TAREA 3

Fuegos artificiales

Programación con p5.js

¿Qué vamos a hacer?

En esta tarea vas a crear una simulación de **fuegos artificiales interactivos**. Cada vez que hagas clic en el lienzo, un cohete saldrá disparado desde ese punto, subirá frenado por la gravedad y explotará en una lluvia de chispas de colores que se desvanecen poco a poco.

El programa introduce el concepto central de esta tarea: los **objetos**. Verás por qué agrupar propiedades relacionadas en un objeto hace el código más claro y más fácil de ampliar, y lo compararás directamente con el estilo de arrays paralelos que ya conoces de la Tarea 2.

Antes de empezar

Este tutorial presupone que conoces setup(), draw(), background(), circle(), stroke(), fill(), random() y el bucle for. También necesitarás el concepto de array de la Tarea 2. Si alguno de estos puntos te resulta poco familiar, repásalo antes de continuar.

Paso 1 — Un cohete que sube

Empezamos con lo más sencillo: **un único cohete** que sube desde el centro del lienzo. Tiene posición (x, y) y velocidad (vx, vy). La gravedad se simula sumando un pequeño valor positivo a vy en cada fotograma: el cohete va frenando, se detiene y luego cae.

```
let x, y;
let vx, vy;

function setup(){
  createCanvas(windowWidth, windowHeight)
  x = width / 2;
  y = height;
  vx = random (-2,2)
  vy = random (-20,-5) //siempre hacia arriba

}

function draw(){
  background(0)
  x = x + vx;
  y = y + vy;
  vy = vy + 0.1 // gravedad: vy aumenta cada fotograma
  noStroke();
  fill(255,80,0); //naranja
  circle(x,y,8);
}
```

Cambia el valor de vy inicial (ahora random(-4,-2) a random(-8,-5). ¿A qué altura llega el cohete? Prueba también a cambiar la gravedad (0.1) a 0.05 a 0.2 y observa cómo varía la trayectoria.

Paso 2 — El cohete como objeto

Concepto: objeto literal

En la Tarea 2 usamos **arrays paralelos** para guardar los datos de cada partícula: un array para x, otro para y, otro para vx, otro para vy. El índice i relacionaba los datos que pertenecían a la misma partícula.

Un **objeto literal** agrupa todas las propiedades relacionadas bajo un mismo nombre. Se escribe entre llaves { }, separando cada propiedad con coma. Compara los dos estilos:

```
// Con variables sueltas (lo que conocemos de tareas anteriores)
let x, y, vx, vy;

// Con un objeto literal: todas las propiedades agrupadas
let cohete = {
  x:  width / 2,  // posición x inicial
  y:  height,  // posición y inicial (base del lienzo)
  vx: random(-2, 2),
  vy: random(-5, -2)
```

Para acceder a una propiedad se usa la **notación de punto**: `cohete.x`, `cohete.vy`, etc. Aquí está el programa completo reescrito con el objeto:

```
let cohete ={}

function setup(){
  createCanvas(windowWidth, windowHeight)
  cohete.x = width/2,
  cohete.y = height,
  cohete.vx = random(-2,2),
  cohete.vy = random(-10,-5)
}

function draw(){
  background(0)
  cohete.x = cohete.x + cohete.vx;
  cohete.y = cohete.y + cohete.vy;
  cohete.vy = cohete.vy + 0.1 // gravedad: vy aumenta cada fotograma
  noStroke();
  fill(255,80,0); //naranja
  circle(cohete.x,cohete.y,8);
}
```

¿Qué ha cambiado?
El resultado en pantalla es idéntico al Paso 1. Lo que ha cambiado es la organización del código: ahora x, y, vx y vy están agrupadas en el objeto cohete. Esto no parece una gran ventaja con un único cohete, pero en el Paso 4, cuando tengamos varios, la diferencia será evidente.

Paso 3 — La explosión

Cuando el cohete empieza a caer (vy > 0), queremos que explote. Hay dos versiones: un destello rápido de implementar y una lluvia de chispas más realista.

Versión A: destello simple

Añadimos al objeto dos propiedades nuevas: `explotado` (booleano que indica si ya ha llegado al punto más alto) y `radioDest` (radio de un círculo que crece cada fotograma).

```
function setup(){
  createCanvas(windowWidth, windowHeight)
  cohete.x = width/2,
  cohete.y = height,
  cohete.vx = random(-2,2),
  cohete.vy = random(-10,-5)
  //añadimos dos propiedades nuevas al objeto cohete:
  cohete.explotado = false
  cohete.radioDest = 0
}
```

Atención

Recuerda que draw() está en bucle muchas veces por segundo, por esto hemos creado la variable booleana cohete.explotado. Para revisar en cada vuelta de draw() si el cohete ha explotado ya o no.

En draw(), después de hacer subir al cohete, decidimos que en cuanto llegue a su punto máximo de altura, en vez de empezar a caer, explotará:

```
  if (cohete.vy > 0 && !cohete.explotado){
    cohete.explotado = true; //empieza a caer: explotar
  }
  if (!cohete.explotado){
    fill(255,80,0); //naranja
    circle(cohete.x,cohete.y,8);
  } else {
    cohete.radioDest += 2; //destello que crece
    noFill()
    stroke(255,200,0);
    circle(cohete.x, cohete.y, cohete.radioDest * 2)
  }
```

Experimenta

Cambia el incremento de radioDest (ahora 4) a 8 o a 2. ¿Cómo afecta a la velocidad de expansión? Añade una condición para que el destello desaparezca cuando radioDest supere 80: if (cohete.radioDest > 80) { cohete.radioDest = 0; cohete.explotado = false; }.

Versión B: lluvia de chispas

Para un efecto más realista, el objeto cohete puede contener un **array de chispas**. Cada chispa es a su vez un objeto con posición, velocidad y vida. Aquí es donde los objetos muestran toda su potencia: un cohete contiene su propia explosión.

```
let cohete = { } //se llena en setup


function setup(){
  createCanvas(windowWidth, windowHeight)
  cohete.x = width/2,
  cohete.y = height,
  cohete.vx = random(-2,2),
  cohete.vy = random(-10,-5)
  cohete.explotado = false
  cohete.radioDest = 0
  //nueva variable de tipo array que contendrá las chispas.
  cohete.chispas = []
}
```

Creamos una nueva función llamada explotar, que necesitará como entrada un objeto, en este caso un cohete. Dentro de la función trabajará con algunas variables del cohete dado en la entrada, por ejemplo: cambiando su booleana explotado a true y rellenando el array de chispas con push().

```
function explotar(c){
  c.explotado = true;
  for (let i=0; i<60; i++){
    c.chispas.push({
      x:c.x, y:c.y,
      vx: random(-4,4),
      vy: random(-4,4),
      vida:255
    });
  }
}
```

Atención:

Las chispas son un nuevo objeto literal que crea y da valor a las variables que el programa considera oportuno en este momento, creando una nueva variable como vida.

Completamos draw con estas estructuras de control que deciden qué hacer con las partículas según estén o no subiendo, o según si ya han explotado o no.

```
if (cohete.vy > 0 && !cohete.explotado){
  cohete.explotado = true; //empieza a caer: explotar
  explotar(cohete);
}
if(!cohete.explotado){
  fill(255,80,0); //naranja
  circle(cohete.x,cohete.y,8);
} else {
  for(let i = cohete.chispas.length -1; i>=0; i--){
    let ch= cohete.chispas[i];
    ch.x += ch.vx;
    ch.y += ch.vy;
    ch.vy += 0.1; //gravedad en chispas
    ch.vida -=4;
    stroke(255,150,0,ch.vida);
    point(ch.x,ch.y);
    if(ch.vida<=0){cohete.chispas.splice(i,1);}
  }
}
```

Fíjate en splice:
chispas.splice(i, 1) elimina la chispa en la posición i del array. El bucle recorre el array al revés — de length-1 hasta 0 — para que al eliminar un elemento no se salten los siguientes. Usaremos el mismo truco en el Paso 4 para eliminar cohetes agotados.

Paso 4 — Varios cohetes: array de objetos

Ahora queremos lanzar un cohete nuevo en cada clic del ratón. La solución es un **array de objetos cohete**: `let cohetes = [];`. Cada elemento del array es un objeto completo con todas sus propiedades y sus propias chispas. La función `mousePressed()` añade un cohete nuevo; `draw()` los gestiona todos con un bucle for.

```
let cohete = [] //se llena en setup

function setup(){
  createCanvas(windowWidth, windowHeight)
  colorMode(HSB,360,100,100,255)
}

function mousePressed(){
  cohete.push({
    x:mouseX, y:mouseY, //sale desde donde haces clic
    vx:random(-2,2),
    vy:random(-7,-4),
    tono: random(360),
    explotado: false,
    chispas: []
  });
}

function draw(){
  background(0,0,0,40)//dejamos estela
  for(let i=cohete.length-1;i>=0;i--){
    let c = cohete[i];
    if(!c.explotado){
      c.x += c.vx; c.y +=c.vy; c.vy+=0.1;
      noStroke(); fill (c.tono, 80,100);
      circle (c.x, c.y, 7);
      if (c.vy > 0) explotar(c);
    }else{
      gestionarChispas(c);
      if(c.chispas.length === 0) cohete.splice(i, 1);
    }
  }
}
```

Atención:

Compara este código con el de pasos anteriores y descubre así diferentes sintaxis para programar lo mismo. Por ejemplo “+=” o expresar todo un if en una misma línea. ¿Qué significa ====?

Las funciones auxiliares `explotar()` y `gestionarChispas()` completan el programa:

```
function explotar(c){
  c.explotado = true;
  for (let k=0; k<60; k++){
    c.chispas.push({
      x:c.x, y:c.y,
      vx: random(-4,4),
      vy: random(-4,4),
      vida:255
    });
  }
}

function gestionarChispas(c){
  for (let i = c.chispas.length -1; i>=0; i--){
    let ch = c.chispas[i];
    ch.x += ch.vx; ch.y += ch.vy; ch.vy += 0.08;
    ch.vida -= 4;
    stroke(c.tono, 90,100, ch.vida);
    strokeWeight(2); point (ch.x, ch.y);
    if (ch.vida <=0) c.chispas.splice(i,1);
  }
}
```

Array de objetos frente a arrays paralelos

Compara este código con el de la Tarea 2. Allí teníamos px[], py[], pvx[], pvy[] y el índice i relacionaba los datos de la misma partícula. Aquí cada cohete lleva consigo todos sus datos agrupados: cohetes[i].x, cohetes[i].vy, cohetes[i].chispas... Al añadir una nueva propiedad — por ejemplo un sonido o un tipo de explosión — solo hay que añadirla al objeto, sin crear un array paralelo nuevo.

Paso 5 — Hazlo tuyo

El programa está completo. Ahora es tu turno de tomar decisiones para diferenciarlo. Ideas ordenadas de menor a mayor dificultad:

Color

- El código del Paso 4 ya usa colorMode(HSB). Experimenta con los valores de saturación y brillo de cadachispa para conseguir efectos distintos: chispas pálidas (baja saturación) o muy brillantes (brillo 100).
- Haz que las chispas cambien de tono según su vida: usa `map(ch.vida, 0, 255, 0, 60)` para ir del rojo al amarillo conforme se apagan.

Comportamiento

- Cambia el fondo de background(0,0,0,40) a background(240,30,10,40) para un cielo oscuro con toqueazulado.

- Haz que la potencia del cohete dependa de la posición vertical del clic: `vy = map(mouseY, 0, height, -10, -3)`. Si haces clic abajo sube más; si haces clic arriba, menos.

Ampliación

- Añade una propiedad `tipo` a cada cohete ("estrella", "cascada") y cambia la distribución de velocidades de las chispas según el tipo: las de "cascada" tienen vy siempre positiva.
- Lanza un cohete automáticamente cada 90 fotogramas usando `if (frameCount % 90 === 0)` dentro de draw(), además de los cohetes manuales.

Entrega

1. Guarda el proyecto en el editor (necesitas cuenta para guardarlo online).
2. Copia el enlace del proyecto (botón **Share** en el editor).
3. Entrega el enlace respondiendo a las preguntas de la ficha siguiente.

Ficha de entrega

Pregunta	Tu respuesta
¿Qué diferencia notaste al usar un objeto en lugar de variables sueltas?	
¿Qué variación añadiste en "Hazlo tuyo"?	
¿Qué fue lo más difícil de esta tarea?	
¿Que añadirías si tuvieras más tiempo?	

Resumen de lo aprendido en esta tarea

Elemento	Para qué sirve en esta tarea
{ }	Agrupa propiedades relacionadas en un objeto literal.
objeto.propiedad	Accede o modifica una propiedad con notación de punto.

array de objetos	Lista donde cada elemento es un objeto con sus propias propiedades agrupadas.
splice(i, 1)	Elimina un elemento de un array por su índice (para borrar chispas o cohetes agotados).
vida / alpha	Hace las chispas más transparentes conforme pierden vida, creando desvanecimiento.
mousePressed()	Función de p5.js que se ejecuta cada vez que se hace clic; aquí lanza un cohete nuevo.