Práctica 2:

Una aplicación en Processing

Objetivo de la práctica

Los objetivos de esta práctica son los siguientes:

- Familiarización con aspectos de programación a través de programas en Processing.

1. Programas dinámicos

Como ya avanzamos en la práctica anterior, en esta sesión continuaremos trabajando en Processing y usaremos como ejemplo recurrente el alienígena *Zoog*, mostrado en la Figura 1.

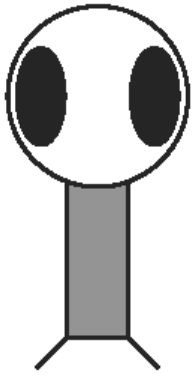

Fig. 1. Zoog

Para dibujar a Zoog, podemos utilizar el siguiente código:

```
size(800, 800);
ellipseMode(CENTER);
rectMode(CENTER);
stroke(0);
fill(150);
rect(100,100,20,100);
fill(255);
ellipse(100,70,60,60);
fill(0);
ellipse(81,70,16,32);
ellipse(119,70,16,32);
stroke(0);
line(90,150,80,160);
line(110,150,120,160);
```

Antes de continuar, cabe destacar que Processing destaca utilizando un color diferente aquellas palabras del lenguaje que conoce, como el método "`size`" o el valor "`CENTER`". Por otra parte, hemos visto que si cometemos algún error se indicará en la consola inferior.

Ahora vamos a modificar el código anterior introduciendo algunas mejoras. En primer, vamos a introducir *comentarios*, es decir, frases en lenguaje natural que ayudan a los humanos a entender el código pero no son consideras como instrucciones por el ordenador. Existen dos maneras de introducir comentarios:

- Usando "`//`", se define un comentario que dura hasta el fin de la línea actual.
- Usando "`/*`", se define un comentario que dura hasta la aparición de "`*/`", pudiendo abarcar varias líneas.

Añade comentarios al programa anterior explicando las diferentes partes del código.

Además, vamos a dividir el código en varios *métodos*. Más adelante veremos que, además de poder utilizar los métodos existentes en el lenguaje, podemos definir nuestros propios métodos. Por ahora vamos a utilizar dos métodos especiales que aparecen en Processing:

- El método `setup` permite indicar qué instrucciones queremos que se ejecuten una única vez, al comienzo de la ejecución de nuestro programa.
- El método `draw` permite indicar qué instrucciones queremos que se ejecuten repetidamente, hasta que finalicemos la ejecución de nuestro programa.

El nuevo código (crea un nuevo fichero para no perder el archivo anterior) quedaría así:

```
void draw() {
      // Pinta un fondo blanco
      background(255);

      // Fija las coordenadas basadas en el centro
      ellipseMode(CENTER);
      rectMode(CENTER);

      // Dibuja el cuerpo
      stroke(0);
      fill(150);
      rect(100, 100, 20, 100);

      // Dibuja la cabeza
      stroke(0);
      fill(255);
      ellipse(100, 70, 60, 60);

      // Dibuja los ojos
      fill(0);
      ellipse(81, 70, 16, 32);
      ellipse(119, 70, 16, 32);

      // Dibuja las piernas
      stroke(0);
      line(90, 150, 80, 160);
      line(110, 150, 120, 160);
}

void setup() {
      // Tamaño del lienzo
      size(800, 800);
}
```

Observa que el uso que hacemos de los métodos es diferente al visto hasta ahora. Antes usábamos una instrucción que Processing sabía cómo ejecutar, mientras que ahora estamos enseñando a Processing a realizar una instrucción. Por ahora, no nos preocuparemos demasiado de la palabra `void`, los paréntesis o las llaves. Baste decir que forman parte del mecanismo que ofrece Processing para usar sus métodos.

Ejecuta el código anterior. ¿Hay alguna diferencia con la primera versión de Zoog?

Aunque no lo parezca sí que la hay, ya que ahora se está ejecutando repetidamente el código incluido en el método `draw` y por tanto se volviendo a dibujar al marciano continuamente. Diremos que este nuevo código es *dinámico* y que el anterior es *estático*.

Aunque pueda no parecerlo, es una diferencia importante y en seguida comprobarás su utilidad. Por ejemplo, podemos decir que la posición de algún elemento gráfico son las coordenadas del cursor del ratón, de forma que al mover el ratón cambie la posición del elemento gráfico.

Las coordenadas del cursor del ratón se definen como `mouseX` y `mouseY`. Modifica el programa dinámico para que el cuerpo de Zoog tenga como su esquina superior izquierda en la posición del cursor del ratón.

Puedes comprobar fácilmente que utilizar `mouseX` y `mouseY` en el código estático no produce el efecto deseado, ya que no se actualizará la posición del elemento gráfico al mover el ratón.

El código dinámico puede continuar mejorándose. Haz que el círculo que representa la cabeza esté centrado en el punto (`mouseX`, `mouseY – 30`). Ejecuta el programa.

¿Serías capaz de modificar también los ojos y las piernas para que todo el cuerpo reaccione a los movimientos del ratón?

2. Mejorando la interacción

En esta sección veremos como interactuar con los programas más allá de las posibilidades estudiadas hasta ahora.

Otra posibilidad interesante es el uso de `pmouseX` y `pmouseY`, que almacenan la posición anterior a la actual del cursor del ratón. Esto permite hacer un programa sencillo que muestre el rastro del ratón.

```
void setup() {
      size(800, 800);
      background(255);
      smooth();
}

void draw() {
      stroke(0);
      line(pmouseX, pmouseY, mouseX, mouseY);
}
```

El método `smooth` simplemente aplica una técnica conocida como *antialiasing* que mejora la representación gráfica de nuestro programa. Puedes probar el efecto que produce eliminarla sustituyéndola por `noSmooth`.

¿Qué pasaría si la instrucción `background(255)` estuviera dentro del método `draw`?

A continuación explicaremos dos métodos especiales de Processing y que permiten interaccionar con el programa por medio del ratón y el teclado.

- El método `mousePressed` permite indicar qué instrucciones queremos que se ejecuten cuando se pulsa un botón del ratón.

- El método `keyPressed` es igual pero cualquier tecla del teclado.

Esto nos permite modificar el código dinámico de Zoog para que haga lo siguiente:

- El cuerpo completo de Zoog se situará en la posición del cursor del ratón
- El color de los ojos de Zoog irá cambiando según lo haga el cursor del ratón.
- Al hacer click con el ratón, se mostrará un mensaje en la consola de mensajes.

El código sería el siguiente:

```
void setup() {
      // Tamaño del lienzo
      size(800, 800);
      smooth();
      frameRate(30);
}

void draw() {
      // Pinta un fondo blanco
      background(255);

      // Fija las coordenadas basadas en el centro
      ellipseMode(CENTER);
      rectMode(CENTER);

      // Dibuja el cuerpo
      stroke(0);
      fill(175);
      rect(mouseX, mouseY, 20, 100);

      // Dibuja la cabeza
      stroke(0);
      fill(255);
      ellipse(mouseX, mouseY - 30, 60, 60);

      // Dibuja los ojos
      fill(mouseX, 0, mouseY);
      ellipse(mouseX - 19, mouseY - 30, 16, 32);
      ellipse(mouseX + 19, mouseY - 30, 16, 32);

      // Dibuja las piernas
      stroke(0);
      line(mouseX - 10, mouseY + 50, pmouseX - 10, pmouseY + 60);
      line(mouseX + 10, mouseY + 50, pmouseX + 10, pmouseY + 60);
}

void mousePressed() {
      println( "Se ha pulsado el botón del ratón");
}
```

El método `frameRate` sirve para especificar la tasa de actualización del lienzo, en frames por segundo. El valor por defecto es de 60 frames por segundo.

3. Variables, condicionales y bucles

Hemos visto que el método `draw` se ejecuta repetidamente. Es posible que nos interese que el valor de un cierto dato dependa del valor que haya tomado en la anterior ejecución del método. Para hacer este tipo de cosas existe el concepto de *variable*. Una variable no es más que un dato de cierto tipo (por ejemplo, entero, carácter o real) cuyo valor puede cambiar a lo largo de la ejecución de un programa.

Por ejemplo, vamos a dibujar un círculo y vamos a hacer que el círculo cambie su posición en cada repetición de `draw`. Para ello, usaremos una variable `circleX` que tomará el valor inicial 1 y que irá aumentando progresivamente su valor, permitiendo desplazar el círculo a lo largo del eje de abscisas.

```
int circleX = 100;
int circleY = 100;

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

void draw() {
      background(255);
      stroke(0);
      fill(175);
      ellipse(circleX, circleY, 50, 50);
      circleX = circleX + 1;
}
```

Otro concepto importante en informática es el de *instrucción condicional*. Las instrucciones condicionales permitan que un código se ejecute solamente si se cumple una determinada condición. La sintaxis es la siguiente:

```
if (...) // condición1
{
      // Instrucciones si se cumple la condición1
      // ..
}
```

Una posible situación práctica donde esto puede ser de utilidad es el siguiente. Podemos hacer que el fondo no tenga un color constante, sino que dependa de la posición del ratón. Esto se podría hacer del siguiente modo:

- Si la posición X del cursor del ratón está situada en el primer tercio del lienzo, entonces el fondo será de color blanco.
- Si la posición X del cursor del ratón está situada en el segundo tercio del lienzo, entonces el fondo será de color gris.
- Si la posición X del cursor del ratón está situada en el tercer tercio del lienzo, entonces el fondo será de color negro.

Para poder realizar esto, Processing proporciona la instrucción `if-else`, cuya sintaxis es:

```
if (...) // condición1
{
      // Instrucciones si se cumple la condición1
      // ...
} else if (...) // condición2
{
      // Instrucciones si no se cumple condición1 pero sí  condición2
      // ...

} else {
      // Instrucciones si no se cumple condición1 ni condición2
      // ...
}
```

Modifica el código del programa para obtener el funcionamiento anteriormente descrito.

Obviamente, la instrucción `if-else` anterior puede utilizarse en más casos. Puede haber únicamente un `if` sin ningún `else`, puede haber más casos de la forma `else if` , etc.

Nuestro próximo objetivo va a ser dibujar varios alienígenas. Para ello, escribe el código adecuado para mostrar 2 Zoogs en la pantalla. Puedes copiar y pegar el código actual y modificar adecuadamente el código pegado.

Obviamente, este procedimiento no es el más adecuado. Para facilitar la ejecución repetida de alguna acción, los lenguajes de programación ofrecen los *bucles*, que son instrucciones que permitir repetir la ejecución de una o más instrucciones. Por ejemplo, la siguiente instrucción `for` permite repetir *n* veces las instrucciones que se añadan entre las llaves.

```
for (int i = 1; i <= n; i++)
{
      // Instrucciones a repetir
      // ...
}
```

Utiliza la instrucción `for` para dibujar 4 Zoogs en fila, de forma que cada Zoog estará representado una distancia horizontal de 100 píxeles del anterior.

Una vez conseguido lo anterior, modifica el código de creación de Zoog para que cada alienígena tenga z pares de brazos. Comprueba que funciona para, por ejemplo, `z = 3`.