Universidad de Murcia
Grado en Ingeniería Informática

Sistemas Multimedia
Creación de un escenario en Processing
Curso 2022 – 2023
(Convocatoria de junio)

Memoria

Pablo Jesús Meca Martínez
pablojesus.mecam@um.es
Grupo 1, subgrupo 1.1

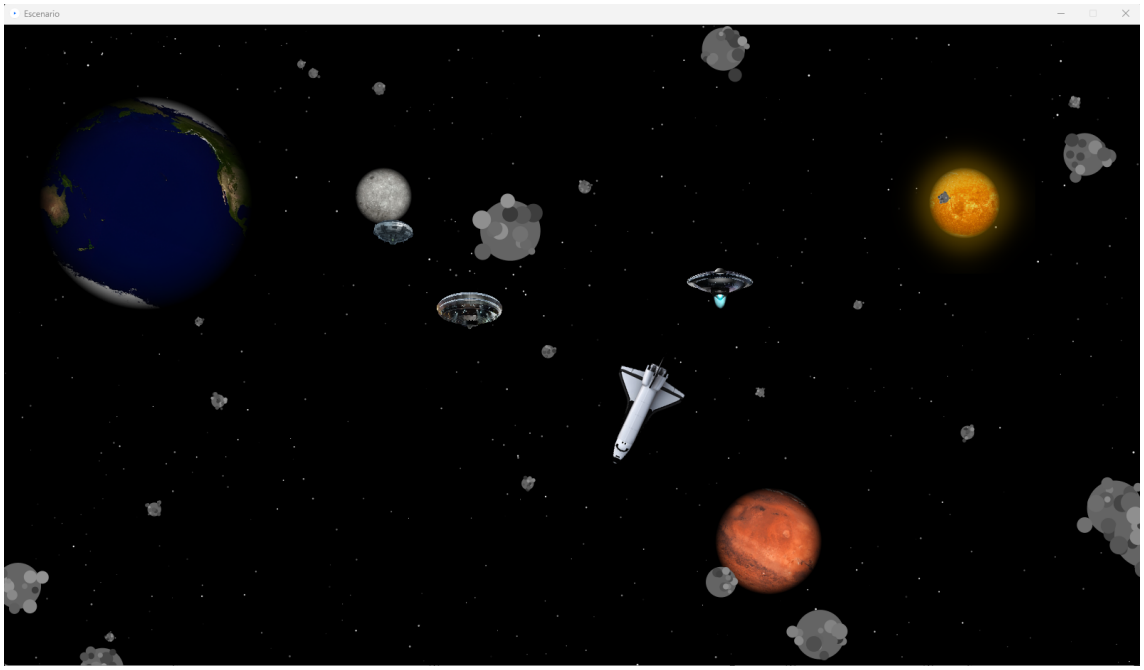
Profesora
María José Majado Rosales

Abril de 2023

Índice

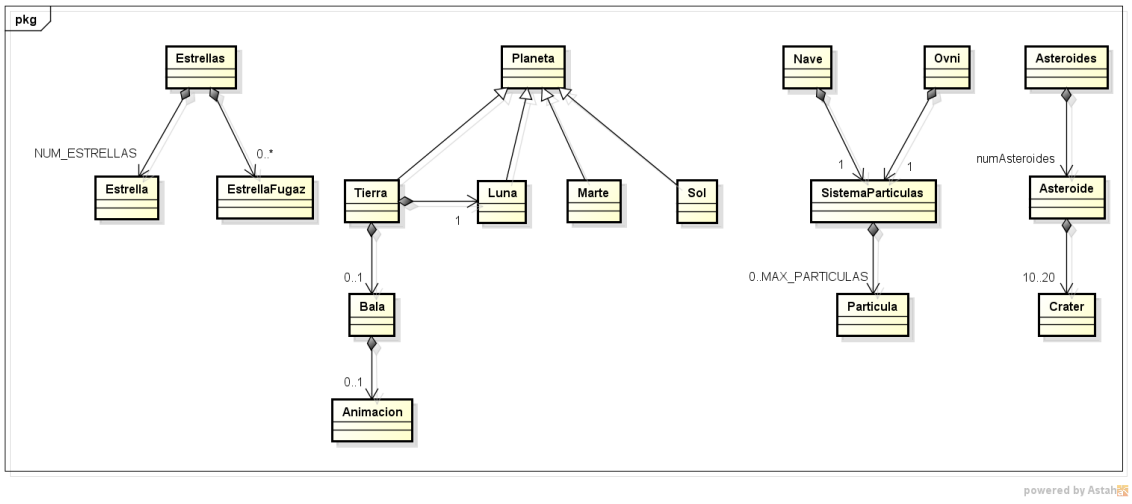
1. Introducción

En el presente documento se expondrá el diseño de un escenario interactivo desarrollado en Processing, la temática escogida ha sido el espacio. El escenario se compone de diversos elementos, tales como: estrellas, planetas, naves y asteroides. También incluye un sistema de partículas, una animación y una variedad de movimientos: rectilíneos, acelerados, parabólicos, giros, sinusoidales y una función de ruido Perlin. En su mayoría, el escenario utiliza *sprites* para darle un aspecto más realista, aunque también se han dibujado elementos con Processing, como los asteroides, la bala y las estrellas.



2. Estructura del escenario

El escenario se organiza en cuatro grupos principales de clases: las estrellas, los planetas, las naves y los asteroides. A su vez, el planeta Tierra contiene al grupo de clases relacionadas con la animación de la bala y las naves contienen al grupo del sistema de partículas. A continuación, se presenta un diagrama de las clases que componen el programa:



1. Estrellas

Este primer grupo se refiere a las estrellas que conforman el fondo del escenario. Cuando se crea el escenario, se generan 500 estrellas (constante `NUM_ESTRELLAS`) en posiciones aleatorias del fondo. Cada estrella tiene, además de una posición y un tamaño, un tiempo de vida, que va disminuyendo en 1 con cada fotograma; la vida de una estrella está directamente relacionada con su trasparencia, por lo que las estrellas van *apagándose* conforme se mueren, esta lógica la maneja la variable `vidaPorcentaje`. Cuando muere una estrella, automáticamente se recoloca en otro punto de la pantalla y se reinicializa.

Además de las estrellas del fondo, existe otro tipo de estrellas interactivas: las estrellas fugaces. Cuando el usuario pulsa la barra espaciadora del teclado, aparecerá una estrella fugaz en una posición aleatoria de la pantalla e irá bajando hacia la esquina inferior izquierda. Las estrellas fugaces se caracterizan por su *estela*, que consiste en una lista de estrellas que va dejando conforme avanza por la pantalla, las cuales nacen con el porcentaje de vida al máximo y comienzan a morir al instante[1] (es decir, las estrellas no nacen con un difuminado, aparecen directamente con el porcentaje de vida al 100%). Para aportar cierta variedad, las estrellas fugaces tienen una duración aleatoria de entre 80 y 200 fotogramas.

2. Planetas

Los planetas probablemente sean la parte más interesante de esta práctica debido a su efecto tridimensional. Todos los planetas se caracterizan por tener una textura[2] y una posición, además de diámetro, velocidad y ángulo respecto a su eje.

Para efectuar la rotación del planeta, se desplazan los píxeles del rectángulo de la textura hacia la derecha recorriendo inversamente la matriz por filas. Dependiendo del

[1] Las estrellas que conforman la estela son eliminadas de la lista conforme van muriendo para evitar que se recoloquen al morir, no volviendo a ser dibujadas y siendo destruidas por el recolector de basura de Java.

[2] Las texturas provienen de la web Solar System Scope.

valor de la constante `DEFAULT_SPEEDCOUNTER`, esta rotación puede efectuarse o no en todos los fotogramas de la animación (si las texturas tienen resolución baja, el planeta se moverá más rápido, por lo que puede ser interesante reducir las rotaciones a la mitad).

Posteriormente, aplicamos a la matriz una rotación en base a su ángulo respecto al eje y la trasladamos a la posición del planeta. En estos momentos aún contamos con un rectángulo que debemos convertir a una elipse. Primero, colocamos una elipse negra detrás del planeta para ocultar las estrellas que haya detrás y creamos un objeto `PGraphics` con la textura, de esta manera, habremos convertido el rectángulo en un cuadrado de alto y ancho: `diametro`. Seguidamente, creamos una elipse que haga de máscara, pero, en lugar de crear la elipse con `ellipse()`, la crearemos manualmente recorriendo la matriz de la máscara y asignando a cada posición una opacidad inversamente proporcional a la distancia respecto al centro de la elipse; de esta manera, logramos sombrear los bordes del planeta[3]. Ahora ya tenemos el planeta dibujado en forma de elipse.

Sin embargo, el planeta aún no se sentía verdaderamente tridimensional, ya que no era más que una elipse con bordes sombreados. Para conseguir el verdadero efecto 3D, realizaremos una distorsión de ojo de pez antes de aplicar la máscara, esto se consigue en la función `distorsionar()`. Esta función recorre el cuadrado de la imagen y analiza si cada píxel se encuentra o no en el radio de distorsión[4], en cuyo caso, obtiene el píxel desplazado en base a la cantidad de distorsión[5] mediante un mapeo, de manera que los píxeles más cercanos al centro se ven más desplazados que los píxeles más alejados.

Esta parte es la más costosa computacionalmente y supone una pérdida de 5 veces el rendimiento del escenario[6], no obstante, la simulación lograda es bastante satisfactoria.

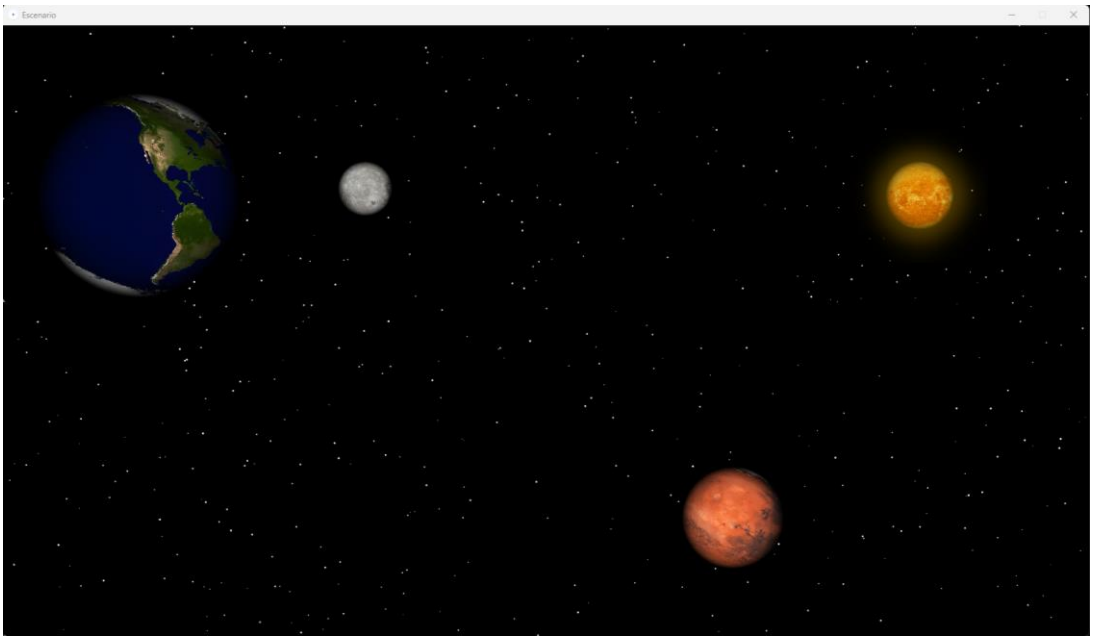
El escenario cuenta con cuatro tipos de *planetas*, de menor a mayor complejidad: Marte, la Luna, el Sol, y la Tierra, los últimos dos son interactivos al hacer clic sobre ellos.

El Sol cuenta con una elipse desenfocada que se crea en el constructor usando el filtro `BLUR` (para ahorrar recursos, solo se desenfoca una vez y se mantiene el efecto como una imagen). Cuando se clica el Sol, se efectúa una animación que aumenta su brillo durante unos instantes, el funcionamiento es el siguiente: el brillo del sol cuenta con una escala y un objetivo de escalado, la escala siempre busca igualarse al objetivo de escalado, por tanto, al hacer clic aumentamos el objetivo de escalado a 1.5 y, cuando la escala lo alcance, lo reducimos de nuevo a 1. De esta manera, el brillo aumentará durante unos instantes a 1.5 su escala, para posteriormente volver a reducirse a su tamaño original.

---

[3] La razón por la que anteriormente dibujábamos un fondo negro es precisamente para que, al sombrear los bordes reduciendo la opacidad, no se viesen estrellas detrás del planeta.
[4] El radio de distorsión es ligeramente superior (11/20) al radio real de la máscara para que el borde del efecto quede totalmente oculto fuera de los límites de esta.
[5] Tras una medición empírica, se ha determinado que 0.65 es un buen valor.
[6] Medición realizada teniendo en cuenta los 4 planetas del escenario, sin ellos se obtiene una media de 120 fps, mientras que con ellos es de 24 fps.

La Luna está agrupada dentro de la Tierra y se caracteriza por tener dos texturas que se intercambian después de producirse una colisión al final de la animación ejecutada por la Tierra.



3. Animación de la bala

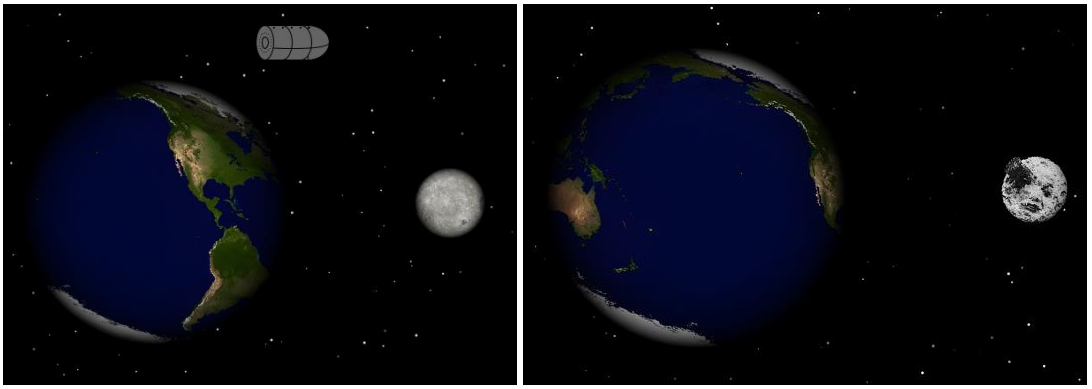
Esta animación consiste en el lanzamiento de una bala desde el planeta Tierra hasta la Luna. Cuando se hace clic sobre la Tierra, si aún no se ha ejecutado la animación (la Luna tiene su textura inicial), se crea una nueva bala y se le añade una nueva animación hacia la posición de la luna.

La bala está dibujada usando Processing. Se compone de un rectángulo central, un arco a la derecha simulando la punta, dos arcos en el rectángulo para dar la sensación de que se trata de un cilindro y dos elipses en el lado izquierdo simulando el final de la nave. Además, contiene una serie de puntos para simular puntos de soldadura sobre el metal. La bala solo se dibujará si cuenta con una animación que le indique la posición donde debe colocarse.

La animación consiste en el recorrido de una parábola desde un punto inicial hasta un punto final a una determinada velocidad. El punto de la animación en el que nos encontramos viene dado por el valor de la variable `t`, que va desde 0 (punto inicial) hasta 1 (punto final). Para obtener el punto actual en la parábola, se calcula el centro y el radio entre el final y el inicio. A continuación, se obtiene el ángulo de giro del sistema respecto al origen y se le suma los radianes que ya se ha desplazado la animación[7]. El valor del punto en el que debe encontrarse la bala se obtiene mediante el coseno (coordenada `x`) y el seno (coordenada `y`) de dicho ángulo. La bala también cambia de tamaño para simular que está alejándose/acercándose de los planetas, su tamaño será del 100% en el punto medio de la animación (`t`=0.5) y de 0% en los extremos (`t`=0 y `t`=1).

[7] Cuando la animación se completa, t valdrá 1 (o cercano a 1) que, al multiplicarlo por PI, nos dará un giro de 180º completo.

El planeta Tierra comprueba en todo momento si la animación ha terminado y, en ese momento, avisa a la Luna de que ha sucedido una colisión, cambiando esta su textura. El cambio de textura de la Luna provoca también que deje de rotar y de aplicarse la distorsión de ojo de pez. La nueva textura de la Luna se corresponde con una referencia al clásico cinematográfico de 1902: "Viaje a la Luna".



4. Nave interactiva y Ovnis

El escenario cuenta con una nave en primer plano que gira de manera indefinida en una dirección, determinada por el valor de la variable `incrementoAngulo`. Cuando se clica sobre ella, se propulsará hacia la dirección (ángulo) que esté mirando en ese momento. La nave realizará entonces un movimiento rectilíneo uniformemente acelerado, en el cual su velocidad se verá reducida por una determinada fricción (por defecto es 0.75). Una vez la nave vuelve a pararse, se determinará la nueva dirección de giro y continuará girando indefinidamente hasta que vuelva a ser pulsada.

En caso de que la nave salga de los límites de la pantalla, debe aplicarse una lógica adicional que se realiza en el método `checkSalirPantalla()`. Este método comprueba si se sale por alguno de los lados, en cuyo caso, la sitúa en el lado opuesto y le aplica un aumento de 0.5 a su velocidad. Puesto que en algunos casos puede suceder que la nave quede lo suficientemente fuera como para no ser realmente visible, pero lo suficientemente dentro como para que no se detecte que haya salido, también se analiza la posición del centro del *sprite* y, si está fuera, se vuelve a aumentar la velocidad y se aplica una ligera desviación en su ángulo. La razón de esta desviación es que puede suceder que la nave quede atrapada en un lateral si su eje queda prácticamente paralelo al mismo, sumando de forma infinita un aumento de la velocidad que provoca que se pierda fuera de la pantalla; para evitar esto, cada vez que el centro queda fuera, se aplica una pequeña desviación en su ángulo.

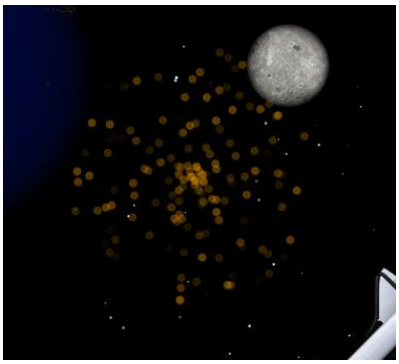
El escenario también cuenta con Ovnis, unas naves espaciales que se mueven por la pantalla. Cuando se crea un Ovni se determina, además de posición y velocidad, la imagen que tendrá de forma aleatoria a partir de las disponibles en el array `imagenes`. Los ovnis se mueven de forma independiente siguiendo una función de ruido Perlin, esto permite generar un movimiento mucho más orgánico que si se obtuviesen las posiciones de forma aleatoria. El usuario puede hacer clic sobre los ovnis para matarlos, en ese momento desaparecerá el *sprite* y se reemplazará por una serie de partículas que se irán disipando hasta que ovni desaparezca por completo.

5. Sistema de partículas

El escenario cuenta con un sistema de partículas controladas desde la clase `SistemaParticulas`, que contiene todas las partículas emitidas, además de la fuerza con la que debe emitirlas y el radio. Para crear una nueva partícula debe llamarse al método `addParticula()`, indicando, además de la posición, el ángulo con el que debe ser lanzada. El sistema cuenta con un límite preestablecido de 500 partículas para evitar que el rendimiento de la aplicación se vea afectado por un exceso de estas. Cuando una partícula muere, el sistema la borra de la lista de partículas después de mostrarla, esto lo hace recorriendo la lista en orden inverso.

Una partícula no es más que una elipse de color `RGB(255, 165, 0)` y una transparencia equivalente al valor de su vida, de forma similar a como ocurría con las estrellas. Las partículas se emiten en la dirección del ángulo a una fuerza aleatoria máxima de `fuerzaEmision` (por lo tanto, no todas las partículas se emiten a la misma velocidad, aunque tengan la misma fuerza de emisión) con una ligera dispersión determinada a partir de un valor aleatorio siguiendo una distribución Gaussiana, de manera que sea más probable que una partícula se dirija en el ángulo establecido.

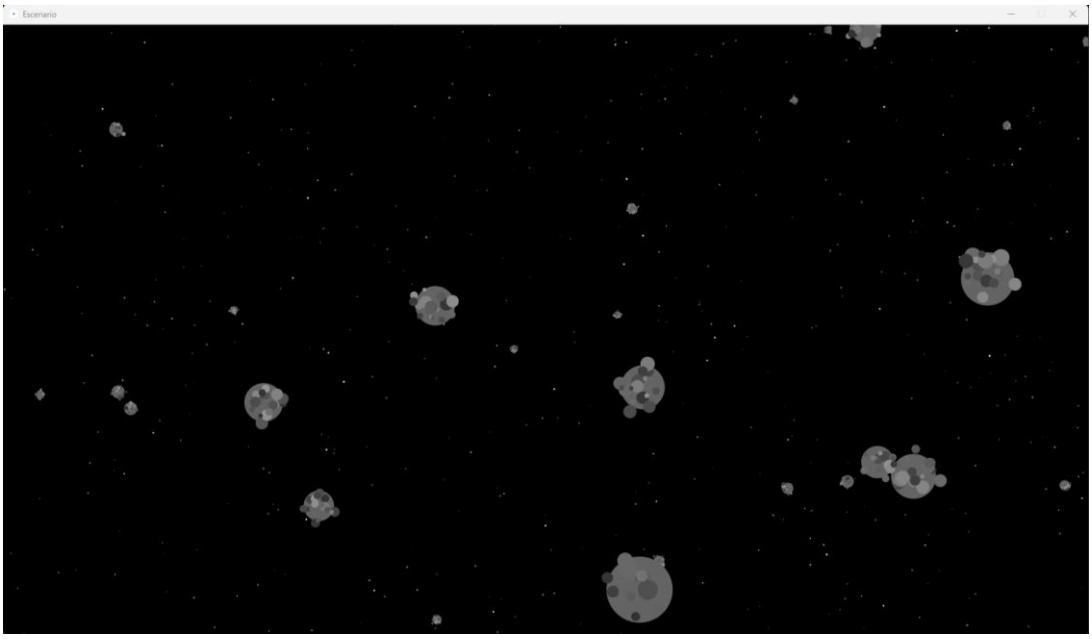
Este sistema de partículas es utilizado en dos ocasiones: al desplazar la nave y al matar a un ovni. En el primer caso, el sistema de partículas emite a un ángulo de 180º respecto al ángulo en el que mira la nave, simulando un sistema de propulsión; gracias a la dispersión, las partículas no se emiten en línea recta, sino que da la sensación de una llamarada de fuego. En el segundo caso, en el instante en el que muere el ovni se generan 500 partículas, cada una con un ángulo aleatorio, de manera que se distribuyen circularmente alrededor del centro de emisión del sistema, que es el centro del ovni.



6. Asteroides

Finalmente, el grupo de los asteroides representa a un conjunto de asteroides de distintos tamaños que recorren la pantalla trazando un movimiento sinusoidal. Cada asteroide está dibujado usando Processing y se compone de una serie de elipses con distinto sombreado. Cuando un asteroide sale de la pantalla, reaparece en el lado opuesto.

El escenario tiene tres grupos de asteroides de distinto tamaño y número de asteroides: el grupo más pequeño es el que tiene más asteroides y se dibujan encima de la capa del sol; el siguiente grupo tiene un tamaño intermedio y menor cantidad, se dibuja sobre el resto de los planetas; finalmente, el grupo frontal es el que menos asteroides tiene, pero son de mayor tamaño, se dibuja encima de la nave. Esto permite dar una sensación de profundidad y entropía.



3. Sistema de Interacción

Para facilitar el manejo de la entrada del usuario, se ha agrupado la gestión del ratón y del teclado en el fichero `Inputs.pde`.

Las primeras dos funciones de este archivo gestionan la interacción con el ratón. El método `triggerRaton()` evalúa si el ratón ha sido pulsado dentro de un área rectangular determinada por una posición, un ancho y un alto. Tanto si ha sido pulsado como si no, si el ratón se encuentra dentro de dicha área, cambia el icono del cursor por una *mano*, para facilitar al usuario identificar que se encuentra en una región clicable. Como el ratón solo puede estar en un sitio a la vez, cuando la función detecta que el ratón se encuentra sobre el área, almacena el objeto que la ha llamado, de manera que podrá reestablecer el cursor a una *flecha* cuando sea llamada por dicho objeto y ya no se encuentre en el área. El método `disableTriggerRaton()` sirve para deshabilitar manualmente el *trigger* de un objeto; esto es necesario, por ejemplo, en los *ovnis*, ya que se están moviendo constantemente y, si no se desactivase, la colisión quedaría desincronizada respecto al *sprite*. También lo usan los planetas para evitar que puedan ser pulsados mientras se están ejecutando sus animaciones.

Por otro lado, el método `triggerTecla()` controla la interacción con el teclado. Este método comprueba si una determinada tecla ha iniciado su pulsación durante el *frame* actual. El método es necesario porque el comportamiento por defecto de `keyPressed` es devolver `true` mientras la tecla está siendo pulsada, en lugar de hacerlo al inicio de la pulsación, lo cual puede dar lugar a efectos inesperados, como que se creen varias estrellas fugaces con una sola pulsación, especialmente si la aplicación se ejecuta a una alta tasa de fotogramas por segundo.

4. Conclusión

En esta memoria se ha presentado el diseño de un escenario espacial interactivo desarrollado en Processing. El escenario está compuesto por diversos elementos y efectos, como la simulación de planetas tridimensionales o el control de una nave interactiva. Considero que esta práctica ha sido muy interesante, ya que me ha permitido explorar y conocer de una manera creativa la plataforma Processing, que puede llegar a ser muy útil en el futuro, sobre todo en su versión de JavaScript, para el desarrollo web.