in 3 píxeles
37    if (keyCode == RIGHT) {
38      nX = nX + 3;
39    }
40    // Decrease the coordinates in 3 píxeles
41    if (keyCode == LEFT) {
42      nX = nX - 3;
43    }
44    // Limit the X coordinates
```

```

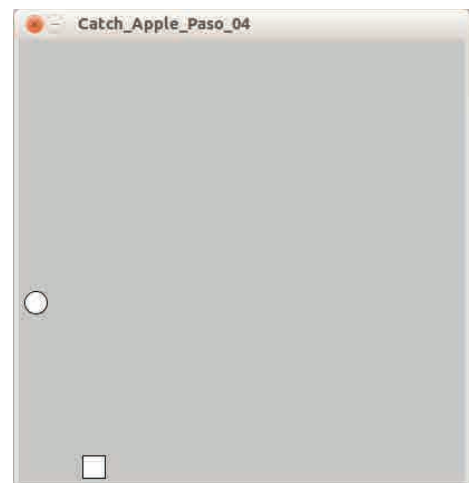
45   if (nX < 0) {
46       nX = 0;
47   }
48   if (nX > width - 20) {
49       nX = width - 20;
50   }
51 }

```

Hemos creado una variable `mY` para almacenar la coordenada Y de la manzana. Cada vez que `draw()` se ejecuta, `mY` se incrementa en 1, haciendo que la manzana vaya acercándose al suelo de la ventana. Una vez que la manzana llega al suelo, es decir, `mY` es mayor que `height`, debe reaparecer mágicamente en lo alto de la ventana.

Un poco de azar

Hasta ahora, las manzanas siempre salen de la misma posición en lo alto de la pantalla, así es bastante predecible. Para cambiar la X de origen y que cada vez salga de un sitio distinto del árbol, haremos uso de una función llamada `random()` que permite generar números aleatorios. Necesitarás una nueva variable para almacenar la posición en X de la manzana. Ten en cuenta que la coordenada habrá que cambiarla solo cuando la manzana llegue al suelo, porque si no cambiará aleatoriamente durante su caída.



```

1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 5:
10  *   - the apple is always falling from the same place on the screen, change its X
11  *     coordinate when created to show up at different places
12  *   - the 'random()' function will create a random number, it returns a number
13  *     between 0 and whatever number you pass to it as a parameter between the brackets
14  *   - you will need a new variable to store the X position of the apple
15  *   - the coordinate value has to be changed only when the apple touches the ground
16  *
17  * (c) 2013 D. Cuartielles, Arduino Verktad, Sweden
18  */
19
20 int nX = 0;
21 int mY = 0;

```

```

22 int mX = 15; // Apple's X coordinate
23
24 void setup() {
25   size(400, 400);
26 }
27
28 void draw() {
29   background(200);
30
31   mY = mY + 1;
32   if (mY > height) {
33     mY = 15;
34     mX = int(random(width - 20)); // Initialize the X coordinate of the apple to a
    random number
35   }
36   ellipse(mX, mY, 20, 20); // Include the changes to the X coordinate to the circle's
    movement
37   rect(nX, height - 25, 20, 20);
38 }
39
40 void keyPressed() {
41   // Increase the coordinates in 3 píxeles
42   if (keyCode == RIGHT) {
43     nX = nX + 3;
44   }
45   // Decrease the coordinates in 3 píxeles
46   if (keyCode == LEFT) {
47     nX = nX - 3;
48   }
49   // Limit the X coordinates
50   if (nX < 0) {
51     nX = 0;
52   }
53   if (nX > width - 20) {
54     nX = width - 20;
55   }
56 }

```

Nuevos comandos:

- **random(high)**: genera un número aleatorio entre 0 y el número high. Puedes también usar **random(low,high)** para generar un número entre low y high.

Con este cambio en el programa, podrás ver que las manzanas salen desde cualquier punto en lo alto de la pantalla.

Nuevos comandos:

Detección de colisión

Detecta que la manzana aterriza en la cabeza de Newton para poder contar puntos; la acción de detectar que dos objetos se chocan en la pantalla se llama detección de colisión. Utiliza las funciones condicionales **if-else** para ver si el círculo está sobre el cuadrado y, de ser así, cambia el color del cuadro a rojo. Para hacer el programa más sencillo, crearemos una variable que almacene las coordenadas Y del cuadrado, y así poder hacer las comparaciones de un modo más sencillo.

```
1  /*
2  * Catch the Apple
3  *
4  * Create a videogame using Processing. The game for this exercise is
5  * getting your programming skills a little further and making a computer
6  * game where our hero, the famous scientist Newton, will not let the chance
7  * go of having an apple hitting his head.
8  *
9  * Step 6:
10 * - detect the apple landing on Newton's head to be able of counting points;
11 *   checking whether two objects on a screen touch each other is called
12 *   "collision detection"
13 * - use the conditional functions 'if-else' to check whether the circle is
14 *   on top of the square, if so, change the filling color to red
15 * - to make the program easier, make a variable to store the square's Y coordinate
16 *
17 * (c) 2013 D. Cuartielles, Arduino Verktad, Sweden
18 */
19
20 int nX = 0;
21 int nY = 0; // Y coordinate for the square
22 int mY = 0;
23 int mX = 15;
24
25 void setup() {
26   size(400, 400);
27   nY = height - 25; // Init the coordinate Y for the square to be at the end of the
   screen
28 }
29
30 void draw() {
31   background(200);
32
33   mY = mY + 1;
34   if (mY > height) {
35     mY = 15;
36     mX = int(random(width - 20));
37   }
38
39   fill(255); // By default fill in the shapes white
40
41   // Collision detection
```

```

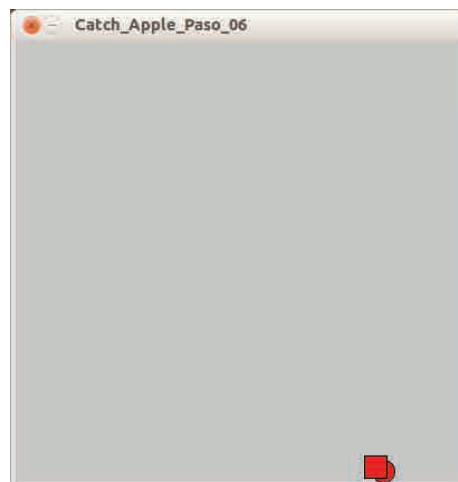
42  if (mY + 10 > nY && mY - 10 < nY + 20) { // Is the circle at the same height as the
square?
43      if (mX + 10 > nX && mX - 10 < nX + 20) { // Is the circle on top of the square?
44          fill(255, 0, 0); // Change the filling color to red
45      }
46  }
47
48  // Lines to understand collision detection
49  // uncomment them to test how things work
50  //line(0,mY-10,width,mY-10);
51  //line(mX-10,0,mX-10,height);
52  //line(0,mY+10,width,mY+10);
53  //line(mX+10,0,mX+10,height);
54
55  ellipse(mX, mY, 20, 20);
56  rect(nX, nY, 20, 20); // Include a variable to control the Y coordinate
57 }
58
59 void keyPressed() {
60     // Increase the coordinates in 3 píxeles
61     if (keyCode == RIGHT) {
62         nX = nX + 3;
63     }
64     // Decrease the coordinates in 3 píxeles
65     if (keyCode == LEFT) {
66         nX = nX - 3;
67     }
68     // Limit the X coordinates
69     if (nX < 0) {
70         nX = 0;
71     }
72     if (nX > width - 20) {
73         nX = width - 20;
74     }
75 }

```

Nuevos comandos:

- **if(test && test) {statements}**: esto se utiliza cuando varias comprobaciones deben ser realizadas en un único **if()**. En este ejemplo, comprobamos si **mY+10 > nY** y si **mY-10 < nY+20**. Si estas dos comprobaciones son ciertas simultáneamente, ejecutamos el código entre llaves.

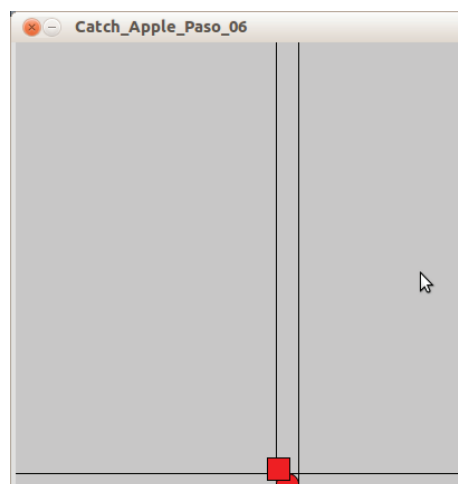
La detección de colisión consiste en dos `if()`, el primero comprueba si el círculo está a la misma altura que el cuadrado. Si es así, el segundo `if()` comprueba si el círculo está arriba del cuadrado. De ser así, el `fill()` colorea el círculo en rojo. Una colisión debe ser similar a esto:



Por otra parte, si activas las siguientes líneas en el programa:

```
1 // lines of code to understand how collision
  works
2 // erase the comment in order to see the code
3 line(0,mY-10,width,mY-10);
4 line(mX-10,0,mX-10,height);
5 line(0,mY+10,width,mY+10);
6 line(mX+10,0,mX+10,height);
```

Verás una serie de líneas en la pantalla enmarcando el movimiento de la manzana. Puedes emplearlas para ver cómo funciona la detección de colisión.



Más rápido

¿No quieres que las manzanas caigan más rápido? Modifica la velocidad de las manzanas para hacer el juego algo más interesante. Para ello, crea una variable `float` para que `mV` almacene la velocidad de caída de la manzana. Puedes cambiar la velocidad modificando el valor de `mV`. Al mismo tiempo, para poder controlar mejor el movimiento de la manzana en el eje Y, vamos a modificar el tipo de variable de `mY` para que sea `float`. ¿Recuerdas `float` de la sección de variables? Una variable con datos del tipo `float` puede almacenar números decimales.

```

1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 7:
10  * - modify the way the apples fall to reach higher speeds
11  * - introduce a variable to store the speed
12  * - declare that variable as 'mV' as a floating point number
13  * - you can modify the speed by changing just this variable
14  * - to control the apple's movement on the Y axis, modify the type of
15  *   the 'mY' variable to be a 'float'
16  *
17  * (c) 2013 D. Cuartielles, Arduino Verktstad, Sweden
18  */
19
20 int nX = 0;
21 int nY = 0;
22 float mY = 0; // Make mY into a float
23 int mX = 15;
24 float mV = 3; // Apple's falling speed
25
26 void setup() {
27   size(400, 400);
28   nY = height - 25;
29 }
30
31 void draw() {
32   background(200);
33
34   // Apple's movement
35   mY = mY + mV; // Introduce the speed as an increment
36   if (mY > height) {
37     mY = 15;
38     mX = int(random(width - 20));
39   }
40
41   fill(255);
42
43   // Collision detection
44   if (mY + 10 > nY && mY - 10 < nY + 20) {
45     if (mX + 10 > nX && mX - 10 < nX + 20) {
46       fill(255, 0, 0);
47     }
48   }
49
50   ellipse(mX, mY, 20, 20);
51   rect(nX, nY, 20, 20);

```

```

52 }
53
54 void keyPressed() {
55     // Increase the coordinates in 3 píxeles
56     if (keyCode == RIGHT) {
57         nX = nX + 3;
58     }
59     // Decrease the coordinates in 3 píxeles
60     if (keyCode == LEFT) {
61         nX = nX - 3;
62     }
63     // Limit the X coordinates
64     if (nX < 0) {
65         nX = 0;
66     }
67     if (nX > width - 20) {
68         nX = width - 20;
69     }
70 }

```

En lugar de incrementar **mY** en uno, vamos a incrementarlo con **mV**. A parte de la velocidad, nada ha cambiado en el programa, por lo que sólo hay una pequeña diferencia entre este y el programa anterior.

A Newton le gusta la gravedad, dale más

Modifica la caída de las manzanas para que responda a la aceleración de la gravedad. De este modo, las manzanas irán cada vez más rápido cuanto más tiempo lleven cayendo. Como sabes, la velocidad se calcula a partir de la aceleración, y del mismo modo, la posición a partir de la velocidad. Para hacer estas operaciones de la forma más sencilla posible, introduce una variable en coma flotante (**float**) que represente la aceleración.

```

1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 8:
10  * - modify the apples' falling speed for them to follow the gravity
11  * - as you know, speed can be calculated from acceleration, and position
12  *   from speed
13  * - introduce a variable representing acceleration, make it a 'float'
14  *
15  * (c) 2013 D. Cuartielles, Arduino Verktstad, Sweden
16  */
17
18 int nX = 0;
19 int nY = 0;

```

```

20 float mY = 0;
21 int mX = 15;
22 float mV = 0;    // Apple's initial speed is zero
23 float mA = 0.05; // Apple's initial acceleration (0.98 would be too much)
24
25 void setup() {
26     size(400, 400);
27     nY = height - 25;
28 }
29
30 void draw() {
31     background(200);
32
33     // Apple's movement
34     mV = mV + mA; // Estimate the speed according to the acceleration
35     mY = mY + mV; // Estimate the position according to the speed
36     if (mY > height) {
37         mY = 15;
38         mX = int(random(width - 20));
39         mV = 0; // Apples start falling at zero speed
40     }
41
42     fill(255);
43
44     // Collision detection
45     if (mY + 10 > nY && mY - 10 < nY + 20) {
46         if (mX + 10 > nX && mX - 10 < nX + 20) {
47             fill(255, 0, 0);
48         }
49     }
50
51     ellipse(mX, mY, 20, 20);
52     rect(nX, nY, 20, 20);
53 }
54
55 void keyPressed() {
56     // Increase the coordinates in 3 pixels
57     if (keyCode == RIGHT) {
58         nX = nX + 3;
59     }
60     // Decrease the coordinates in 3 pixels
61     if (keyCode == LEFT) {
62         nX = nX - 3;
63     }
64     // Limit the X coordinates
65     if (nX < 0) {
66         nX = 0;
67     }
68     if (nX > width - 20) {
69         nX = width - 20;
70     }
71 }

```

Cada vez que incrementes `mY` incrementaremos también `mV` un poco. De esta manera, la velocidad será un poco más grande cada vez. A parte de la aceleración, nada ha cambiado en el programa, por lo que sólo hay una pequeña diferencia entre este programa y el anterior.

Cuenta los puntos

Implementa un contador que muestre cuántas manzanas golpearon a Newton. Para mostrar el texto, puedes utilizar la función `text()`. Además, necesitarás un contador para anotar los puntos.

Nota: como empiezas a tener muchas variables en tu programa, es recomendable que añadas comentarios para recordar qué hace cada una.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will no let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 9:
10  * - implement a counter to show home many apples Newton caught
11  * - the function 'text()' will allow add text to the sketch
12  * - you will also need a counter to follow the points scored while playing
13  * - you start to have many variables in your program, it is considered good
14  *   practice to add comments to explain each one of them
15  *
16  * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
17  */
18
19 int nX = 0;      // X coordinate, Newton
20 int nY = 0;      // Y coordinate, Newton
21 float mY = 0;    // Y coordinate, apples
22 int mX = 15;     // X coordinate, apples
23 float mV = 0;    // Y speed, apples
24 float mA = 0.05; // Y acceleration, apples
25 int p = 0;       // Points
26
27 void setup() {
28   size(400, 400);
29   nY = height - 25;
30 }
31
32 void draw() {
33   background(200);
34
35   // Apple's movement
36   mV = mV + mA;
37   mY = mY + mV;
38   if (mY > height) {
```

```

39     mY = 15;
40     mX = int(random(width - 20));
41     mV = 0;
42 }
43
44 fill(255);
45
46 // Collision detection
47 if (mY + 10 > nY && mY - 10 < nY + 20) {
48     if (mX + 10 > nX && mX - 10 < nX + 20) {
49         fill(255, 0, 0);
50         // If collision increase the points
51         p = p + 1;
52     }
53 }
54
55 ellipse(mX, mY, 20, 20);
56 rect(nX, nY, 20, 20);
57
58 // Show the points on the screen
59 fill(0);
60 text("Hits: " + p, 3 * width / 4, 20); // Text to the right on the screen
61 }
62
63 void keyPressed() {
64     // Increase the coordinates in 3 pixeles
65     if (keyCode == RIGHT) {
66         nX = nX + 3;
67     }
68     // Decrease the coordinates in 3 pixeles
69     if (keyCode == LEFT) {
70         nX = nX - 3;
71     }
72     // Limit the X coordinates
73     if (nX < 0) {
74         nX = 0;
75     }
76     if (nX > width - 20) {
77         nX = width - 20;
78     }
79 }

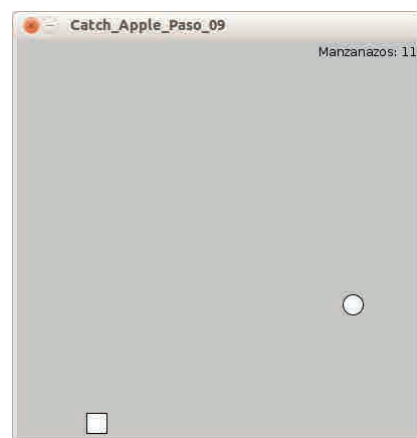
```

Nuevos comandos:

- `text(text, x, y)`: escribe un texto en la pantalla en las coordenadas `x` e `y`.

58

Hemos declarado una nueva variable `p` que se incrementa en una unidad cada vez que se detecta una colisión. La última cosa que hacemos en `draw()` es mostrar el contador de los puntos en la esquina superior derecha.



Uoops, error

Te habrás dado cuenta que tu programa ahora mismo está contabilizando puntos de más. Cada vez que la manzana cae sobre la cabeza de Newton... eh... cuando el círculo toca el cuadrado, tu contador sube más o menos 5 puntos. Para corregir esto tienes un par de opciones:

- Vuelve a lanzar la manzana en cuanto se detecte colisión.
- Deja de contar cuando haya colisión, hasta que se lance una nueva manzana.

Para evitar cualquiera de estas dos posibilidades, necesitas una variable de tipo **boolean** donde almacenar el estado de la manzana. ¿Recuerdas qué era una variable **boolean**? Podemos usar esta variable para decirle al programa si contar puntos o no.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 10:
10  * - the points are being counted extra, fix it
11  * - you can achieve this in two different ways:
12  *   + you can throw the apple again as soon as you detect collision, or
13  *   + stop counting once there is a collision and until a new apple
14  *     falls from the tree
15  * - you will need a 'boolean' variable to check whether to add points or
16  *   not, you will have to reset it to 'true' each time a new apple
17  *   falls from the top of the tree
18  * - los puntos se estan contabilizando extra, corrígelo
19  *
20  * (c) 2013 D. Cuartielles, Arduino Verktad, Sweden
21  */
22
23  int nX = 0;      // X coordinate, Newton
24  int nY = 0;      // Y coordinate, Newton
25  float mY = 0;    // Y coordinate, apples
26  int mX = 15;     // X coordinate, apples
27  float mV = 0;    // Y speed, apples
28  float mA = 0.05; // Y acceleration, apples
29  int p = 0;       // Points
30  boolean pCount = true; // Check whether to count points or not
31
32  void setup() {
33    size(400, 400);
34    nY = height - 25;
35  }
36
```

```

37 void draw() {
38     background(200);
39
40     // Apple's movement
41     mV = mV + mA;
42     mY = mY + mV;
43     if (mY > height) {
44         mY = 15;
45         mX = int(random(width - 20));
46         mV = 0;
47         // When throwing a new apple it will be possible
48         // to start counting points again
49         pCount = true;
50     }
51
52     // Collision detection
53     if (mY + 10 > nY && mY - 10 < nY + 20) {
54         if (mX + 10 > nX && mX - 10 < nX + 20) {
55             fill(255, 0, 0);
56             // If collision increase the points
57             if (pCount) p = p + 1;
58             pCount = false; // Whenever you make it at this point, do not
59                             // count any more points
60         }
61     }
62
63     ellipse(mX, mY, 20, 20);
64     rect(nX, nY, 20, 20);
65
66     // Show the points on the screen
67     fill(0); // Text color
68     text("Hits: " + p, 3 * width / 4, 20); // Text aligned to the right
69 }
70
71 void keyPressed() {
72     // Increase the coordinates in 3 pixels
73     if (keyCode == RIGHT) {
74         nX = nX + 3;
75     }
76     // Decrease the coordinates in 3 pixels
77     if (keyCode == LEFT) {
78         nX = nX - 3;
79     }
80     // Limit the X coordinates
81     if (nX < 0) {
82         nX = 0;
83     }
84     if (nX > width - 20) {
85         nX = width - 20;
86     }
87 }

```

Nuevos comandos:

- `if( boolean )`: comprueba si `boolean` es `true`. Puedes también comprobar si es falso escribiendo `if( !boolean )`.

Ahora prueba el programa para ver cómo cuenta un único punto por manzana. Esto es porque sólo añadimos puntos cuando la variable `boolean pCount` es `true` y cambia a `false` justo después de contar el punto. Cuando la manzana es relanzada, `pCount` se vuelve a poner en `true`.

Y el tiempo empieza

El objetivo del juego es recibir tantas manzanas como sea posible en un periodo de tiempo determinado. Para poder contabilizar esto, sólo necesitamos un contador de tiempo, que también se muestre en la pantalla, para que los jugadores sepan cuándo ha terminado la partida. La duración óptima del juego es de medio minuto, si es demasiado corto, no habrá forma de conseguir puntos mientras que, si es demasiado largo, se hará aburrido. Para medir el tiempo, puedes usar una función llamada `millis()` que contabiliza los milisegundos que han pasado desde la última vez que se le llamó. De este modo puedes ver si han pasado 30000 milisegundos (30 segundos) para terminar el juego. Para almacenar el tiempo, necesitas una variable de tipo `long`. La función `noLoop()` le dirá a tu programa que termine una vez hayas llegado al final del mismo. Finalmente, usa `text()` para mostrar el tiempo que queda en la pantalla.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 11:
10  * - add a time counter to limit the game duration
11  * - an optimal duration for the game is half a minute, there is a
12  *   function called 'millis()' that counts the milliseconds that
13  *   passed since the last time you called it. In this way you can check if
14  *   30000 milliseconds passed (or 30 seconds) in order to end the game
15  * - to store the time you need a variable of type 'long'
16  * - end the game with the function 'noLoop()'
17  * - use 'text()' to show the remaining time on the screen
18  *
19  * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
20  */
21
22 int nX = 0;      // X coordinate, Newton
23 int nY = 0;      // Y coordinate, Newton
24 float mY = 0;    // Y coordinate, apples
25 int mX = 15;     // X coordinate, apples
26 float mV = 0;    // Y speed, apples
27 float mA = 0.05; // Y acceleration, apples
```

```

28 int p = 0;          // Points
29 boolean pCount = true; // Check whether to count points or not
30 long t = 0;         // Store the time
31
32 void setup() {
33     size(400, 400);
34     nY = height - 25;
35     t = millis(); // Initialize the time counter
36 }
37
38 void draw() {
39     background(200);
40
41     // Apple's movement
42     mV = mV + mA;
43     mY = mY + mV;
44     if (mY > height) {
45         mY = 15;
46         mX = int(random(width - 20));
47         mV = 0;
48         // When throwing a new apple it will be possible
49         // to start counting points again
50         pCount = true;
51     }
52
53     fill(255);
54
55     // Collision detection
56     if (mY + 10 > nY && mY - 10 < nY + 20) {
57         if (mX + 10 > nX && mX - 10 < nX + 20) {
58             fill(255, 0, 0);
59             // If collision increase the points
60             if (pCount) p = p + 1;
61             pCount = false; // Whenever you make it at this point, do not
62                             // count any more points
63         }
64     }
65
66     ellipse(mX, mY, 20, 20);
67     rect(nX, nY, 20, 20);
68
69
70     // Count the time
71     float timer = (millis() - t) / 1000; // Count how much time has passed in seconds
72
73     // GAME OVER
74     if (timer >= 30) { // If time reaches 30 seconds, end the game
75         noLoop();
76     }
77
78     // Show the time on the screen
79     fill(0);
80     text("Time: " + (30 - timer), 10, 20);
81

```

```

82 // Show the points on the screen
83 fill(0);
84 text("Hits: " + p, 3 * width / 4, 20);
85 }
86
87 void keyPressed() {
88 // Increase the coordinates in 3 píxeles
89 if (keyCode == RIGHT) {
90     nX = nX + 3;
91 }
92 // Decrease the coordinates in 3 píxeles
93 if (keyCode == LEFT) {
94     nX = nX - 3;
95 }
96 // Limit the X coordinates
97 if (nX < 0) {
98     nX = 0;
99 }
100 if (nX > width - 20) {
101     nX = width - 20;
102 }
103 }

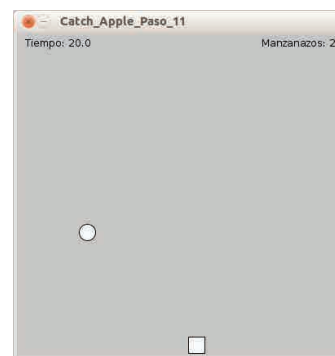
```

Nuevos comandos:

- **long**: este es un tipo de datos para números enteros largos. Es conveniente usarlos cuando manejamos variables temporales, porque estos datos pueden ser muy grandes. ¿Recuerdas cómo explicamos que una variable es un contenedor de datos? Bien, se puede decir que **long** es un contenedor más grande que **int**. Un **int** se queda sin espacio antes que un **long**.
- **noLoop()**: detiene la ejecución continua dentro de **draw**. Para volver a ejecutar el programa tendrás que llamar a **loop()**.

La variable de tiempo **t** se inicializa en el **setup()**. Puesto que **millis()** cuenta la cantidad de milisegundos que el programa se ha ejecutado, **t** tomará un valor cercano a 0. Digamos que, para hacerlo más fácil, es igual a 0. Después de haber dibujado a Newton y la manzana, creamos una nueva variable llamada **timer** que calcula cuanto tiempo ha pasado. Esto se hace con **(millis()-t)/1000**. Cuando han pasado 10 segundos, este cálculo sería: **(10000-0)/1000**. Si **timer** es mayor que 30, llamamos a **noLoop()** para parar el programa. Pero puesto que **timer** no es mayor que 30, utiliza **text()** para mostrar cuanto tiempo queda.

El juego con contador de tiempo se verá como sigue:



Añade imágenes al juego

Incluye imágenes para la manzana, el fondo y Newton. Las puedes crear tú, buscarlas en Internet, o usar las que te adjuntamos. Es importante que las imágenes sean de tipo PNG si quieres que haya transparencia entre las imágenes y el fondo. Ten en cuenta que, al cambiar las formas por imágenes, las proporciones también cambian, por lo que tendrás que hacer encajar esos valores en la parte del programa dedicado a la detección de colisiones.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will not let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 12:
10  *   - include images to show the apple, the background picture and Newton
11  *   - you can make your own images or look for them on the internet
12  *   - it is important that you use PNG images if you want to have transparency
13  *     between the background and the other images
14  *   - check that when you change the images, you are also changing the game's
15  *     proportions, you might have to adjust that in the code
16  *
17  * (c) 2013 D. Cuartielles, Arduino Verktstad, Sweden
18  */
19
20 String[] imFiles = {"fondo.png", "manzana.png", "newton1.png", "newton2.png"};
21 PImage[] im = new PImage[4];
22
23 int nX = 0;      // X coordinate, Newton
24 int nY = 0;      // Y coordinate, Newton
25 float mY = 0;    // Y coordinate, apples
26 int mX = 15;     // X coordinate, apples
27 float mV = 0;    // Y speed, apples
28 float mA = 0.05; // Y acceleration, apples
29 int p = 0;       // Points
30 boolean pCount = true; // Check whether to count points or not
31 long t = 0;      // Store the time
32
33 void setup() {
34   size(400, 400);
35   nY = height - 135;
36   t = millis();
37
38   // Load the images
39   for(int i = 0; i < 4; i = i + 1) {
40     im[i] = loadImage(imFiles[i]);
41   }
42 }
43
44 void draw() {
```

```

45 background(200);
46 image(im[0], 0, 0, width, height); // Background image
47
48 // Apple's movement
49 mV = mV + mA;
50 mY = mY + mV;
51 if (mY > height) {
52     mY = 15;
53     mX = int(random(width - 20));
54     mV = 0;
55     // When throwing a new apple it will be possible
56     // to start counting points again
57     pCount = true;
58 }
59
60 fill(255);
61
62 // Collision detection
63 if (mY + 50 > nY && mY < nY + 135) {
64     if (mX + 40 > nX && mX < nX + 128) {
65         fill(255, 0, 0);
66         // If collision increase the points
67         if (pCount) p = p + 1;
68         pCount = false; // Whenever you make it at this point, do not
69                         // count any more points
70     }
71 }
72
73 image(im[1], mX, mY); // Apple
74 if(pCount) {
75     image(im[2], nX, nY); // Newton looking for apples
76 } else {
77     image(im[3], nX, nY); // Newton got an apple
78 }
79
80 // Count the time
81 float timer = (millis() - t) / 1000; // Count how much time has passed in seconds
82
83 // GAME OVER
84 if (timer >= 30) { // If time reaches 30 seconds, end the game
85     noLoop();
86 }
87
88 // Show the time on the screen
89 fill(0);
90 textSize(20); // Increase the font size
91 text("Time: " + (30 - timer), 10, 20);
92
93 // Show the points on the screen
94 fill(0);
95 textSize(20); // Increase the font size
96 text("Hits: " + p, 3 * width / 4, 20);
97 }

```

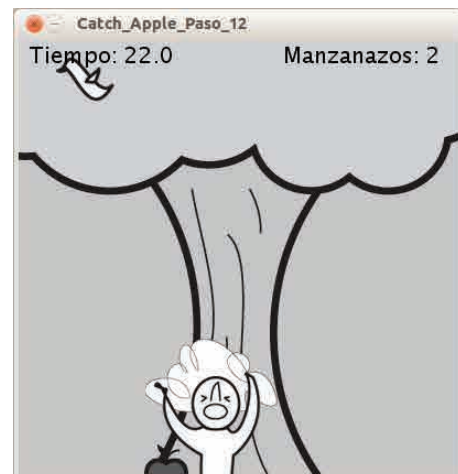
```

98
99 void keyPressed() {
100   // Increase the coordinates in 3 píxeles
101   if (keyCode == RIGHT) {
102     nX = nX + 3;
103   }
104   // Decrease the coordinates in 3 píxeles
105   if (keyCode == LEFT) {
106     nX = nX - 3;
107   }
108   // Limit the X coordinates
109   if (nX < 0) {
110     nX = 0;
111   }
112   if (nX > width - 20) {
113     nX = width - 20;
114   }
115 }

```

Para cargar las imágenes utilizamos el mismo método que en los proyectos anteriores, utilizando arrays. En lugar de cambiar el color de las formas cuando una colisión sea detectada, utilizamos el boolean `pCount` para decidir qué imagen de Newton mostrar.

El resultado final del juego se ve como sigue:



¡Sigue experimentando!

Para mejorar este juego puedes hacer varias cosas:

- Haz tus propias imágenes.
- Crea una pantalla de inicio y que se pase al juego una vez se presione un botón.
- Crea una pantalla final que muestre el resultado una vez se haya terminado el tiempo.
- Haz posible reiniciar el juego cuando el tiempo haya terminado. No te olvides de reiniciar todas las variables necesarias.
- Usa superpoderes científicos, haz que Newton se pueda mover con aceleración y no solo a velocidad constante.