

Programa videojuegos con Processing desde cero

Juan López Rubio lopez@ac.upc.edu

Processing (I)

Processing es un lenguaje de programación de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos.

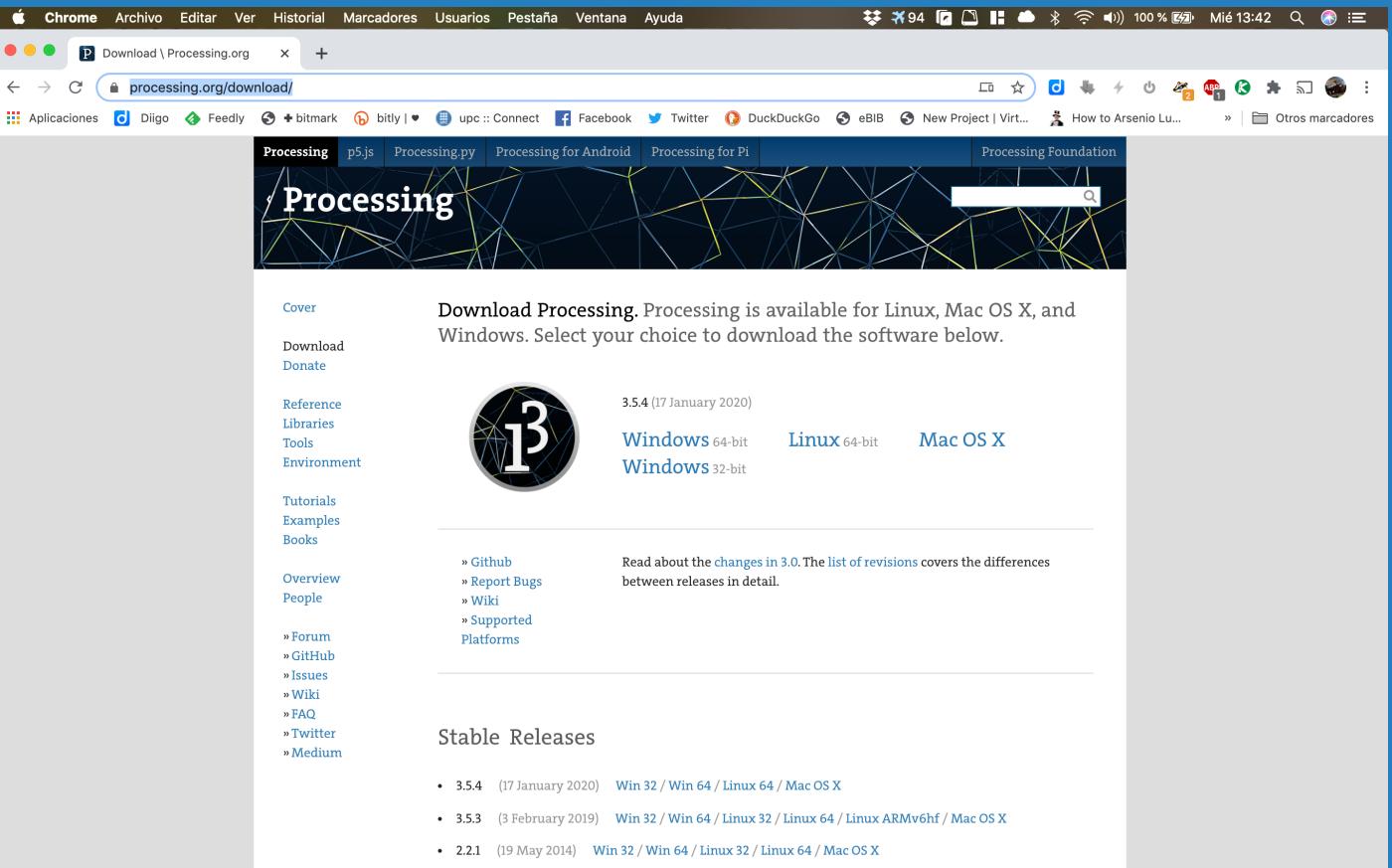
Processing (II)

En su pagina web podemos encontrar la **referencia** (en inglés).

Aquí tenéis la misma información en castellano.

Instalación de Processing (I)

- Vamos a la pagina <https://processing.org/download>
- Seleccionamos nuestro sistema operativo, p.e Windows 64-bit
- Nos descargara un archivo tipo processing-3.5.4-windows64.zip

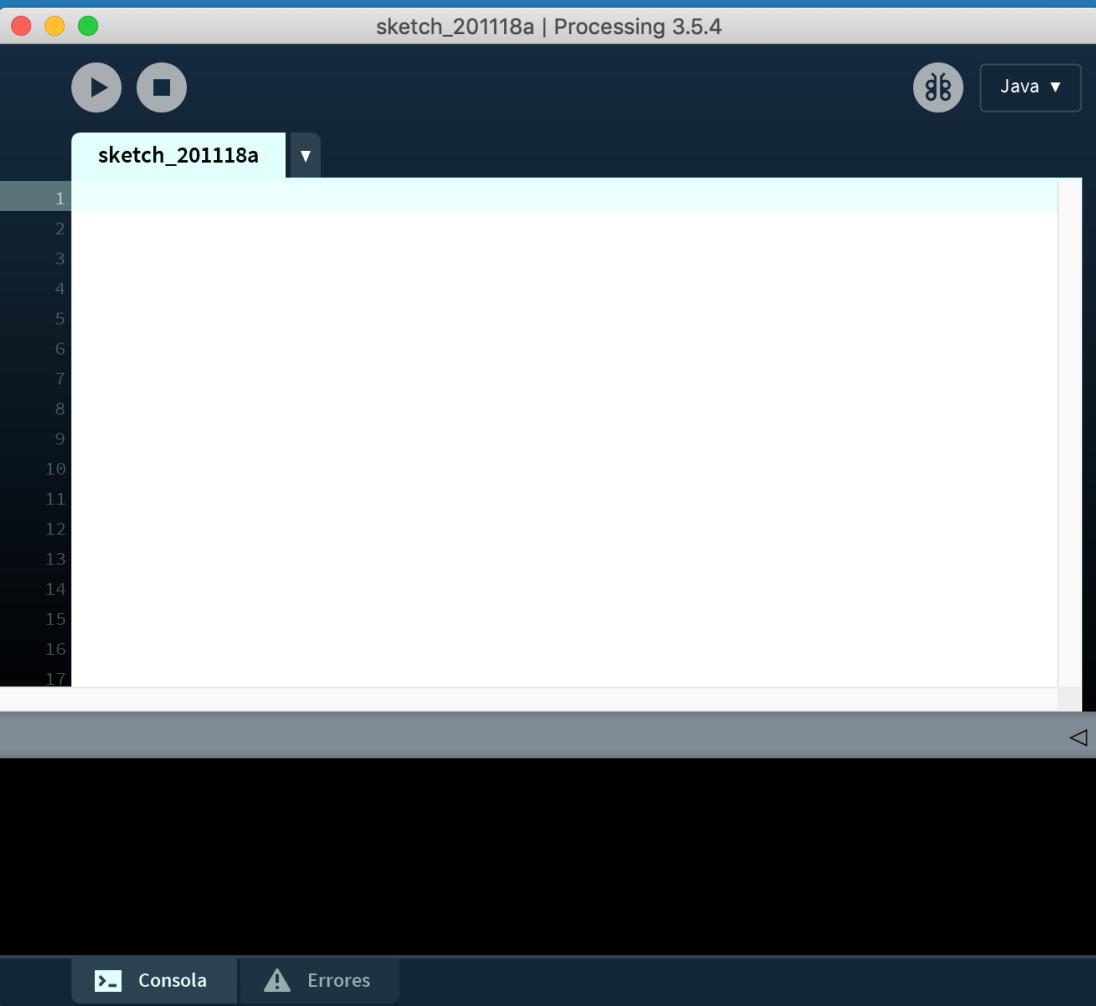


Instalación de Processing (II)

- Lo descomprimimos, p.e en el escritorio y nos crea una carpeta processing-3.5.4
- Dentro encontramos el programa Processing, hacemos click y ya arrancara



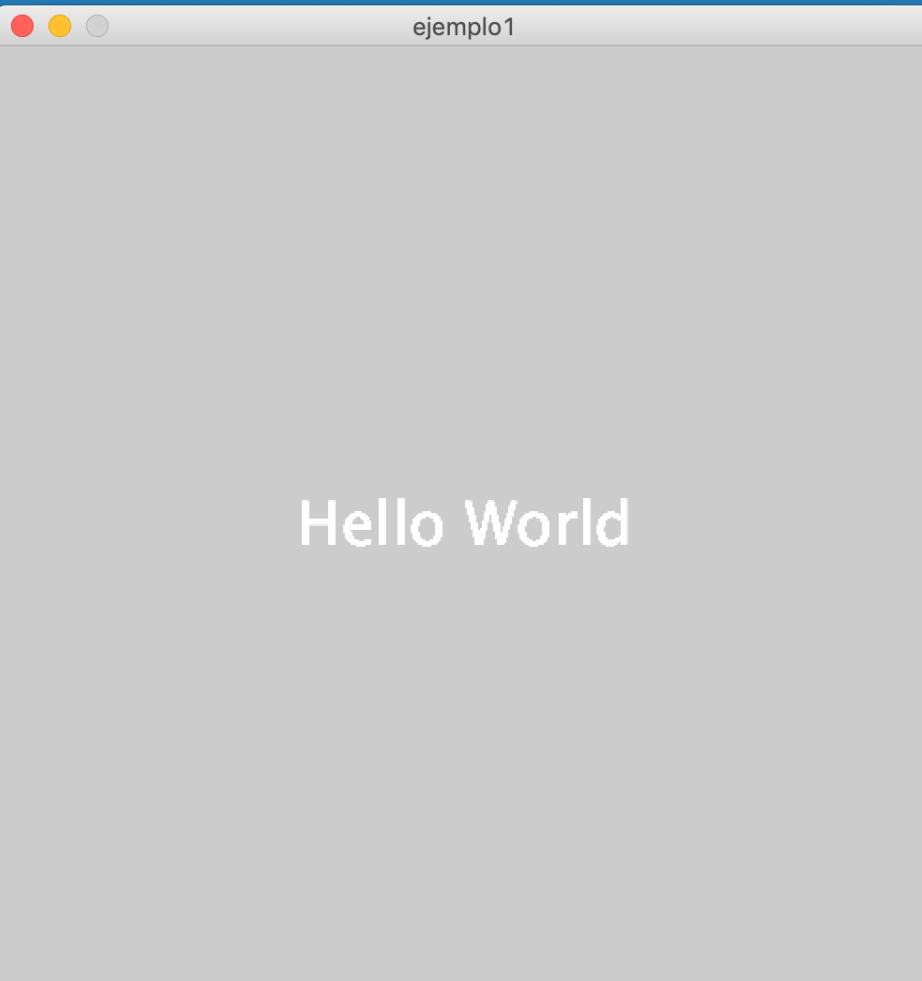
Instalación de Processing (III)



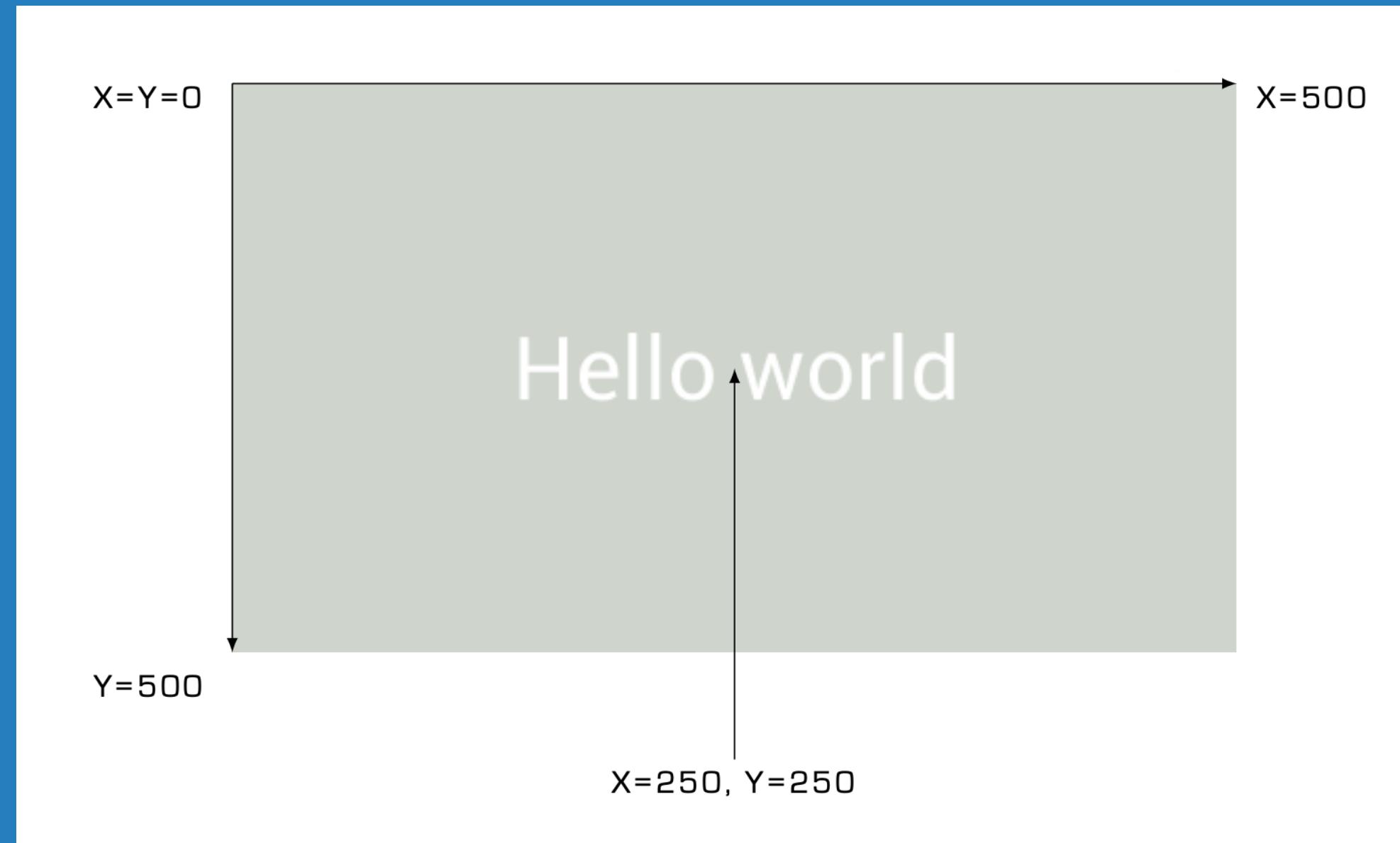
Hello World! (I)

```
void setup() {  
    size(500,500);  
}  
  
void draw() {  
    textSize(32);  
    textAlign(CENTER, CENTER);  
    text("Hello World", 250, 250);  
}
```

Hello World! (II)



Sistema de coordenadas

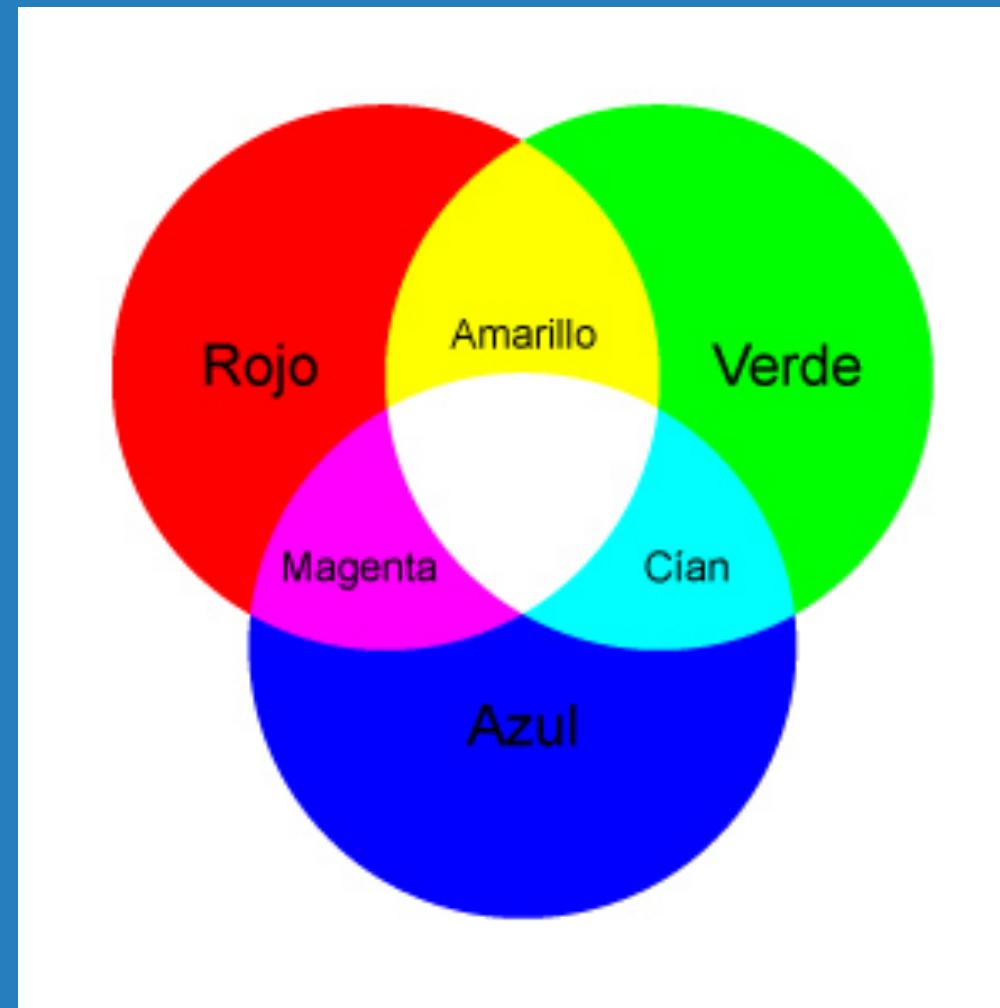


Setup y Draw

```
void setup() {  
    // Se ejecuta una vez al iniciar el programa  
}  
  
void draw() {  
    // Se repite continuamente. 1 vez para dibujar cada fotograma  
}
```

Colores (I)

Cualquier color lo podemos **generar como mezcla de los tres colores primarios: rojo, verde y azul**



Colores (II)

```
fill(150, 255, 100);
```

150 es el porcentaje de rojo

255 es el porcentaje de verde

100 es el porcentaje de azul

Donde el porcentaje esta entre 0 y 255.

Colores (III)

```
background(0,0,0);
```

Controla el color con el que limpiamos la pantalla no con el que pintamos.

También podemos pasar los tres componentes por separado:

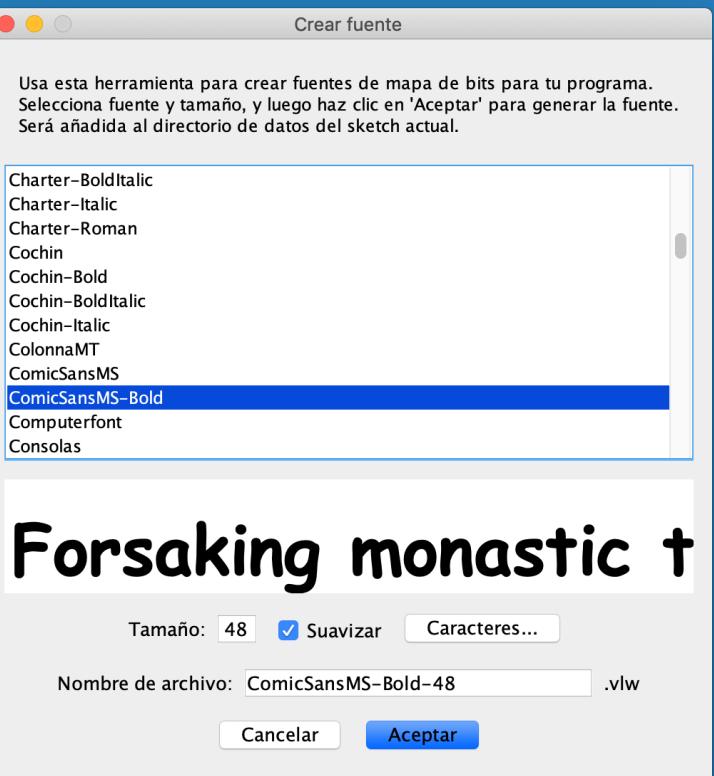
Dibujar texto

```
textSize(tamaño_texto);  
textAlign(CENTER, CENTER);  
text("texto", x, y);
```

LEFT	// Izquierda
CENTER	// Centrado
RIGHT	// Derecha
TOP	// Arriba
CENTER	// Centrado
BOTTOM	// Abajo

Fuentes (I)

Podemos cambiar la fuente pero tendremos que convertirla al formato VLW: Herramientas > Crear fuente...



Fuentes (II)

Una vez creada la fuente solo tenemos que cargarla.

```
PFont font;  
font = loadFont("ComicSansMS-Bold-48.vlw");  
textFont(font, 32);
```

Dibujar líneas

```
void setup() {  
    size(500,500);  
}  
  
void draw() {  
    background(0,0,0);  
    stroke(255,255,255); // Color de la linea  
  
    line(10,10,490,10);  
    line(10,10,10,490);  
    line(490,10,490,490);  
    line(10,490,490,490);  
}
```

Dibujar rectángulos

```
line(x1,y1,x2,y2);           // Dibujar lineas  
rect(x1,x2,ancho,alto);      // Dibujar rectángulos
```

Fíjate que ahora nos pide el ancho y el alto del rectángulo, y no coordenadas.

Borde y relleno

Los rectángulos y otras figuras pueden estar llenos o no. Processing trabaja con tres colores:

background(r,g,b) es color del fondo.

stroke(r,g,b) es color de los bordes.

fill(r,g,b) es color del relleno (también el que usa el texto).

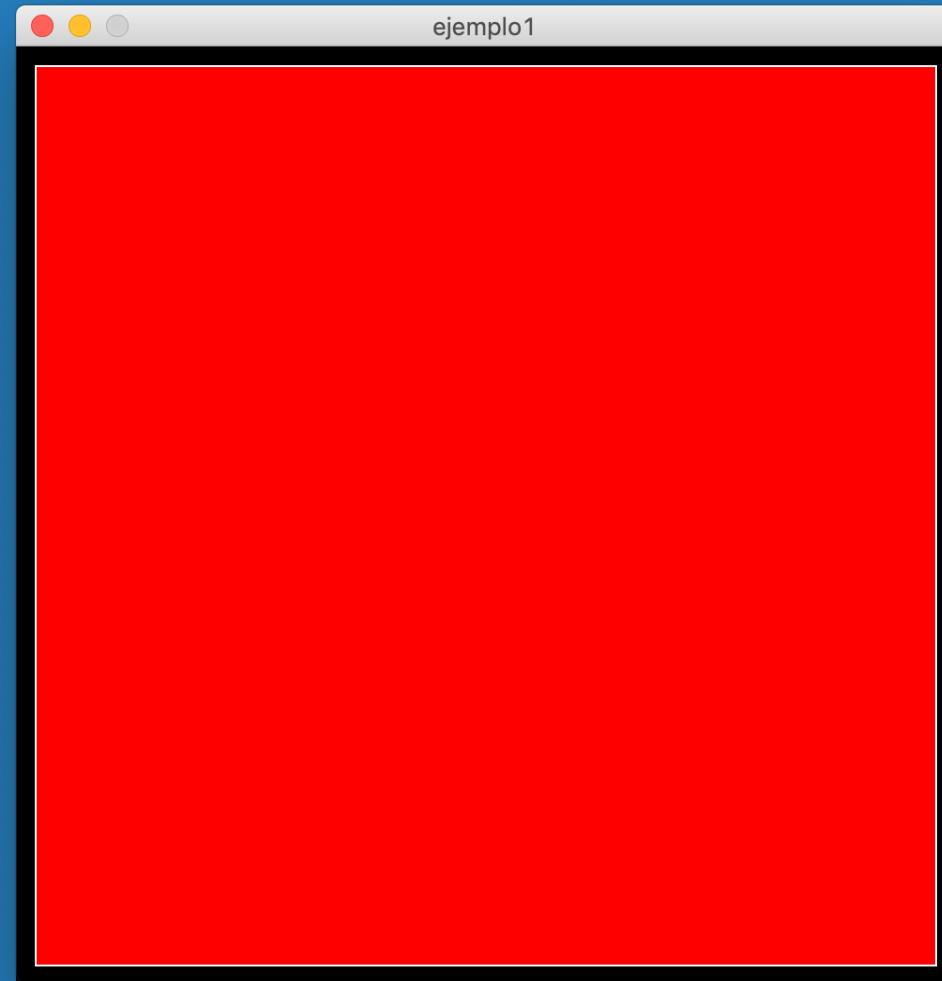
noFill() indica que no queremos llenar la figura.

noStroke() indica que no queremos borde.

Dibujar un rectangulo rojo de 480x480 (I)

```
void setup() {  
    size(500,500);  
}  
  
void draw() {  
    background(0,0,0);  
    stroke(255,255,255);  
    fill(255,0,0);  
    //noFill();  
  
    rect(10,10,480,480);  
}
```

Dibujar un rectangulo rojo de 480x480 (II)



Pantallas Retina

Si tenemos un mac con pantalla retina:

```
void setup() {  
    size(500,500);  
    pixelDensity(displayDensity());  
}
```

Esto hara que los dibujos se vean mejor en nuestra pantalla.

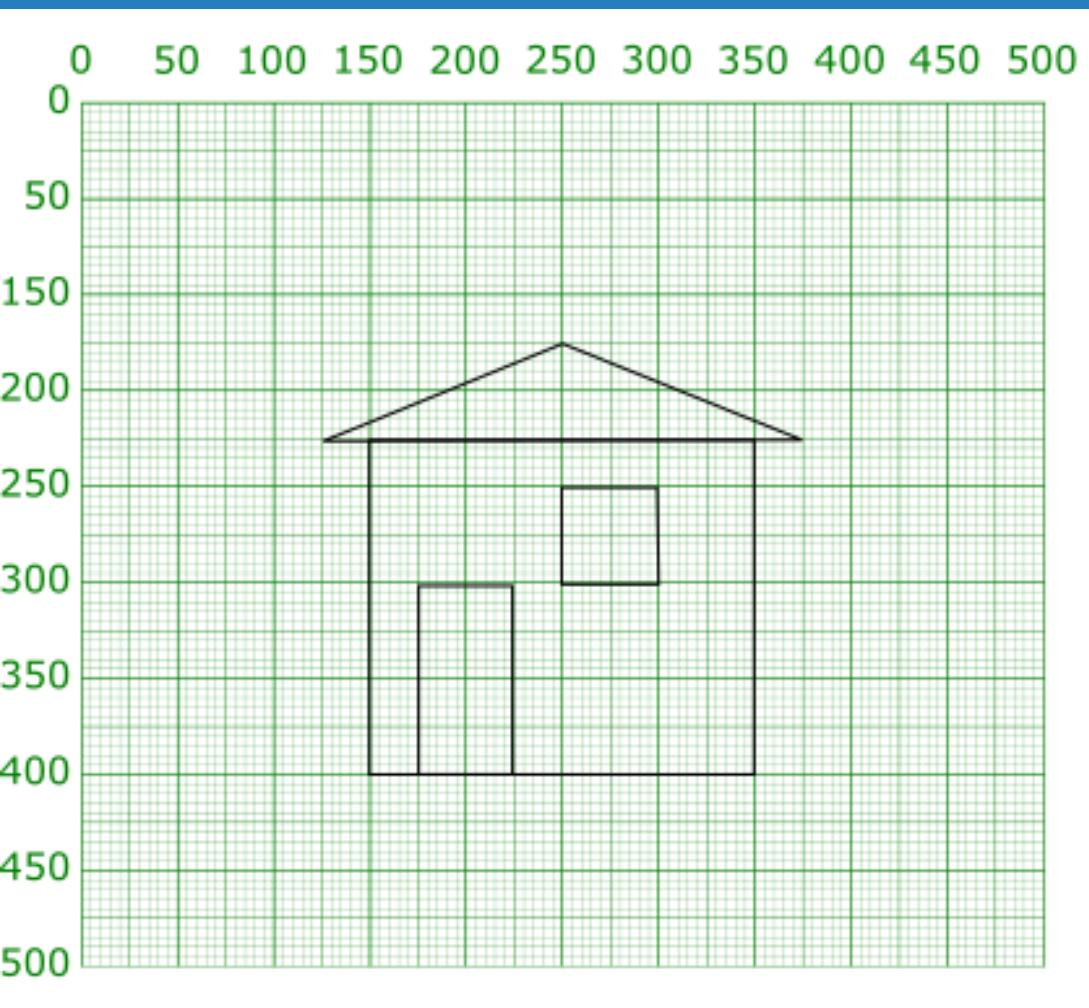
Dibujar puntos y círculos

```
void draw() {  
    strokeWeight(8);  
    point(250,250);  
}
```

```
void draw() {  
    circle(250,250,8);  
}
```

Ejercicio 1

Dibujar esta figura:



Variables

Una variables es una "caja" donde el ordenador puede guardar un dato en la memoria RAM.

Tiene una nombre para identificarla.

Y también un tipo de dato que indica que tipo de información (numérica, letras...) contiene.

```
int miNumero = 12345;  
char miLetra = 'a';  
String nombre = "Juan López"
```

Animaciones

```
void setup() {  
    size(200,200);  
    stroke(255,0,0);  
    frameRate(30); // Queremos 30 frames por segundo  
}  
  
float y = 200;  
  
void draw() {  
    background(0,0,0);  
    line(0,y,width,y);  
    y=y-1;  
}
```

Practicamos las animaciones

- 1. ¿Cómo modificaremos la animación anterior para que la linea vaya de arriba a abajo?**
- 2. ¿Cómo haríamos para que fuera de izquierda a derecha?**
- 3. ¿Y para qué fuera cambiando de color?**

Podemos animar cualquier cosa

```
float x = 0;

void setup() {
    size(500, 500);
}

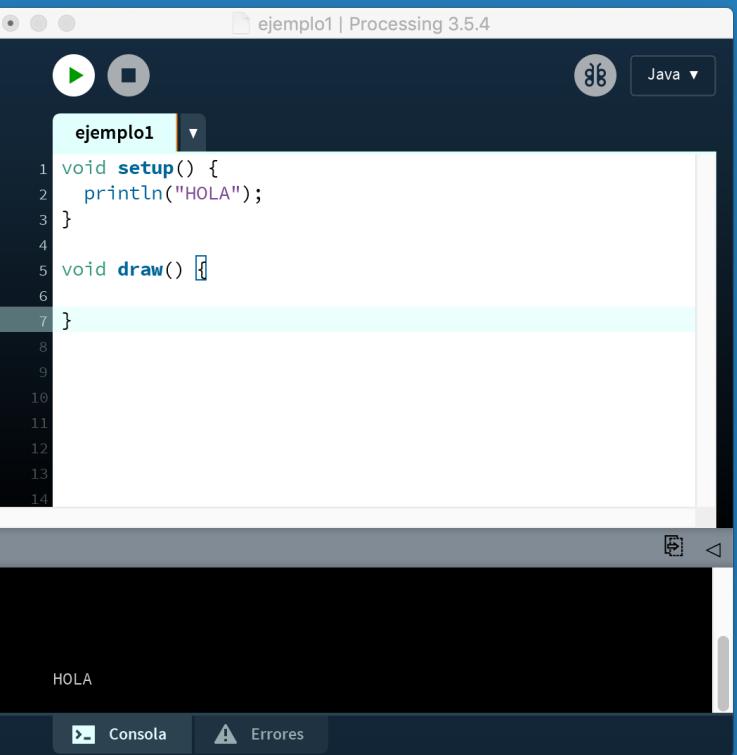
void draw() {
    background(255,255,255);
    noFill();
    rect(x+100, 150, 300, 300); // casa
    line(x+0, 450, x+500, 450); // suelo
    triangle(x+75, 150, x+425, 150, x+250, 50); // tejado
    rect(x+200,350,75,100); // puerta
    rect(x+150,200,75,75); // ventana izq
    rect(x+275,200,75,75); // ventana der

    x = x + 0.5;
}
```

Comunicación por la consola

Podemos enviar mensajes a la consola para ver qué va haciendo nuestro programa.

```
println("HOLA");
```



Mouse

```
void draw()
{
    print(mouseX);
    print(" ");
    println(mouseY);
}
```

En mouseX y mouseY tenemos las coordenadas del ratón.

Ejercicio 2

Hacer un programa que dibuje un rectángulo de 20x70 a la izquierda de la pantalla y a la altura donde este el ratón del usuario.

Solución ejercicio 2

Hacer un programa que dibuje un rectángulo de 20x70 a la altura donde este pulsando el usuario.

```
void setup() {  
    size(500,500);  
}  
  
void draw() {  
    background(0);  
    rect(10, mouseY-35, 20, 70);  
}
```

Funciones

Son trozos de código a los que ponemos un nombre.

```
void hola() {  
    println("Hola!");  
}
```

Los podemos ejecutar o llamar poniendo su nombre con paréntesis al final:

```
hola();
```

Funciones

Tienen dos usos muy claros:

1. Reutilizar código que se repite varias veces por nuestro programa.
2. Descomponer nuestro código en trozos más pequeños para entenderlo mejor y hacerlo más manejable.

Parámetros

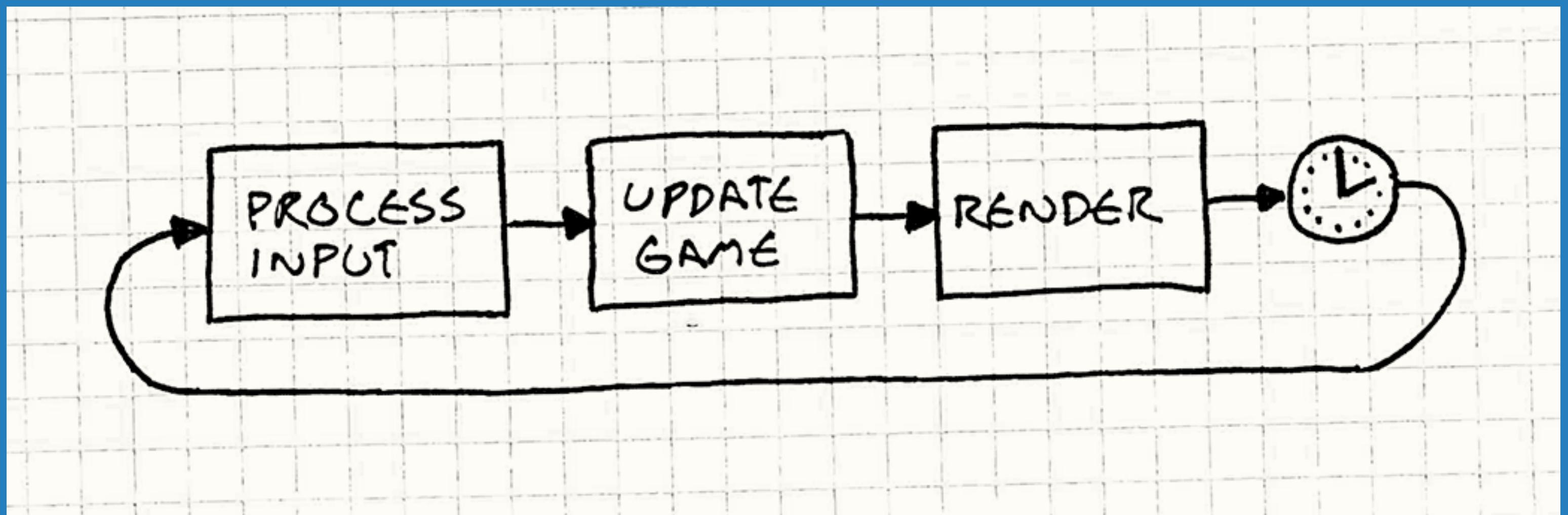
La función no siempre tiene que hacer lo mismo exactamente, le podemos pasar parámetros que modifican su comportamiento:

```
void hola(String nombre) {  
    print("Hola ");  
    print(nombre);  
    println("!");  
}
```

```
hola("Juan");
```

Game Loop (I)

La lógica de un juego siempre sigue el mismo esquema:



Game Loop (II)

```
while (true) {  
    processInput();      // Analizo que entradas (pulsaciones) me da el jugador  
    update();           // Actualiza las variables del juego  
    render();           // Dibujo en función de las variables  
}
```

Y en Processing quedaría:

```
void draw {  
    processInput();  
    update();  
    render();  
}
```

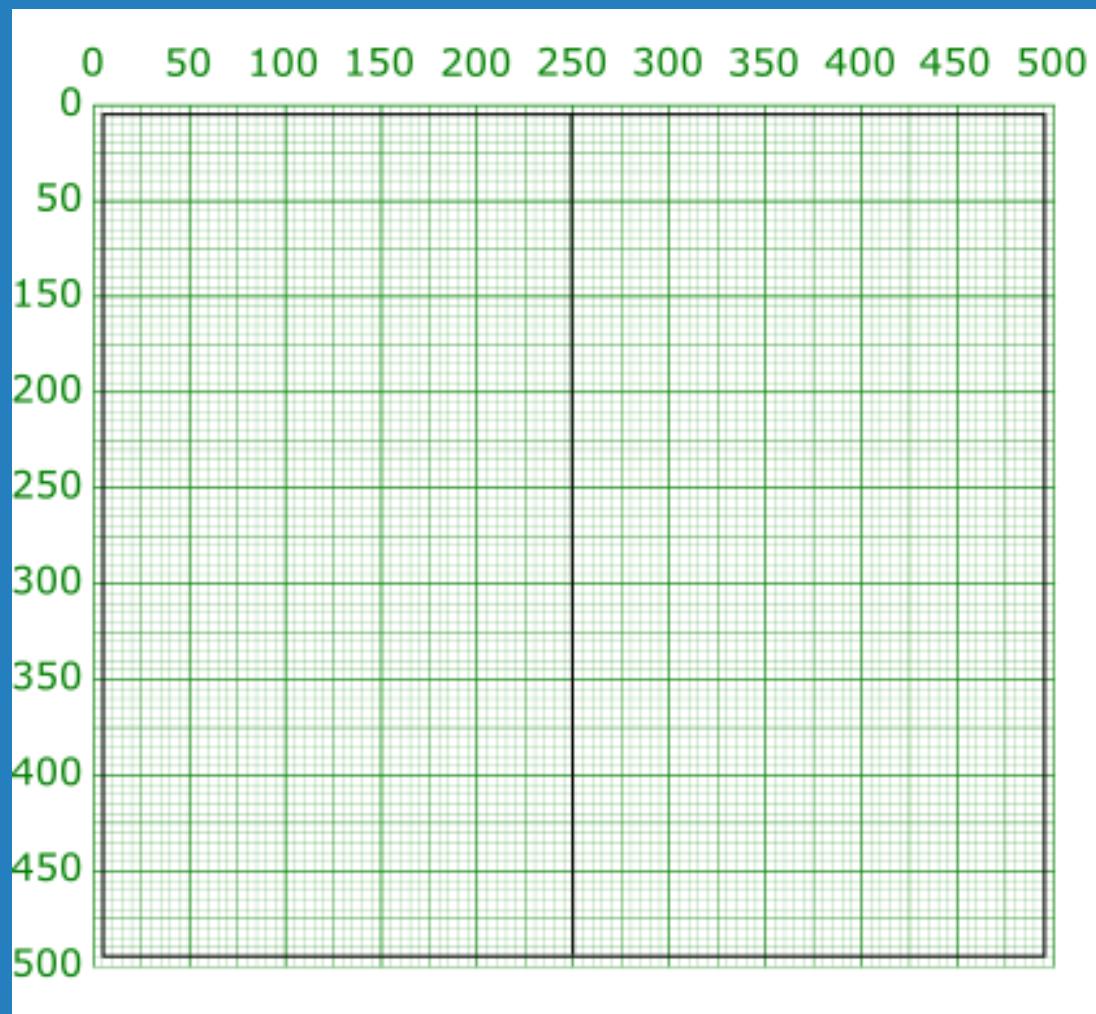
Pong

Pong fue un videojuego de la primera generación de videoconsolas publicado por Atari, creado por Nolan Bushnell y lanzado el 29 de noviembre de 1972. Pong está basado en el deporte de tenis de mesa.



1. Dibujar campo

Hacemos una función dibujarCampo que nos dibuje la siguiente figura:



Solucion 1. Dibujar campo

```
void setup() {  
    size(500,500);  
}  
  
void draw() {  
    dibujarCampo();  
}  
  
void dibujarCampo() {  
    background(0,0,0);  
    stroke(255,255,255);  
    noFill();  
    rect(10,10,480,480);  
    line(250,10,250,490);  
}
```

2. Dibujar palas

Vamos a dibujar las palas en la funcion dibujarPalas. Usaremos dos variables pala1 y pala2 para guardar la fila donde estan la pala1 y la pala2.

Solucion 2. Dibujar palas

```
float pala1 = 250;  
float pala2 = 250;  
  
void dibujarPalas() {  
    fill(255, 255, 255);  
    rect(20, pala1, 30, 70);  
    rect(450, pala2, 30, 70);  
}  
  
void draw() {  
    dibujarCampo();  
    dibujarPalas();  
}
```

3. Mover palas

Vamos a hacer que la pala de la izquierda se mueva verticalmente con el raton en una función que se llame muevePala. Recuerda que en mouseX y mouseY tenemos las coordenadas donde se encuentra el ratón.

3. Solucion 3. Mover palas

```
void muevePala() {  
    palal = mouseY;  
}
```

```
void draw() {  
    muevePala();  
    dibujarCampo();  
    dibujarPalas();  
}
```

3. Añadimos la bola

Creamos una función dibujarBola que pintara la bola en las coordenadas bolaX y bolaY. La bola tiene que tener 10 pixeles.

Más adelante programaremos la lógica de su movimiento.

Solucion 3. Añadimos la bola

```
float bolaX = 250;  
float bolaY = 250;  
  
void dibujarBola() {  
    circle(bolaX, bolaY, 10);  
}
```

Un poco de física

¿Qué es el movimiento?

$$x = x_0 + v * t$$

Y en dos dimensiones:

$$\begin{aligned}x &= x_0 + v_x * t \\y &= y_0 + v_y * t\end{aligned}$$

Un poco de física (II)

Si gestionamos el tiempo en fotogramas o *frames* y actualizamos la posición en cada uno de ellos, por lo tanto el tiempo es 1.

$$x = x + vx;$$

$$y = y + vy;$$

Las variables x e y contienen los valores previos antes de la asignación y los actuales, después.

Números aleatorios

En algunos casos necesitamos que nuestro juego se comporte siempre igual

- `random(n)` devuelve un valor al azar entre 0 y N
- `random(a,b)` devuelve un valor al azar entre a y b-1

4. Movemos la bola

Crearemos la función `iniciaBola` que posiciona una nueva pelota en el centro de la pista con velocidad y dirección aleatoria.

Sugerencia: Una buena velocidad esta entre 1 y 3 pixeles por Frame.

La función `moveBola` actualizará la posición de la pelota en base a su velocidad.

Solucion 4. Movemos la bola

```
void setup() {  
    size(500, 500);  
    iniciaBola();  
}  
  
float velocidadX;  
float velocidadY;  
  
void iniciaBola() {  
    bolaX = 250;  
    bolaY = 250;  
    velocidadX = random(-3,3);  
    velocidadY = random(-3,3);  
}  
  
void mueveBola() {  
    bolaX = bolaX + velocidadX;  
    bolaY = bolaY + velocidadY;  
}
```

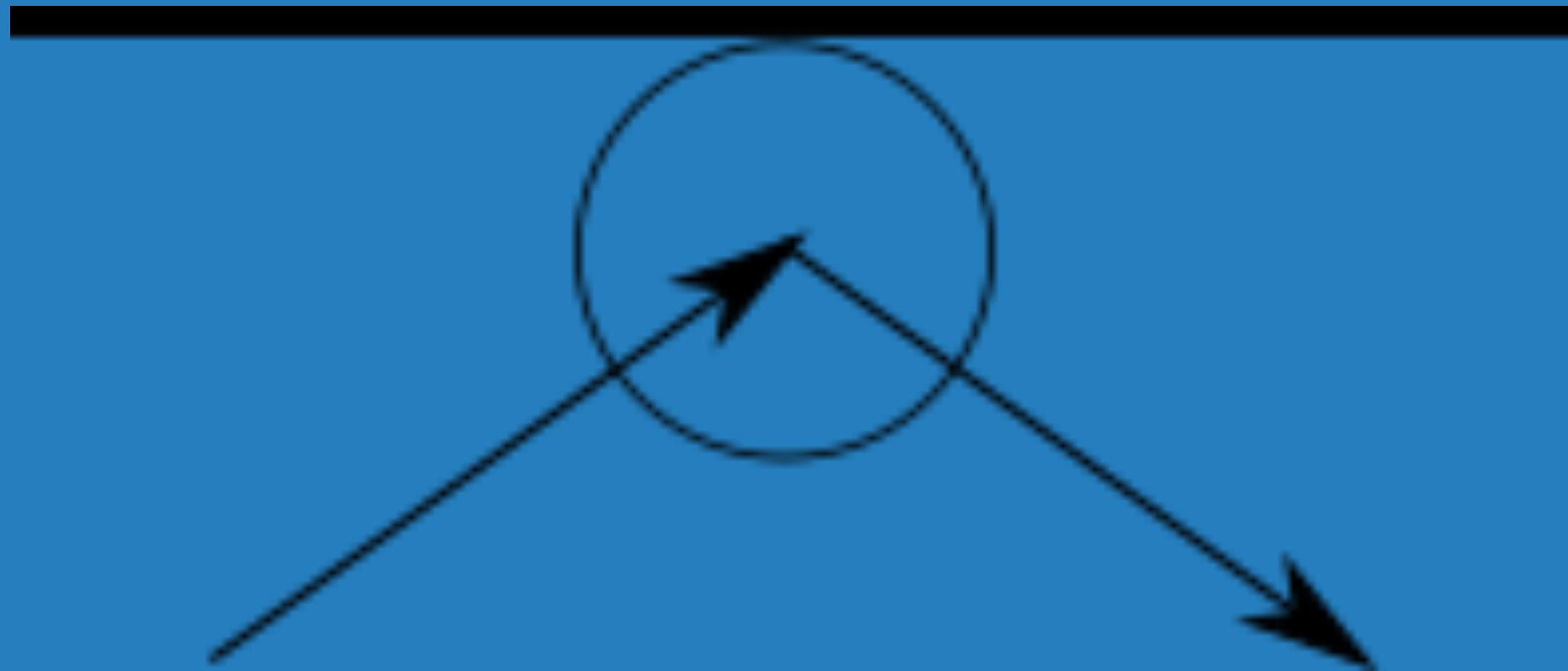
Funciones utiles Processing

- constrain(x ,a ,b) **devuelve el valor x limitado al rango entre a y b**
- map(x ,minOrig,maxOrig,minResul,maxResul)
devuelve el valor x que va entre los valores 'minOrig' y 'maxOrig' mapeado o trasladado sobre un nuevo rango que va entre los valores 'minResul' y 'maxResul'

OPCIONAL. Mejoramos el movimiento de la pala del jugador

Usaremos la función constrain para limitar el movimiento de nuestra pala y que no se pueda salir de los límites del campo (ni la pantalla).

Colisiones y rebotes



$$\begin{aligned} velocidadX_{final} &= velocidadX_{inicial} \\ velocidadY_{final} &= -velocidadY_{inicial} \end{aligned}$$

Condiciones

Processing puede decidir entre ejecutar dos caminos diferentes en base al resultado de un calculo.

```
if(2+5 < 10) {  
    println("El resultado es más pequeño que 10");  
} else {  
    println("El resultado es más grande que 10");  
}
```

5. Rebotar la bola

Vamos a modificar la función `moveBall` para implementar los rebotes en las 4 paredes (recordad que sabemos sus coordenadas porque las hemos dibujado).

Solucion 5. Rebotar la bola

```
void mueveBola() {  
    if(bolaX < 10) {  
        velocidadX = -velocidadX;  
    }  
    if(bolaX > 490) {  
        velocidadX = -velocidadX;  
    }  
    if(bolaY < 10) {  
        velocidadY = -velocidadY;  
    }  
    if(bolaY > 490) {  
        velocidadY = -velocidadY;  
    }  
  
    bolaX = bolaX + velocidadX;  
    bolaY = bolaY + velocidadY;  
}
```

6. Añadimos la puntuación

Implementamos una función dibujarPuntuacion que muestre las puntuaciones de cada jugador almacenadas en las variables score1 y score2.

Solucion 6. Añadimos la puntuación

```
int score1 = 0;  
int score2 = 0;  
  
void dibujarPuntuacion() {  
    textSize(20);  
    text(score1, 50, 50);  
    text(score2, 450, 50);  
}  
  
void draw() {  
    muevePala();  
    mueveBola();  
    dibujarCampo();  
    dibujarPalas();  
    dibujarBola();  
    dibujarPuntuacion(); // Añadimos esta función  
}
```

7. Contar puntos

Cuando la posición de la bola llegue a una de las paredes izquierda o derecha, incrementamos la puntuación del jugador que toque y llamamos a `iniciaBola` para lanzar una nueva bola.

Esto lo haremos en la función `mueveBola`.

Solucion 7. Contar puntos

```
void mueveBola() {  
    if(bolaX < 10) {  
        score2 = score2 + 1;  
        iniciaBola();  
    }  
    if(bolaX > 490) {  
        score1 = score1 + 1;  
        iniciaBola();  
    }  
    if(bolaY < 10) {  
        velocidadY = -velocidadY;  
    }  
    if(bolaY > 490) {  
        velocidadY = -velocidadY;  
    }  
  
    bolaX = bolaX + velocidadX;  
    bolaY = bolaY + velocidadY;  
}
```

8. Rebotar en las palas

Ahora modificaremos la función `mueveBola` para que la bola rebote en las palas.

Tened en cuenta que para que la bola rebote en una pala tienen que pasar tres cosas:

1. Que este justo en la columna antes de la pala
2. Que su fila sea superior a la fila donde empieza la pala (`pala1` o `pala2`)
3. Que su fila sea inferior a la fila donde acaba la pala, que sera la posición de la pala más el tamaño del rectangulo que hemos pintado.

Solucion 8. Rebotar en las palas

```
void mueveBola() {  
    if(bolaX < 10) {  
        score2 = score2 + 1;  
        iniciaBola();  
    }  
    if(bolaX > 490) {  
        score1 = score1 + 1;  
        iniciaBola();  
    }  
    if(bolaY < 10) {  
        velocidadY = -velocidadY;  
    }  
    if(bolaY > 490) {  
        velocidadY = -velocidadY;  
    }  
    if(bolaX < 50 && bolaY > pala1 && bolaY < pala1 + 70) {  
        velocidadX = -velocidadX;  
    }  
    if(bolaX > 450 && bolaY > pala2 && bolaY < pala2 + 70) {  
        velocidadX = -velocidadX;  
    }  
  
    bolaX = bolaX + velocidadX;  
    bolaY = bolaY + velocidadY;  
}
```

9. Inteligencia Artificial

Vamos a implementar una inteligencia artificial muy simple para poder jugar contra la maquina.

La función muevePlayer2 moverá 1 pixel hacia arriba la pala si la pelota esta encima de la pala del jugador o 1 pixel hacia abajo si esta por debajo.

Podemos utilizar la función constrain para mantener la pala dentro del campo de juego.

Solucion 9. Inteligencia Artificial

```
void muevePlayer2() {  
    if(bolaY < pala2) {  
        pala2 = pala2 + 1;  
    }  
    if(bolaY > pala2 + 70) {  
        pala2 = pala2 - 1;  
    }  
}  
  
void draw() {  
    muevePala();  
    mueveBola();  
    muevePlayer2();  
    dibujarCampo();  
    dibujarPalas();  
    dibujarBola();  
    dibujarPuntuacion();  
}
```

Teclado

La variable keyPressed indica cuando se pulsa una tecla y la variable key guarda que tecla se ha pulsado.

```
void draw() {  
    if(keyPressed && key == 'q') {  
        println("Has pulsado la Q");  
    }  
    if(keyPressed && key == 'a') {  
        println("Has pulsado la A");  
    }  
}
```

Objeto String

Processing nos proporciona un **objeto para manipular las cadenas de texto, i.e frases.**

```
String string1 = str('a');           // convirtiendo un caracter
String string2 = "Soy un string";    // a partir de un texto
String string3 = string2 + " más larga"; // concatenando 2 strings
String string4 = str(13);            // convirtiendo un numero
```

```
if(string1.equals(string2)) {
  // Comparo si dos cadenas son iguales
}
```

Retornos

Una función también puede devolver un resultado que calcule:

- para guardarlo en una variable**
- para usarlo en una condición**

```
float precioConIva(float precio, float iva) {  
    return precio+precio*iva/100.0;  
}
```

```
float total = precioConIva(80.25, 12.0);
```

OPCIONAL Simplificar los rebotes

Vamos a implementar una función auxiliar que nos retorne si un numero esta dentro de un rango o no:

```
boolean in_range(int number, int min, int max)
```

Con su ayuda modificaremos mueveBola para que la bola también rebote en las palas.

Musica

Processing puede reproducir sonidos. En principio soporta WAV y MP3 pero el formato MP3 tiene bastantes incompatibilidades por lo que es recomendable convertir los MP3 a WAV. Importaremos el fichero resultante:

Sketch > Importar archivo

Tambien tendremos que añadir la libreria de sonido a nuestro programa:

Sketch > Importar biblioteca... > Añadir biblioteca ..

Buscamos la libreria Sound y pulsamos instalar.

Reproducir musica o sonido

```
import processing.sound.*;  
  
SoundFile musica;  
  
void setup() {  
    musica = new SoundFile(this, "sonido.wav");  
}  
  
void draw() {  
    musica.play();  
}
```

Galaga

Galaga es un videojuego matamarcianos creado en 1981 por la empresa Namco.

Fue diseñado como el sucesor del Galaxian (1979).



1. Añadimos la nave del jugador

Empezamos con una pantalla de 500x500 y un fondo negro.

En este juego vamos a intentar tener las funciones setup y draw organizadas de forma que solo llamen a nuestras funciones crear*, dibujar*, mover*, etc.

Solucion 1.

```
void setup() {  
    size(500,500);  
}
```

```
void draw() {  
    background(0);  
}
```

2. Añadimos la nave del jugador

En una nueva pestaña nave, añadimos una variable playerX que contendrá la posición de la nave, inicialmente estará en la mitad de la pantalla. Programamos dos funciones:

dibujarNave: Dibuja un rectángulo de 50x50 en la fila 440 y en la columna que nos diga la variable playerX.

moverNave: Mueve la nave 5 pixeles a la izquierda o la derecha cuando se toquen las flechas.

Solucion 2 - pestaña Nave.

```
float playerX = 250;

void dibujarNave() {
    rect(playerX-25, 440, 50, 50);
}

void moverNave() {
    if (keyPressed && keyCode == LEFT) {
        playerX = playerX-5;
    }

    if (keyPressed && keyCode == RIGHT) {
        playerX = playerX+5;
    }
}
```

Solucion 2 - pestaña Principal.

```
void setup() {  
    size(500,500);  
}
```

```
void draw() {  
    background(0);  
    moverNave();  
    dibujarNave();  
}
```

Imagenes

**Processing tambien puede cargar una imagen.
Primero la importamos en el proyecto con la opción:**

Sketch > Importar archivo

Cargar y dibujar una imagen

```
PImage img; // Declaramos una variable para la imagen

void setup() {
    size(500,500);
    // Carga la imagen que hemos importado previamente
    img = loadImage("imagen.jpg");
}

void draw() {
    // Muestra la imagen a tamaño real en la posicion 0,0
    image(img, 0, 0);
    // // Muestra la imagen a tamaño 50x50 en la posicion 0,0
    image(img, 0, 0, 50, 50);
}
```

3. Ponemos una imagen a la nave

Vamos a cambiar el rectangulo de la nave por una imagen que añadiremos al proyecto.

- Añadimos una variable PImage nave para cargar la imagen de nuestra nave en la función crearNave.
- Modificamos la función dibujarNave para sustuir el rectangulo por la imagen que hemos cargado previamente.

Solucion 3 - pestaña Nave.

```
float playerX = 250;  
  
PImage nave;  
  
void crearNave() {  
    nave = loadImage("nave.png");  
}  
  
void dibujarNave() {  
    image(nave, playerX-25, 440, 50, 50);  
}  
  
void moverNave() { // No ha cambiado nada respecto al punto anterior  
    if (keyPressed && keyCode == LEFT) {  
        playerX = playerX-5;  
    }  
  
    if (keyPressed && keyCode == RIGHT) {  
        playerX = playerX+5;  
    }  
}
```

Solucion 3 - pestaña Principal.

```
void setup() {  
    size(500,500);  
    crearNave();  
}  
  
void draw() {  
    background(0);  
    moverNave();  
    dibujarNave();  
}
```

4. Añadimos el laser del jugador

En una nueva pestaña Laser, creamos dos variables laserX e laserY para guardar la posicion del disparo laser.

La funcion dibujarLaser dibuja un rectangulo rojo de 5x20. Intentamos que quede centrado horizontalmente.

La funcion moverLaser lo mueve 5 pixeles hacia arriba.

La funcion dispararLaser pone el laser justo encima de la posicion de la nave cuando pulsamos el espacio.

BONUS TRACK: ¿Podemos hacer que el laser solo puede volver a ser disparado cuando el ultimo rayo haya salido de la pantalla?

Solucion 4 - pestaña Laser.

```
float laserX = -10;
float laserY = -10;

void dibujarLaser() {
    fill(255,0,0);
    rect(laserX-2.5, laserY, 5, 20); // Resto 2.5 para centrarlo horizontalmente
}

void moverLaser() {
    laserY = laserY - 5;
}

void dispararLaser() {
    if(keyPressed && key==' ' && laserY < 0) { // si y<0 se ha ido de la pantalla
        laserX = playerX;
        laserY = 440-20; // Resto 20 para que aparezca justo encima de la nave
    }
}
```

Solucion 4 - pestaña Principal.

```
void setup() {  
    size(500,500);  
    crearNave();  
}  
  
void draw() {  
    background(0);  
    moverNave();  
    moverLaser();  
    dibujarNave();  
    dibujarLaser();  
    dispararLaser();  
}
```

Estructuras (I)

En Processing podemos agrupar varias variables en una estructura "clase" para organizar mejor nuestro código

Por ejemplo, podríamos tener una estrella definida por 2 enteros: las coordenadas x,y

```
class Star {  
    int x;  
    int y;  
};
```

Estructuras (II)

Para acceder a un campo de dentro de una estructura

```
Star cassiopeia = new Star();
```

```
cassiopeia.x = 1;
```

```
cassiopeia.y = 2;
```

```
println(cassiopeia.x);
```

```
println(cassiopeia.y);
```

Arrays de estructuras I

También podemos crear un array de estructuras

```
Star[] stars = new Star[100]; // Creamos sitio para 100 estrellas  
  
void creaEstrellas() {  
    for(int i=0;i<100;i++) {  
        star[i] = new Star(); // Creamos cada una de las 100 estrellas  
        star[i].x = 1;  
        star[i].y = 2; // Les asignamos valores a las coordenadas  
    }  
}
```

Arrays de estructuras II

Luego podemos usar el array, para mostrar sus valores:

```
void escribeEstrellas() {  
    for(int i=0;i<100;i++) {  
        println(stars[i].x);  
        println(stars[i].y);  
    }  
}
```

Arrays de estructuras III

O para dibujarlas:

```
void dibujaEstrellas() {  
    for(int i=0;i<100;i++) {  
        rect(stars[i].x, stars[i].y, 5, 5);  
    }  
}
```

Constantes (I)

```
static final int STARS = 100;
```

Esto nos define una constante, es decir una variables que no se puede cambiar durante el programa

Es útil para poder cambiar de golpe un numero en varios sitios de nuestro programa, p.e el tamaño de nuestro vector de estrellas

Constantes (II)

```
static final int STARS = 100;

void creaEstrellas() {
    Star[] stars = new Star[STARS];

    for(int i=0;i<STARS;i++) {
        stars[i] = new Star();
        stars[i].x = i;
        stars[i].y= i+1;
    }
}
```

Así podemos cambiar el numero de estrellas de forma fácil

5. Añadimos los enemigos

En una nueva pestaña Enemigos, vamos a crear una estructura Enemigo para guardar tres variables de tipo int: x,y y estado.

Crearemos un array de 10 enemigos.

La función crearEnemigos los creará y les pondrá sus coordenadas iniciales:

- Fila 70
- Y distribuiremos los 10 enemigos entre las 500 columnas que tenemos.

La función dibujarEnemigos los dibujara en base las coordenadas que hemos puesto en su estructura.

Solucion 5 - pestaña Enemigos.

```
class Enemigo {  
    int x;  
    int y;  
    int estado;  
}  
  
static final int ENEMIGOS = 10;  
Enemigo[] enemigos = new Enemigo[ENEMIGOS];  
  
void crearEnemigos() {  
    for(int i=0;i<ENEMIGOS;i++) {  
        enemigos[i] = new Enemigo();  
        enemigos[i].x = 50*i;  
        enemigos[i].y = 70;  
    }  
}  
  
void dibujarEnemigos() {  
    for(int i=0;i<ENEMIGOS;i++) {  
        rect(enemigos[i].x, enemigos[i].y, 50, 50);  
    }  
}
```

Solucion 5 - pestaña Principal.

```
void setup() {  
    size(500,500);  
    crearNave();  
    crearEnemigos();  
}  
  
void draw() {  
    background(0);  
    moverNave();  
    moverLaser();  
    dibujarNave();  
    dibujarEnemigos();  
    dibujarLaser();  
    dispararLaser();  
}  
}
```

6. Ponemos una imagen a los enemigos

En una nueva pestaña Enemigos, vamos a crear una estructura Enemigo para guardar tres variables de tipo int: x,y y estado.

Crearemos un array de 10 enemigos.

La funcion crearEnemigos los creara y les pondra sus coordenadas iniciales:

- Fila 70
- Y distribuiremos los 10 enemigos entre las 500 columnas que tenemos.

La funcion dibujarEnemigos los dibujara en base las coordenadas que hemos puesto en su estructura.

Solucion 6 - pestaña Enemigos.

```
class Enemigo {  
    int x;  
    int y;  
    int estado;  
}  
  
static final int ENEMIGOS = 10;  
Enemigo[] enemigos = new Enemigo[ENEMIGOS];  
PImage enemigo; // Nueva variable para almacenar la imagen de los enemigos  
  
void crearEnemigos() {  
    enemigo = loadImage("enemigo.png"); // Cargamos la imagen  
    for(int i=0;i<ENEMIGOS;i++) {  
        enemigos[i] = new Enemigo();  
        enemigos[i].x = 50*i;  
        enemigos[i].y = 70;  
    }  
}  
  
void dibujarEnemigos() {  
    for(int i=0;i<ENEMIGOS;i++) {  
        image(enemigo, enemigos[i].x, enemigos[i].y, 50, 50); // Mostramos la imagen del enemigo en vez del rectangulo  
    }  
}
```

7. Poder destruir a los enemigos

Vamos a usar la variable estado de cada enemigo como indicador de si esta vivo y por lo tanto lo dibujamos o muerto y por lo tanto no lo dibujamos.

Modificamos la funcion dibujarEnemigos para que solo dibuje un enemigo si su estado es igual a 0.

La funcion moverEnemigos marcará un enemigo como muerto (estado=1) si las coordenadas del laser estan dentro del cuadrado que ocupa un enemigo.

Solucion 7 - pestaña Principal.

```
void setup() {  
    size(500,500);  
    crearNave();  
    crearEnemigos();  
}  
  
void draw() {  
    background(0);  
    moverNave();  
    moverLaser();  
    moverEnemigos();  
    dibujarNave();  
    dibujarEnemigos();  
    dibujarLaser();  
    dispararLaser();  
}  
}
```

Solucion 7 - pestaña Enemigos.

```
void dibujarEnemigos() {  
    for(int i=0;i<ENEMIGOS;i++) {  
        if(enemigos[i].estado==0) { // Solo dibujamos si el estado es 0  
            image(enemigo, enemigos[i].x, enemigos[i].y, 50, 50);  
        }  
    }  
}  
  
void moverEnemigos() {  
    for(int i=0;i<ENEMIGOS;i++) {  
        if(laserX > enemigos[i].x && laserX < enemigos[i].x + 50 &&  
            laserY > enemigos[i].y && laserY < enemigos[i].y + 50) {  
            enemigos[i].estado = 1;  
        }  
    }  
}
```

Solucion 7 - pestaña Enemigos (otra opción).

```
void dibujarEnemigos() {
    for(int i=0;i<ENEMIGOS;i++) {
        if(enemigos[i].estado==0) { // Solo dibujamos si el estado es 0
            image(enemigo, enemigos[i].x, enemigos[i].y, 50, 50);
        }
    }
}

// Esta funcion nos dice si un punto esta dentro de un rectangulo o no
boolean boxCollision(float x, float y, float enemyX, float enemyY, float width, float height) {
    if(x>enemyX && x < enemyX+width && y>enemyY && y < enemyY+height) {
        return true;
    } else {
        return false;
    }
}

void moverEnemigos() {
    for(int i=0;i<ENEMIGOS;i++) {
        if(boxCollision(laserX, laserY, enemigos[i].x, enemigos[i].y, 50, 50)) { // El codigo queda así más sencillo
            enemigos[i].estado = 1;
        }
    }
}
```

8. Mover a los enemigos

En La funcion moverEnemigos podemos modificar la variable x de los enemigos y estos se moveran. Simplemente vamos a ir sumando 1 pixel a la x en cada frame.

Solucion 8 - pestaña Enemigos.

```
void moverEnemigos() {  
    for(int i=0;i<ENEMIGOS;i++) {  
        if(boxCollision(laserX, laserY, enemigos[i].x, enemigos[i].y, 50, 50)) {  
            enemigos[i].estado = 1;  
        }  
        enemigos[i].x = enemigos[i].x + 1;  
    }  
}
```

Controlar el tiempo (I)

Para saber cuánto tiempo ha pasado podemos utilizar un **contador**.

```
int frames = 0;  
  
void draw() {  
    frames++;  
  
    if(frames == 100) {  
        // 100 fotogramas han pasado :)  
    }  
}
```

Controlar el tiempo (II)

Esto solo cuenta el primer intervalo de tiempo.

```
int frames = 0;  
int waitUntil = 100;  
  
void draw() {  
    frames++;  
  
    if(frames == waitUntil) {  
        // 100 fotogramas han pasado :)  
        waitUntil += 100;  
    }  
}
```

9. Mover a los enemigos hacia los dos lados

Con el cambio anterior en moverEnemigos los enemigos se mueven, pero acaban despareciendo todos por la derecha!

Tenemos que hacer que vayan un cierto tiempo hacia un lado, y luego el mismo tiempo hacia el otro. Para que vayan en una u otra dirección añadiremos una variable velocidadEnemigos que utilizaremos en moverEnemigos para que no siempre sumemos 1 a la x sino esta variable, que podra valer 1 o -1.

9. Mover a los enemigos hacia los dos lados

Para ir contando cuanto tiempo ha pasado usaremos un contador de frames. Una variable tiempo se ira incrementado en 1 cada vez que hagamos moverEnemigos.

Cuando la variable llega a 50, cambiaremos el signo de velocidadEnemigos para que cambien de direccion y pondremos el contador tiempo a cero, para que vuelva a contar hacia el otro lado.

Solucion 9 - pestaña Enemigos.

```
int tiempo = 0;
int velocidadEnemigos = 1;

void moverEnemigos() {
    tiempo = tiempo+1;

    if(tiempo==50) {
        velocidadEnemigos = -velocidadEnemigos;
        tiempo = 0;
    }

    for(int i=0;i<ENEMIGOS;i++) {
        if(boxCollision(laserX, laserY, enemigos[i].x, enemigos[i].y, 50, 50)) {
            enemigos[i].estado = 1;
        }

        enemigos[i].x = enemigos[i].x + velocidadEnemigos;
    }
}
```

10. Bomba!

Ya tenemos unos enemigos a los que disparar. Ahora vamos a hacer que se defiendan. Cada 5 segundos los enemigos nos lanzaran una bomba, de forma similar a la que nosotros disparamos el laser.

Añadimos una pestana Bomba y dos variables bombaX y bombaY. Inicialmente la posicionamos fuera de la pantalla.

La función dibujarBomba dibuja un cuadrado de 20x20 centrado en la posición que definen estas variables.

La función moverBomba hace que se desplace hacia abajo de 5 en 5 pixeles.

10. Bomba!

Finalmente la función dispararBomba es la que decide cuando hay que disparar la bomba. Vamos a contar frames igual que hicimos para decidir si los enemigos iban a la izquierda o a la derecha.

La variable tiempoBomba que iniciamos a 0 nos indica cuantos frames han pasado. Cuando pasen 300 frames (5 segundos a 60 fps) se lanzara una nueva bomba. Para ello:

- ponemos la bomba en la misma columna que este el jugador.
- ponemos la bomba en lo alto de la pantalla
- volvemos a poner tiempoBomba a 0.

Solucion 10 - pestaña Enemigos.

```
float bombaX;
float bombaY = 510;

void dibujarBomba() {
    fill(255,255,0);
    rect(bombaX-10, bombaY, 20, 20);
}

void moverBomba() {
    bombaY = bombaY + 5;
}

int tiempoBomba = 0;

void dispararBomba() {
    tiempoBomba = tiempoBomba + 1;
    if(tiempoBomba > 5*frameRate) {
        bombaX = playerX;
        bombaY = 10;
        tiempoBomba = 0;
    }
}
```

Solucion 10 - pestaña Principal.

```
void setup() {  
    size(500,500);  
    crearNave();  
    crearEnemigos();  
}  
  
void draw() {  
    background(0);  
    moverNave();  
    moverLaser();  
    moverBomba();  
    dibujarNave();  
    dibujarEnemigos();  
    dibujarLaser();  
    dibujarBomba();  
    dispararLaser();  
    dispararBomba();  
}
```

Enumeraciones I

En ocasiones necesitamos guardar estados o modos, esto lo podemos codificar con números:

Estado	Número
DEAD	0
NORMAL	1
ATTACKING	2

Y luego lo guardamos en una variable numérica:

```
int estadoEnemigo = 1;
```

Enumeraciones I

Pero Processing nos permite hacerlo más elegante y sin tener que recordar que codificación hemos usado:

```
enum State { DEAD, NORMAL, ATTACKING };
```

Y definir variables de este tipo nuevo que hemos definido:

```
State estadoEnemigo = State.NORMAL;
```

OPCIONAL Utiliza enumeraciones

Utiliza enumeraciones en vez de un numero int para indicar si un enemigo esta vivo o no.

Recorridos

Un bucle típicamente sirve para hacer un recorrido, es decir, tratar todos los elementos:

```
for(int i=0; i<NUM_ELEMENTOS; i++) {  
    haz_algo(vec[i]);  
}
```

Búsquedas

Pero también podemos usarlo para buscar un elemento del vector que cumpla una condición.

```
for(int i=0;i<NUM_ELEMENTOS;i++) {  
    if(condicion_a_buscar(vec[i]) {  
        return i;  
    }  
}
```

En este caso, debemos salir del bucle (i.e return o break) una vez localizado un elemento que cumpla la condición.

Busca enemigo vivo

Hacemos una función `buscaEnemigoVivo` que nos retorne el primer enemigo vivo. Muestra por pantalla su numero.

BONUS TRACK: ¿Puedes devolver el n-essimo enemigo vivo?

OPCIONAL Enemigos que atacan

**Ahora que sabemos buscar un enemigo vivo,
¿puedes hacer que cada 10 segundos, el primer
enemigo vivo que quede te ataque en plan kamikaze
como el juego original?**

Scroll parallax

El ultimo toque para nuestro juego van a ser unas estrellas en el fondo que se vayan moviendo hacia abajo.

¿Cómo lo harías?

Para conseguir un efecto más realista, prueba a hacer que las estrellas pares vayan a una velocidad y las impares a otras.