No. Lista: _____ Nombre: _____ Fecha:_____ Firma:_____

El tutorial que realizarás a continuación, y el programa resultante, fueron desarrollados por Eero Prittinen. La traducción al español fue realizada por René Martínez Torres. Su empleo con fines académicos ha sido autorizada por el autor.
Tú ya conoces y manejas algunos conceptos gracias a las prácticas que has realizado anteriormente. En esta ocasión, recibirás l

**Nota:** La versión original contiene un paso adicional, puesto que considera la opción de conectar potenciómetros mediante el empleo de una tarjeta Arduino para controlar el juego con perillas, además de controlarlo mediante el teclado.

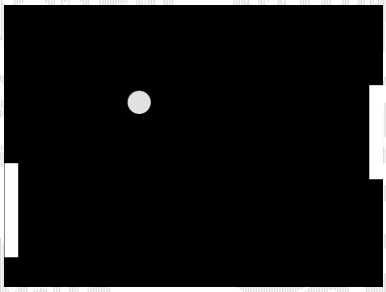
Deberás ir siguiendo el tutorial y creando el código poco a poco. Es muy importante que comprendas lo que estás haciendo. Además, **deberás escribir los comentarios que se te solicitan** en el código, busca la frase "tu comentario aquí". En muchas ocasiones, simplemente transcribirás lo explicado al código, en otras inventarás y escribirás tu explicación. Si el espacio márcado para escrbir tus comentarios es insuficiente, siéntete libre de emplear más renglones para

Recuerda ir probando tu código conforme lo vas construyendo, asegurarte de que todo va bien.

No. Lista: _____ Nombre: _____ Fecha:_____ Firma:_____

# Introducción: Pong con Processing

Por Eero Prittinen



Esta es una guía paso a paso para programar un sencillo juego Pong con Processing. Puedes encontrar información e instrucciones de instalación para Processing en https://processing.org/ .

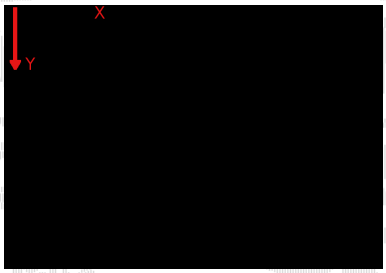
## Paso 1: setup ( ) y draw ( ), operación básica del código de Proceesing

El esquema de funcionamiento básico de Processing se compone de dos funciones, `setup()` y `draw()`. Un programa llama a `setup()` una vez al inicio de la ejecución, y a `draw()` repetidamente durante la ejecución. De forma predeterminada, se llama a `draw()` 60 veces por segundo.

Para tener una ventana para nuestro juego, necesitamos especificar el tamaño de la ventana. Esto se hace llamando al la función `size()`, donde se pueden establecer el ancho y la altura en las dimensiones deseadas. El color de fondo de la ventana se puede configurar con la `función background()` .

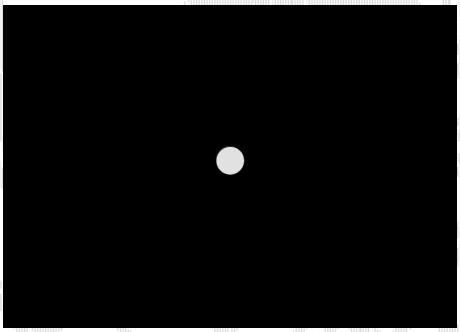
```
void setup(){
  size(800,600);
}

void draw(){
  background(0); //limpia el lienzo
}
```

## Paso 2: coordenadas en Processing




Puede ser bueno introducir en este punto cómo se definen las coordenadas en Processing. Es posible que estés familiarizado con las coordenadas en un ejercicio de geometría típico, donde el origen se define en el medio, el eje `x` positivo se expande hacia la derecha y el eje `y` positivo se expande hacia arriba. Sin embargo, en Processing (y en la programación gráfica en general), el origen está en la esquina superior izquierda, el eje `x` se expande hacia la derecha y el eje `y` se expande hacia abajo. Por lo tanto, podemos expresar las coordenadas de la esquina superior izquierda como `(0,0)`, arriba a la derecha como `(width, 0)`, donde `width` es el ancho de la ventana, abajo a la izquierda como `(0, height)`, donde `height` es el alto de la ventana, y abajo a la derecha como `(width, height)`.

## Paso 3: Pelota Programación Orientada a Objetos



Aquí se explica el concepto básico de un objeto y de una clase en el lenguaje Processing. Se puede encontrar una introducción más detallada (y más extensa) en el sitio web de Processing

Para empezar a programar el juego de Pong, primero definiremos a la pelota. En Processing podemos crear a la pelota como a un objeto. En programación, un objeto es una colección de variables y métodos.

En una forma más práctica de pensar, se puede pensar que un objeto en la programación es como cualquier otro objeto en el mundo real. Por ejemplo, una pelota puede tener un diámetro, peso, color, ubicación en el espacio, velocidad, rebote, y sus métodos podrían ser, por ejemplo, rodar y rebotar. Para nuestro juego solo necesitamos la ubicación (como coordenadas `x` y `y`), la velocidad (como componentes `x` y `y`), diámetro y color.

No. Lista: _____ Nombre: _____ Fecha:_____ Firma:_____

Para crear un objeto, necesitamos tener una plantilla. En Processing a esto se le llama **Clase** (`class`) . Una clase es la descripción del objeto, y contiene las variables y funciones incluidas en el objeto. En cierto sentido, la relación entre clase y objeto se puede ver de modo que la clase son los dibujos y planos de una casa, y el objeto es la casa completa en la que se puede vivir.

La clase también incluirá una función especial llamada **constructor**, que nos permite crear objetos a partir de la clase. Con nuestra pelota tomaremos la ubicación y el diámetro como variables del constructor, para que puedan definirse cuando se cree el objeto "pelota". La velocidad se establece en cero por defecto y el color en blanco (225 como escala de grises). Estos parámetros pueden modificarse después de que se haya creado el objeto pelota a partir de la clase `Pelota`.

```
  background(0); //limpia el lienzo
}

class Pelota {
  float x; // tu comentario aquí
  float y; // tu comentario aquí
  float velocidadX; // tu comentario aquí
  float velocidadY; // tu comentario aquí
  float diametro; // tu comentario aquí
  color c; // tu comentario aquí

  // Método constructor
  Pelota(float tempX, float tempY, float tempDiametro) {
    x = tempX; /* tu comentario aquí                */
    y = tempY; /* tu comentario aquí                */
    diametro = tempDiametro; /*                      */
    velocidadX = 0; // tu comentario aquí
    velocidadY = 0; // tu comentario aquí
    c = (225); // tu comentario aquí
  } // constructor Pelota
} // clase Pelota
```

Necesitamos definir algunos métodos para que la pelota pueda hacer cosas. En nuestro caso, necesitamos poder moverla y dibujarla en la ventana. Para ello definimos las funciones de movimiento y visualización. La función de movimiento toma la ubicación actual de la pelota, y le agrega la velocidad, por separadamente para los componentes `x` y `y`. La función de visualización establece el color que se utilizará, y dibuja un círculo en la ubicación de la pelota.

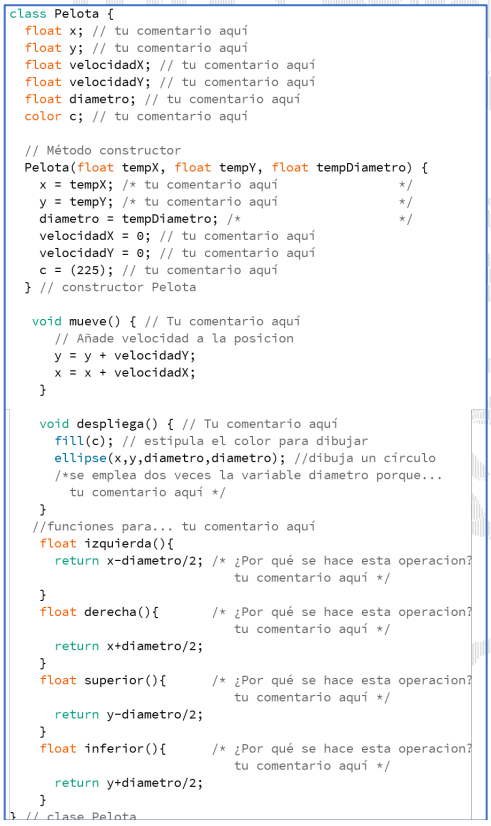
```
   void mueve() { // Tu comentario aquí
      // Añade velocidad a la posicion
      y = y + velocidadY;
      x = x + velocidadX;
    }

    void despliega() { // Tu comentario aquí
      fill(c); // estipula el color para dibujar
      ellipse(x,y,diametro,diametro); //dibuja un círculo
      /*se emplea dos veces la variable diametro porque...
        tu comentario aquí */
  }

} // clase Pelota
```

No. Lista: _____ Nombre: _____ Fecha:_____ Firma:_____

Además, hay funciones que calcularán los valores $x$ y $y$ de los lados superior, inferior, izquierdo y derecho de la pelota. Estas se utilizarán más adelante para la detección de colisiones. Este tipo de pequeñas "funciones auxiliares" hacen que el código principal sea más fácil de escribir, comprender y mantener, ya que lo hace parecer más un lenguaje humano que un conjunto de funciones matemáticas.

```
    }
   //funciones para... tu comentario aquí
    float izquierda(){
      return x-diametro/2; /* ¿Por qué se hace esta operacion?
                              tu comentario aquí */
    }
    float derecha(){       /* ¿Por qué se hace esta operacion?
                              tu comentario aquí */
      return x+diametro/2;
    }
    float superior(){      /* ¿Por qué se hace esta operacion?
                              tu comentario aquí */
      return y-diametro/2;
    }
    float inferior(){      /* ¿Por qué se hace esta operacion?
                              tu comentario aquí */
      return y+diametro/2;
    }
} // clase Pelota
```

Es importante reconocer que se pueden crear objetos, y que estos contienen parámetros y funciones que se pueden utilizar para lograr la operación deseada.



```
class Pelota {
  float x; // tu comentario aquí
  float y; // tu comentario aquí
  float velocidadX; // tu comentario aquí
  float velocidadY; // tu comentario aquí
  float diametro; // tu comentario aquí
  color c; // tu comentario aquí

  // Método constructor
  Pelota(float tempX, float tempY, float tempDiametro) {
    x = tempX; /* tu comentario aquí                */
    y = tempY; /* tu comentario aquí                */
    diametro = tempDiametro; /*                      */
    velocidadX = 0; // tu comentario aquí
    velocidadY = 0; // tu comentario aquí
    c = (225); // tu comentario aquí
  } // constructor Pelota

   void mueve() { // Tu comentario aquí
      // Añade velocidad a la posicion
      y = y + velocidadY;
      x = x + velocidadX;
    }

    void despliega() { // Tu comentario aquí
      fill(c); // estipula el color para dibujar
      ellipse(x,y,diametro,diametro); //dibuja un círculo
      /*se emplea dos veces la variable diametro porque...
        tu comentario aquí */
    }
   //funciones para... tu comentario aquí
    float izquierda(){
      return x-diametro/2; /* ¿Por qué se hace esta operacion?
                              tu comentario aquí */
    }
    float derecha(){       /* ¿Por qué se hace esta operacion?
                              tu comentario aquí */
      return x+diametro/2;
    }
    float superior(){      /* ¿Por qué se hace esta operacion?
                              tu comentario aquí */
      return y-diametro/2;
    }
    float inferior(){      /* ¿Por qué se hace esta operacion?
                              tu comentario aquí */
      return y+diametro/2;
    }
} // clase Pelota
```

Como ahora tenemos la plantilla de una pelota en la clase `Pelota`, vamos a crear a un objeto pelota para nuestro juego.

Primero, definimos a la pelota como a un objeto global fuera de las funciones `setup()` y `draw()`, para acceder a ella en cualquier parte del código. Luego, en `setup()` creamos al objeto "pelota" llamando al constructor de la clase `Pelota` con el operador `new()`. Al constructor le pasamos las coordenadas `x` y `y` deseadas en la pelota, así como el diámetro. Las coordenadas se establecen en el centro de la ventana utilizando los parámetros de `width` y `height` disponibles en Processing.

```
Pelota pelota; // Define a la pelota como a un objeto global

void setup(){
  size(800,600);
   pelota = new Pelota(width/2, height/2, 50); /* crea una nueva pelota al centro
                                                  de la ventana */
}
```

Para poder ver a la pelota, todavía necesitamos hacer que el programa la dibuje en la pantalla.

```
void draw(){
  background(0); //limpia el lienzo
  pelota.despliega(); // Dibuja a la pelota en la ventana
}
```

## Paso 4: Configurar el movimiento de la pelota

El siguiente paso es hacer que la pelota se mueva en el juego. En el paso anterior, ya implementamos el movimiento como un método de la clase `Pelota`, por lo que ahora solo necesitamos usarlo. Esto se puede hacer dando a la pelota algo de velocidad en la configuración. Le daremos a la pelota una velocidad constante de 5 en la dirección del eje `x`, y una velocidad aleatoria entre -3 y 3 en la dirección del eje `y`.

```
void setup(){
  size(800,600);
   pelota = new Pelota(width/2, height/2, 50); /* crea una nueva pelota al centro
                                                  de la ventana */
   pelota.velocidadX = 5; // Dando a la pelota la velocidad en el eje x
   pelota.velocidadY = random(-3,3); // Dando a la pelota la velocidad en el eje y
}
```

Para que la pelota se mueva, necesitamos llamar a su método de movimiento. Esto se hará durante cada ciclo de `draw()`, y queremos hacerlo antes de dibujar a la pelota en la pantalla.

```
void draw(){
  background(0); //limpia el lienzo
  pelota.mueve(); // Calcula la nueva posición de la pelota
  pelota.despliega(); // Dibuja a la pelota en la ventana
}
```

Ahora podemos ver la pelota en la ventana moviéndose hacia la derecha.

## Paso 5: Chocar la pelota contra las paredes

En el paso anterior hicimos que la pelota se moviera. Pero como podemos ver cuando ejecutamos el programa, la pelota no se queda dentro de la ventana, sino que continúa fuera de ella. Para mantener la pelota dentro de la ventana, necesitamos poder detectar cuándo choca con los bordes de la ventana. Esto se puede lograr usando algunas comparaciones básicas y una declaración `if`.

```
if (pelota.derecha() > width) {             /* Tu comentario aquí */
  pelota.velocidadX = -pelota.velocidadX;   /*                    */
}
```

Lo que hace este código es que compara la coordenada `x` derecha de la pelota con la coordenada del borde derecho de la ventana, que es la misma que el ancho de la ventana. Si la coordenada `x` derecha de la pelota es mayor que el ancho de la ventana, significa que la pelota está pasando por el borde de la ventana, y se ejecutará el código entre las llaves.

Allí simulamos rebotar en la pared invirtiendo la velocidad en la dirección `x`. Ahora todo lo que queda por hacer es implementar la misma funcionalidad para detectar colisiones con todos los bordes en la función `draw()`.

```
if (pelota.izquierda() < 0) {               /* Tu comentario aquí */
  pelota.velocidadX = -pelota.velocidadX;   /*                    */
}

if (pelota.inferior() > height) {           /* Tu comentario aquí */
  pelota.velocidadY = -pelota.velocidadY;   /*                    */
}

if (pelota.superior() < 0) {                /* Tu comentario aquí */
  pelota.velocidadY = -pelota.velocidadY;   /*                    */
}
```

Ahora, al ejecutar el código, la pelota debe rebotar en las paredes y permanecer dentro de la ventana.

## Paso 6: Agregar las paletas

Para jugar al Pong, necesitamos paletas para golpear la pelota. Para las paletas, podemos crear el mismo tipo de objeto que hicimos para la pelota, con la diferencia de que en lugar de diámetro, ahora tenemos ancho y alto (`w` y `h`), y dibujamos un rectángulo.

```
class Paleta{

  float x; // tu comentario aquí
  float y; // tu comentario aquí
  float w; // tu comentario aquí
  float h; // tu comentario aquí
  float velocidadY; // tu comentario aquí
  float velocidadX; // tu comentario aquí
  color c; // tu comentario aquí

  // tu comentario aquí
  Paleta(float tempX, float tempY, float tempW, float tempH){
    x = tempX; /* tu comentario aquí                */
    y = tempY; /* tu comentario aquí                */
    w = tempW; /* tu comentario aquí                */
    h = tempH; /* tu comentario aquí                */
    velocidadY = 0; /* tu comentario aquí           */
    velocidadX = 0; /* tu comentario aquí           */
    c=(255); /* tu comentario aquí                  */
  }

  void mueve(){ // Tu comentario aquí
    // Tu comentario aquí
    y += velocidadY;
    x += velocidadX;
  }

  void despliega(){ // Tu comentario aquí
    fill(c); // Tu comentario aquí
    rect(x-w/2,y-h/2,w,h); /* Tu comentario aquí      */
  }                        /*                         */

  //funciones auxiliares
  /* Tu comentario aquí explica para qué sirven */
  float izquierda(){
    return x-w/2;
  }
  float derecha(){
    return x+w/2;
  }
  float superior(){
    return y-h/2;
  }
  float inferior(){
    return y+h/2;
  }
} //clase Paleta
```

No. Lista: _____ Nombre: _____ Fecha:_____ Firma:_____

Las paletas se crean de la misma manera que la pelota. Primero se definen como objetos globales.

```
Paleta paletaIzquierda; /* Tu comentario aquí */
Paleta paletaDerecha;   /* Tu comentario aquí */
```

Luego se crean en la función `setup()` y finalmente se mueven y se muestran en la función `draw()` Aquí creamos paletas de 30 píxeles de ancho y 200 píxeles de alto, y están ubicadas en el medio de la ventana, a 15 píxeles de las paredes.

```
/* Tu comentario aquí */
paletaIzquierda = new Paleta(15, height/2, 30,200);
/* Tu comentario aquí */
paletaDerecha = new Paleta(width-15, height/2, 30,200);
```

Luego agregamos el manejo y visualización del movimiento a la función `draw()`,como hicimos con la pelota.

```
paletaIzquierda.mueve(); // Tu comentario aquí
paletaIzquierda.despliega(); // Tu comentario aquí
paletaDerecha.mueve(); // Tu comentario aquí
paletaDerecha.despliega(); // Tu comentario aquí
```

## Paso 7: mover las paletas, entrada de teclas

Para jugar, los jugadores deben poder mover las paletas con el teclado. Esto se hace usando la función `keyPressed()` y `keyReleased()` en Processing. Estas son funciones independientes y se les llama cuando se presiona o suelta una tecla en el teclado. En ellas, podemos usar la instrucción `if` para determinar qué tecla se presionó. Cuando se presiona la tecla, configuramos la velocidad de la paleta en consecuencia, y cuando se suelta, detendremos la paleta estableciendo su velocidad en 0.

```
} // draw()

/* Tu comentario aquí
    Explica con tus palabras cómo funciona el código
    ¿Por qué se cambia la velocidad ?                */
void keyPressed(){
  if(keyCode == UP){
    paletaDerecha.velocidadY=-3;
  }
  if(keyCode == DOWN){
    paletaDerecha.velocidadY=3;
  }
  if(key == 'a'){
    paletaIzquierda.velocidadY=-3;
  }
  if(key == 'z'){
    paletaIzquierda.velocidadY=3;
  }
}

/* Tu comentario aquí
    Explica con tus palabras cómo funciona el código
    ¿Por qué se cambia la velocidad ?                */
void keyReleased(){
  if(keyCode == UP){
    paletaDerecha.velocidadY=0;
  }
  if(keyCode == DOWN){
    paletaDerecha.velocidadY=0;
  }
  if(key == 'a'){
    paletaIzquierda.velocidadY=0;
  }
  if(key == 'z'){
    paletaIzquierda.velocidadY=0;
  }
}

class Pelota {
```

Ahora deberías poder mover las paletas hacia arriba y hacia abajo con las teclas `A` y `Z`, así como con las teclas de flecha hacia arriba y hacia abajo. Nota que las paletas pueden moverse fuera de la ventana hacia arriba y hacia abajo. Para evitar esto, podemos crear el mismo tipo de detección de colisión que para la pelota, en `draw()` Ahora, en lugar de invertir la dirección del movimiento de la paleta, establecemos el valor para que permanezca en la posición máxima en la que la paleta todavía está dentro de la ventana. De esta forma, si la paleta se sale por la ventana, se devolverá inmediatamente al interior.

```
/* Tu comentario aquí para el conjunto de instrucciones siguiente */
if (paletaIzquierda.inferior() > height) {         /* Tu comentario aquí */
  paletaIzquierda.y = height-paletaIzquierda.h/2;  /* Tu comentario aquí */
}

if (paletaIzquierda.superior() < 0) {      /* Tu comentario aquí */
  paletaIzquierda.y = paletaIzquierda.h/2; /* Tu comentario aquí */
}

if (paletaDerecha.inferior() > height) {      /* Tu comentario aquí */
  paletaDerecha.y = height-paletaDerecha.h/2; /* Tu comentario aquí */
}

if (paletaDerecha.superior() < 0) {            /* Tu comentario aquí */
  paletaDerecha.y = paletaDerecha.h/2;         /* Tu comentario aquí */
}
// draw()
```

## Paso 8: Chocando la pelota y la raqueta

Para finalmente hacer que el juego sea funcional, necesitamos agregar la capacidad de bloquear la pelota con la paleta. Para ello ,necesitamos detectar la colisión entre la pelota y las paletas. Esto se hace en la función `draw()` detectando primero cuando la coordenada `x` del borde derecho o izquierdo de la pelota está detrás de la coordenada del borde de la paleta, y luego determinando si la pelota está en el área de la paleta. Aquí, se emplea operador lógico `&&` en la instrucción `if`, lo que significa que todas las condiciones deben ser verdaderas para que la instrucción `if` se cumpla. En el caso de colisión, la pelota simplemente rebota en la paleta, ya que su velocidad en la dirección del eje `x` se invierte.

```
  // Si la pelota pasa detrás de la paleta
  // Y si la pelota está en él area de la paleta (entre la parte superior e inferior de la paleta
  // rebota a la pelota en la otra dirección

  if ( pelota.izquierda() < paletaIzquierda.derecha() && pelota.y > paletaIzquierda.superior() && pelota.y < paletaIzquierda.inferior()){
    pelota.velocidadX = -pelota.velocidadX;
  }

  if ( pelota.derecha() > paletaDerecha.izquierda() && pelota.y > paletaDerecha.superior() && pelota.y < paletaDerecha.inferior()) {
    pelota.velocidadX = -pelota.velocidadX;
  }
} // draw()
```

## Paso 9: puntuación e inicio de un nuevo juego

Para que el juego sea más interesante, podemos agregar una forma de realizar un seguimiento de la puntuación. Para hacer esto, necesitamos detectar cuándo la pelota pasa por la paleta y llega a la pared. Recuerda que ya tenemos el código para esto en el la función `draw()`.

```
if (pelota.derecha() > width) {
  pelota.velocidadX = -pelota.velocidadX;
}

if (pelota.izquierda() < 0) {
  pelota.velocidadX = -pelota.velocidadX;
}
```

Actualmente, la colisión con las paredes izquierda y derecha hace que la pelota rebote y cambie su dirección de movimiento. Ahora, en cambio, queremos agregar un punto al jugador y devolver la pelota al medio y comenzar el juego nuevamente. Para mantener la puntuación, definiremos dos variables enteras globales y estableceremos el valor inicial en 0.

```
int puntuacionIzquierda = 0; /* Tu comentario aquí */
int puntuacionDerecha = 0;   /* Tu comentario aquí */

void setup(){
```

Cuando ocurre la colisión con la pared, queremos aumentar la puntuación del oponente.

```
  if (pelota.derecha() > width) {             /* Tu comentario aquí */
    puntuacionIzquierda = puntuacionIzquierda + 1; /* Tu comentario aquí */
    pelota.velocidadX = -pelota.velocidadX;   /*                    */
  }

  if (pelota.izquierda() < 0) {               /* Tu comentario aquí */
    puntuacionDerecha = puntuacionDerecha + 1; /* Tu comentario aquí */
    pelota.velocidadX = -pelota.velocidadX;   /*                    */
  }
```

En lugar de cambiar su dirección, también queremos devolver la pelota al centro de la pantalla para comenzar una nueva ronda.

```
if (pelota.derecha() > width) {             /* Tu comentario aquí */
  puntuacionIzquierda = puntuacionIzquierda + 1; /* Tu comentario aquí */
  pelota.x = width/2; /* Tu comentario aquí */
  pelota.y = height/2;
  pelota.velocidadX = -pelota.velocidadX;   /*                    */
}

if (pelota.izquierda() < 0) {               /* Tu comentario aquí */
  puntuacionDerecha = puntuacionDerecha + 1; /* Tu comentario aquí */
  pelota.x = width/2; /* Tu comentario aquí */
  pelota.y = height/2;
  pelota.velocidadX = -pelota.velocidadX;   /*                    */
}
```

Para mantener la puntuación, mostraremos la puntuación en la parte superior de la ventana. Esto se implementará al final de la función `draw()`. Primero necesitamos establecer las reglas para escribir texto usando `textSize()` y `textAling ()`.

```
  textSize(40);
  textAlign(CENTER);

} // draw()
```

La puntuación se puede mostrar en la parte superior de la pantalla.

```
  textSize(40);
  textAlign(CENTER);
  text(puntuacionDerecha, width/2+30, 30); // Puntuación del lado derecho
  text(puntuacionIzquierda, width/2-30, 30); // Puntuación del lado izquierdo

} // draw()
```

## Paso 10: controla la dirección de rebote de la pelota

Para hacer el juego más interesante, podemos hacer que la pelota rebote en diferentes direcciones desde las paletas. Esto se puede hacer alterando la velocidad y la dirección `y` de la pelota cuando golpea a la paleta. En lugar de hacerlo aleatorio (lo que puedes hacer si lo deseas), podemos usar la posición de la paleta donde golpeó la pelota, para calcular la nueva velocidad `y` para la pelota.

```
// Si la pelota pasa detrás de la paleta
// Y si la pelota está en él area de la paleta (entre la parte superior e inferior de la paleta
// rebota a la pelota en la otra dirección

if ( pelota.izquierda() < paletaIzquierda.derecha() && pelota.y > paletaIzquierda.superior() && pelota.y < paletaIzquierda.inferior()){
  pelota.velocidadX = -pelota.velocidadX;
  pelota.velocidadY = pelota.y - paletaIzquierda.y;
}

if ( pelota.derecha() > paletaDerecha.izquierda() && pelota.y > paletaDerecha.superior() && pelota.y < paletaDerecha.inferior()) {
  pelota.velocidadX = -pelota.velocidadX;
  pelota.velocidadY = pelota.y - paletaDerecha.y;
}
```

Lo que se hace aquí es que calculamos la distancia del punto donde la bola golpea la paleta desde el centro de la paleta, y la usamos como una nueva velocidad `y`. Puedes ejecutar esto y observar que la bola puede obtener una velocidad `y` muy alta en esta configuración, en cambio, nos gustaría limitar la velocidad máxima a la que puede llegar la pelota, digamos, 10 (y -10). Esto se puede lograr usando la función `map()`. Básicamente, nos permite mapear el valor de una escala a otra sin tener que pensar en las matemáticas nosotros mismos.

```
if ( pelota.izquierda() < paletaIzquierda.derecha() && pelota.y > paletaIzquierda.superior() && pelota.y < paletaIzquierda.inferior()){
   pelota.velocidadX = -pelota.velocidadX;
   /* Tu comentario aqui. Explica con tus palabras cómo funciona la siguiente instrucción */
   pelota.velocidadY = map(pelota.y - paletaIzquierda.y, -paletaIzquierda.h/2, paletaIzquierda.h/2, -10, 10);
}

if ( pelota.derecha() > paletaDerecha.izquierda() && pelota.y > paletaDerecha.superior() && pelota.y < paletaDerecha.inferior()) {
  pelota.velocidadX = -pelota.velocidadX;
  /* Tu comentario aqui. Explica con tus palabras cómo funciona la siguiente instrucción */
  pelota.velocidadY = map(pelota.y - paletaDerecha.y, -paletaDerecha.h/2, paletaDerecha.h/2, -10, 10);
}

textSize(40);
```