

# JavaScript - Juego Nave Espacial

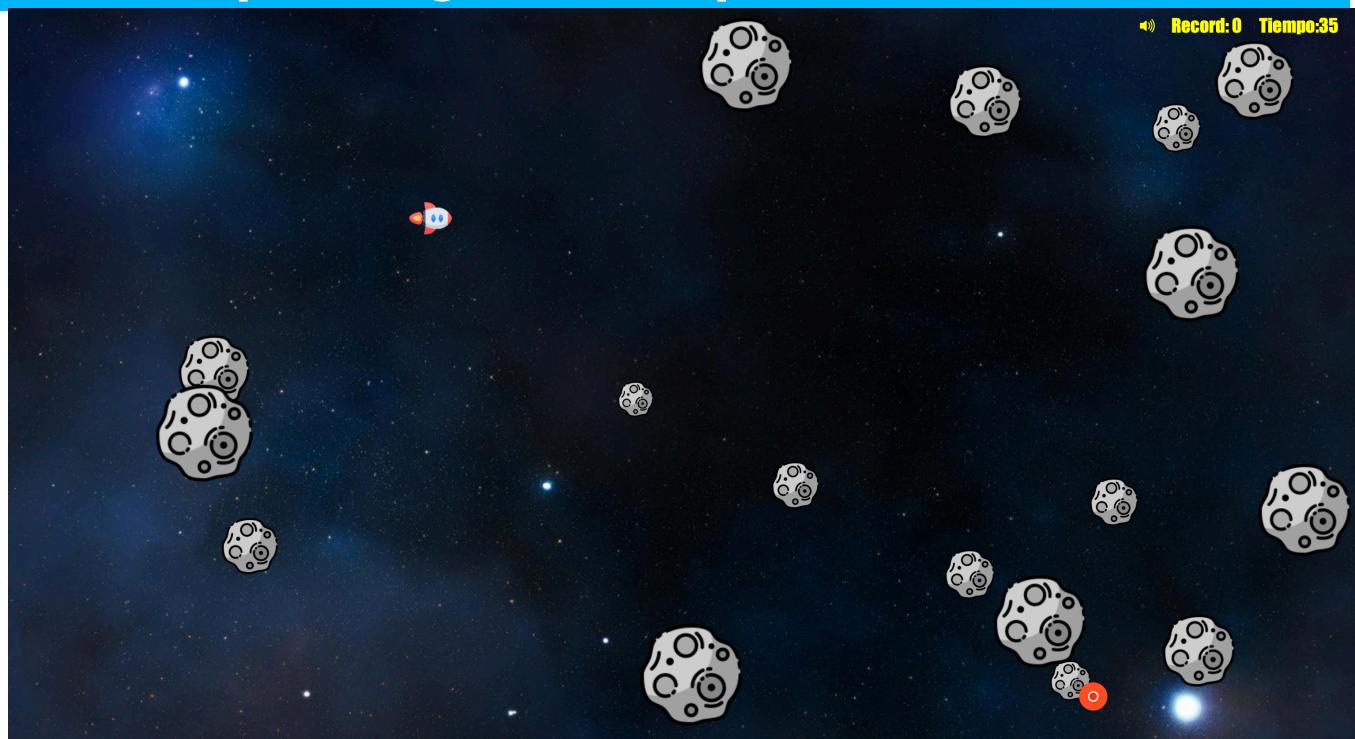
## Tabla de contenido

<i>Introducción</i> .....	1
<i>Estructura del proyecto</i> .....	2
<i>Archivo index.htm inicial</i> .....	2
<i>Archivo css/estilos.css inicial</i> .....	3
<i>Cambiando el puntero del ratón</i> .....	3
<i>Creando los meteoritos</i> .....	4
<i>Mover los meteoritos</i> .....	5
<i>Reloj de tiempo</i> .....	6
<i>Detectar impactos</i> .....	6
<i>Añadir música y sonidos al juego</i> .....	7
<i>Botón Iniciar</i> .....	8
<i>Sonido de explosión</i> .....	9
<i>Aumentando la dificultad (más meteoritos)</i> .....	9
<i>Añadiendo la cápsula</i> .....	9
<i>Icono de la página</i> .....	10
<i>Evitando trampas</i> .....	10

## Introducción

Vamos a crear un juego donde una nave, controlada por el ratón, debe esquivar asteroides, de diferentes tamaños y velocidades; cuanto más duremos más asteroides irán apareciendo. Además de vez en cuando aparecerá una cápsula, la cual, si la recogemos disminuirá el número de meteoritos posteriores.

# JavaScript - Juego Nave Espacial



En lugar de poner todo el código, vamos a ir dando pistas de como realizarlo.

Lo ideal sería usar una etiqueta canvas para este proyecto, pues tiene más posibilidades y rendimiento, pero aún no la hemos estudiado y es más sencillo del modo que vamos a ver.

## Estructura del proyecto

Tendremos las siguientes carpetas:

- css: contendrá el archivo estilos.css
- fuentes: contendrá las fuentes usadas en el proyecto
- sonidos: contendrá los sonidos y música del juego
- imágenes: contendrá las imágenes del juego
- js: contendrá el archivo juego.js del juego
- favicon: para meter el favicon
- index.htm: el archivo HTML del juego

## Archivo index.htm inicial

- Creamos un fichero HTML 5 que incluya la CDN de Bootstrap Icons para los iconos del sonido.
- Creamos una etiqueta header con lo siguiente:
  - o Un icono de sonido, con id **iconoSonido**, que buscaremos en <https://icons.getbootstrap.com/>. Yo he usado bi-volume-up-fill
  - o Un texto Record: seguido de un span con el id **record** y el valor 0
  - o Un texto Tiempo: seguido de un span con el id **tiempo** y el valor 0
- Un img con id puntero con la imagen del cohete (es un PNG animado). La usaremos para cambiarla por el puntero del ratón
- Para la parte principal del juego usaremos la propia etiqueta body

# JavaScript - Juego Nave Espacial

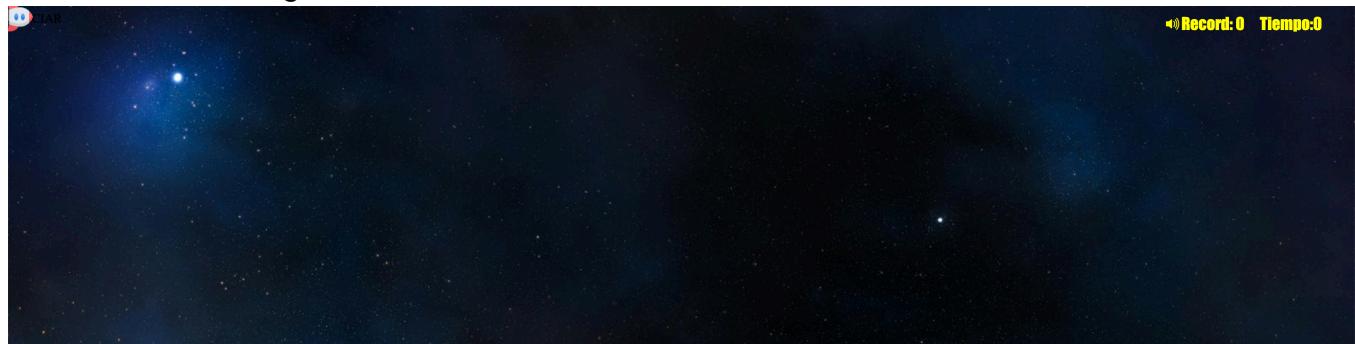
## Archivo css/estilos.css inicial

- Cargamos la fuente Nulshock Bd.otf y le damos el nombre deseado
- Creamos una regla para la etiqueta body donde ponemos:
  - o La imagen de fondo espacio.jpg
  - o Le damos el tamaño de cover para que cubra todo el body (background-size)
  - o Le damos una altura de 100vh, ya que si no el body solo ocupa el espacio donde haya cosas
  - o Damos el valor none a la propiedad cursor para ocultar el cursor del ratón
  - o **Ponemos la propiedad overflow a hidden para que no aparezcan barras de scroll si movemos la nave al borde derecho o inferior.**

Ahora queremos colocar los datos del encabezado en la parte superior izquierda. Para ello vamos a usar posicionamiento absoluto, creando una regla para el header con lo siguiente:

- Posicionamiento absoluto
- La colocamos arriba a la derecha con las propiedades top y right a cero
- Color amarillo, tipo de letra Impact, negrita, tamaño 1.3rem y espacio interno de 10px
- A sus span les damos un margen derecho de 1rem

Deberíamos tener algo como esto:



## Cambiando el puntero del ratón

Podríamos asignar directamente a la propiedad cursor de la etiqueta body una imagen y no tener que hacer nada más, pero no podemos cambiar el tamaño de la imagen, que sea animada, ...

```
cursor: url(..../imagenes/nave.png);
```

## Archivo css/estilos.css

Creamos una regla para nuestro id puntero:

- Posicionamiento fijo para poder moverla libremente y que no le afecte un posible scroll (aunque en este juego valdría también absolute)
- Un ancho de 50px y un alto de 40px
- Un z-index a un número alto para que quede por encima de cualquier otro elemento de la página
- Poner a none la propiedad pointer-events para que la imagen de nuestro puntero no reciba los clics del ratón y si los elementos de debajo (para que funcione como un puntero normal)
- La trasladamos a -50%, -50% para que la imagen se centre en la posición real del puntero del ratón y no se quede en la parte superior izquierda del mismo

# JavaScript - Juego Nave Espacial

## Archivo js/juego.js

Ahora debemos hacer que la imagen siga los movimientos del ratón. Para ello:

- Creamos una constante global llamada **puntero** con la imagen con el id puntero.
- Suscribimos el objeto document al evento mousemove, donde cambiamos las coordenadas de la imagen a las del puntero del ratón:

```
puntero.style.left = e.clientX + "px";
puntero.style.top = e.clientY + "px";
```

Debemos añadir px pues clientX solo incluye la posición en píxeles (el número), mientras que las propiedades left y top necesitan una unidad de medida (como todas las unidades de CSS).

Si probamos la página el puntero del ratón ya debería ser la nave.

## Creando los meteoritos

Vamos a ir creando, en tiempo de ejecución con JavaScript, meteoritos dentro de etiquetas div.

## Archivo css/estilos.css

Creamos una clase **meteorito** con lo siguiente:

- La imagen del meteorito
- Posicionamiento absoluto o fijo
- Un tamaño de contain para que se adapte al tamaño del div, pues vamos a ir creando meteoritos de diferentes tamaños, con lo que no nos interesa poner píxeles aquí
- Un aspecto-ratio de 1 para que, al crear el div con un ancho, el alto se adapte para que no se deforme la imagen. Así nos ahorraremos tener que darle un alto en JavaScript. Lo mismo si indicásemos el alto y no el ancho

## Archivo js/juego.js

Primero creamos una variable global, llamada **numeroMeteoritos**, con el valor 5. Usaremos esta variable para ir incrementando el número de meteoritos según avance el juego.

Creamos una función llamada crearMeteorito con lo siguiente:

- Generamos un número entre el número de meteoritos y 1 y lo guardamos en una constante, ya que no queremos que se cree siempre el mismo número de meteoritos y así darle más aleatoriedad al juego. Recordemos que la fórmula para generar números aleatorios entre dos números, ambos incluidos, es:

```
Math.floor(Math.random() * (valorMáximo - valorMínimo + 1) + valorMínimo);
```

- Creamos un bucle for desde 0 hasta el número generado (sin incluirlo), o desde 1 hasta el número generado incluido
- Dentro del for:
  - o Creamos un div
  - o Le añadimos la clase meteorito
  - o Lo añadimos al body como hijo

# JavaScript - Juego Nave Espacial

- Le damos un ancho al div (style.width) ya que, si no, no se verá, pues el div está vacío. Para crearlos de diferentes tamaños también generamos un número aleatorio entre 128 y 32 (u otros valores) al que luego hay que añadir "px" al asignarlo al width (algo parecido a como hicimos en el puntero)
- También debemos crear los meteoritos en diferentes alturas. Para ello asignamos a la propiedad style.top otro valor generado aleatoriamente entre 0 y 100 y añadirle el valor "vh" (alto del viewport / zona disponible de la página)

Llamamos a la función crearMeteorito para que se ejecute. Si probamos deberíamos tener varios meteoritos (entre 1 y 5) en la zona izquierda de la ventana, a diferentes alturas y tamaños. Podemos recargar la página varias veces para comprobar la aleatoriedad.

Para que los meteoritos aparezcan en la parte derecha, vamos a crear, dentro del bucle anterior, otro número aleatorio entre 120 y 100 del vw y asignarlo a la propiedad style.left del meteorito. Con esto conseguimos que se creen fuera de la pantalla. Si probásemos ahora el resultado, no se verían los meteoritos.

## Mover los meteoritos

### Archivo css/estilos.css

Creamos una animación que mueva el meteorito desde la parte derecha hasta la izquierda (algo más para que desaparezca de vista y no de golpe).

La llamamos animacionMovimientoMeteorito y, en su 100%, le damos el valor -64px a su propiedad left para que desaparezca saliendo de pantalla.

Lo ideal sería usar un valor según el tamaño del meteorito, usando Web Animation Api de JavaScript, pero no nos vamos a complicar.

Asignamos esa animación a la clase meteorito con 2 segundos y linear (lineal) para que vaya a la misma velocidad durante todo el recorrido.

Ya podemos probar y veremos que los meteoritos se crean y se mueven a la izquierda, pero no se eliminan realmente del body ni aparecen más.

### Archivo js/juego.js

Para lo primero, en el for donde creamos los meteoritos, no suscribimos al evento animationend del div del meteorito y creamos una función arrow donde eliminamos el div con su método remove().

También aprovechamos para cambiar la velocidad de la animación de cada meteorito, buscando de nuevo más aleatoriedad, dentro del for. Para ello creamos un número aleatorio entre 1 y 3 y se lo asignamos a la propiedad style.animationDuration del div, añadiendo "s" como hicimos anteriormente con "px".

Probamos y ya tenemos meteoritos a diferentes velocidades.

# JavaScript - Juego Nave Espacial

Ahora el tema es volver a generar meteoritos. Una opción sería crear un nuevo meteorito cuando desaparezca otro, pero vamos a optar por una opción sencilla que consiste en crearlos cada cierto tiempo. Para ello:

- Creamos una variable global **relojMeteoritos** con el valor null. Podríamos crearla sin valor, con lo que se le asignaría undefined, pero así queda más claro que no tiene valor inicial
- Usamos el método setInterval al que le pasamos el nombre de la función creaMeteoritos y el valor 1000 (1 segundo). Esto lo ponemos en lugar de la llamada a la función crearMeteoritos que pusimos anteriormente. setInterval seguirá llamando a la función mientras no paremos el "reloj"
- El valor que devuelve setInterval lo guardamos en la variable relojMeteoritos para poder detenerlo posteriormente

Probamos y ya tenemos meteoritos creándose y moviéndose continuamente.

## Reloj de tiempo

### Archivo js/juego.js

En el juego tenemos un marcador de tiempo para ver cuánto tiempo dura la nave sin ser destruida. Para ello:

- Creamos otra variable global **relojTiempo** con el valor null
- También otra llamada **tiempo** con el valor 0 para llevar la cuenta del tiempo
- Usamos otro método setInterval después del anterior, que llame a una función **incrementarTiempo** cada 500 (medio segundo) o menos tiempo o 1000 si queremos un segundo. Recordemos que estos valores son aproximaciones y JavaScript no asegura que se ejecute la función exactamente cada ese tiempo exacto.
- El valor que devuelve setInterval lo guardamos en la variable relojTiempo para poder detenerlo posteriormente
- Creamos la función incrementarTiempo donde:
  - o Incrementamos la variable tiempo en 1
  - o Cogemos el elemento tiempo (el span) por su id y le asignamos a su propiedadtextContent el valor de tiempo. Para evitar tener que obtener el span continuamente, podríamos crear una constante global con ese span y así obtenerlo solo una vez. Esto podemos aplicarlo también a explicaciones posteriores donde obtengamos el mismo elemento muchas veces

Probemos y ya tenemos el reloj de tiempo funcionando.

## Detectar impactos

### Archivo js/juego.js

Necesitamos detectar cuando la nave impacta con un meteorito. En nuestro juego es tan simple como usar el evento mouseenter de los div meteoritos, que se ejecutará cuando el puntero del ratón "entre" en uno de ellos:

- En el for de la función crearMeteoritos añadimos el listener mouseenter al div y la función **impacto**
- Creamos la función impacto donde:

# JavaScript - Juego Nave Espacial

- Paramos los dos relojs, usando el método clearInterval dos veces, pasándoles entre paréntesis cada uno de los relojs
- Ponemos ambas variables a null, para que quede claro que las variables no tienen ahora valor. No es imprescindible, pero queda claro que ya no tienen valor.
- Cambiamos la imagen del elemento con id puntero a la de la explosión (usando la propiedad src). Tened en cuenta que el js se ejecuta desde la propia página HTML, con lo que la ruta debe ser en base a esta
- Cambiamos la imagen de fondo del body por la de espacioGris

Podemos probar y, cuando impacte la nave, se parará el reloj, se dejarán de crear meteoritos, y se pondrá el fondo gris.

**Nota:** este método de impacto usa las zonas rectangulares de los div de los meteoritos, pero como estos no son rectangulares, en ocasiones la nave impactará, aunque visualmente no toque el meteorito. No vamos a cambiarlo, pues otros métodos son más complejos (varias zonas rectangulares, zonas circulares, comparar los propios píxeles no transparentes, ...) y para este juego nos vale.

## Añadir música y sonidos al juego

### Archivo js/juego.js

Creamos las siguientes constantes globales, **sonidoExplosion** y **musica**, a las que asignamos el audio deseado, usando new Audio("ficheroSonido"). Tened en cuenta que el js se ejecuta desde la propia página HTML, con lo que la ruta debe ser en base a esta.

También le asignamos el valor true a la propiedad loop de la constante musica para que la música se reproduzca indefinidamente, aunque va a ser difícil que duremos tanto tiempo en el juego para que esta terminase.

Ponemos en marcha la música del juego con el método **play()** de la constante musica, después de los setInterval y así suene desde el principio.

Vamos a permitir parar y reanudar los sonidos del juego mediante la tecla s. Para ello nos suscribimos al listener **keydown** del document.body y creamos la función **controlarSonido**, o una arrow, donde:

- Necesitamos el parámetro **evt** en la función
- Si la propiedad key de evt es "s", comprobamos si la propiedad **volume** de la variable musica o sonidoExplosion es 1. Otra opción es comprobar si la propiedad **paused** es true
- Si es 1, ponemos esa propiedad a 0 en ambas propiedades. Otra opción serían usar el método pause() de ambas. También eliminamos la clase bi-volume-off-fill del elemento con id iconoSonido con su método classList.remove y añadimos la clase bi-volume-up-fill
- Si es 0, ponemos esa propiedad a 1 en ambas propiedades. Otra opción serían usar el método play() de ambas. También eliminamos la clase bi-volume-up-fill del elemento con id iconoSonido con su método classList.remove y añadimos la clase bi-volume-offs-fill
-

# JavaScript - Juego Nave Espacial

## Botón Iniciar

En lugar de que el juego comience inmediatamente vamos a crear un "botón" Iniciar para que la partida comience cuando el usuario lo pulse. Además, volverá a aparecer cuando termine la partida y así poder volver a comenzar.

### Archivo index.htm

Añadimos un div con id **iniciar** y el texto INICIAR al body

### Archivo css/estilos.css

Creamos una regla para el id iniciar con lo siguiente:

- Para centrarlo en la ventana:
  - o Posicionamiento fixed o absolute (en este juego da igual)
  - o top y left a 50% para colocar el elemento a partir de la posición central
  - o lo trasladamos -50%, -50% para colocarlo justo en el centro de la posición central
- color amarillo, tamaño 4rem y tipo de letra la que cargamos al comienzo (la de NullShock db) y una sans-serif de alternativa
- Le asignamos una animación llamada animacionIniciar de 2 segundo e infinita

Creamos la animación:

- Al 50% la agrandamos en un 20% y la volvemos a transladar un -50%, -50% para que crezca centrada y no hacia la derecha

Probamos que funcione.

### Archivo js/juego.js

Le añadimos un listener al evento click de ese div iniciar (después de lo setInterval e instrucciones siguientes) a una función **iniciar**.

En la función iniciar con el parámetro evt:

- Ocultamos el evt.currentTarget asignando el valor "none" a su propiedad display del estilo. También podríamos coger el id iniciar si no queremos usar evt
- Cortamos las líneas donde hacíamos los dos setInterval y las metemos aquí
- Ponemos a cero la variable tiempo
- Ponemos a 5 la variable numeroMeteoritos
- Ponemos la música en marcha con play() en la variable musica
- Mostramos el tiempo en la propiedad textContent del id tiempo (el span)
- Ponemos la imagen del puntero del ratón de nuevo a la de la nave (a la propiedad src del id puntero)
- Ponemos de nuevo la imagen espacio a document.body

Para volver a mostrar este div de inicio cuando perdemos, vamos a la función impacto y pongamos a "block" la propiedad display del id iniciar.

Podemos probar y deberíamos poder iniciar la partida cada vez que perdamos

# JavaScript - Juego Nave Espacial

## Sonido de explosión

Ahora nos falta añadir el sonido de explosión cuando la nave impacta con un meteorito.

### Archivo js/juego.js

En la función impacto usamos el método play() en la variable sonidoExplosion y pause() en la de musica.

## Aumentando la dificultad (más meteoritos)

### Archivo js/juego.js

Según avance el juego vamos a ir añadiendo más meteoritos. Para ello al comienzo de la función crearMeteoritos comprobamos si la variable tiempo es divisible por 10 ( tiempo % 10). Si lo es incrementamos la variable del número de meteoritos en 1.

Con esto añadimos una posibilidad más de crear meteoritos cada 10 unidades de tiempo.

## Añadiendo la cápsula

Vamos a añadir una cápsula que la nave puede coger, con lo que se decrementará el número de meteoritos que se crean.

### Archivo js/juego.js

Primero creamos una constante sonidoCapsula con el sonido capsula.mp3 del mismo modo que hicimos con los otros sonidos.

En la función controlarSonido hacemos lo mismo que tenemos con los otros sonidos.

Creamos la función crearCapsula con o siguiente:

- Generamos un número aleatorio entre 1 y 5
- Si sale el número que queramos (lo elegimos nosotros) enter el 1 y 5:
  - o Creamos un div
  - o Lo añadimos al body
  - o Le añadimos la clase **capsula** (que crearemos luego)
  - o Generamos valores aleatorios para el top y left al igual que hicimos con los meteoritos
  - o Le añadimos un listener al evento mouseenter que llame a la función capsulaCapturada

Creamos la función capsulaCapturada con lo siguiente:

- Debe recibir el parámetro evt
- Dividimos la variable numeroMeteoritos entre 2 y guardamos el resultado en la misma variable
- Reproducimos el sonido de la capsula
- Si el número de meteoritos que quedan es menor que 5, le damos el valor 5 a esa variable. Así quedan siempre unos cuantos.
- Eliminamos el div de la capsula con el método remove del currentTarget

# JavaScript - Juego Nave Espacial

Por último, al final de la función crearMeteoritos añadimos una llamada a la función crearCapsula.

## Archivo css/estilos.css

Creamos una regla para la clase capsula con lo siguiente:

- Posicionamiento fijo o absoluto
- Imagen de fondo capsula.png
- Tamaño de la imagen contain
- Ancho y alto de 32 píxeles
- Le asignamos la misma animación de los meteoritos (o creamos una diferente si queremos)

## Icono de la página

Accedemos a la siguiente web y creamos el icono para nuestra página:

<https://realfavicongenerator.net>

- Pulsamos en el botón Pick your favicon image
- Elegimos la imagen del meteorito
- En la parte inferior de la página, en el cuadro de texto Favicon path, escribimos favicon para poder tener todos los iconos y ficheros del favicon en una carpeta separada. No tiene que ser ese nombre
- Pulsamos Next y luego Download
- Descomprimimos la carpeta descargada y metemos el contenido en una carpeta, llamada favicon, que creemos en nuestro sitio. Debe ser en la carpeta raíz del proyecto para que se cargue cuando probemos con Live Server o un servidor real

## Evitando trampas

Si movemos el puntero del ratón fuera de la ventana, la nave no se verá afectada por los meteoritos mientras esto pase, pues las colisiones las detectamos cuando el puntero del ratón entra en un meteorito.

## Archivo js/juego.js

La opción más simple es suscribirnos al evento mouseout del objeto document y usar una función arrow donde llamemos a la función impacto. Así, si el puntero sale de la zona del juego, la nave explotará.

En esa función arrow, o una función normal, primero haremos esta comprobación para asegurarnos de que se salga del body:

```
if (!evt.relatedTarget || evt.relatedTarget.nodeName === "HTML") {  
    // Llamar a la función impacto  
}
```

# JavaScript - Juego Nave Espacial

Otro problema grave del juego es que, si no movemos el ratón, el evento mouseenter no se ejecuta, con lo que si dejamos la nave quieta, no chocará con ningún meteorito.

La solución es usar un sistema clásico de colisiones donde recorramos todos los meteoritos y veamos si su rectángulo interseccióna con el rectángulo de la nave. Esto podríamos hacerlo en un método que ejecutemos cada muy poco tiempo con setInterval o, mejor aún, usar el método **requestAnimationFrame** al que pasaremos la función a ejecutar. Con ella el navegador llama a nuestra función cuando quiera redibujar la pantalla.

Dentro de la función, que he llamado comprobar, recorreremos todos los elementos de la clase meteorito, obtendremos su área rectangular y la compararemos con la de la nave. Si interseccionan, llamaremos a la función impacto:

```
function comprobar() {
    const rectNave = puntero.getBoundingClientRect();

    Array.from(document.getElementsByClassName("meteorito")).forEach((m) => {
        const recMeteo = m.getBoundingClientRect();

        const noColisionan =
            rectNave.right < recMeteo.left ||
            rectNave.left > recMeteo.right ||
            rectNave.bottom < recMeteo.top ||
            rectNave.top > recMeteo.bottom;
        if(noColisionan == false) {
            impacto();
        }
    });
    // Debemos llamar de nuevo a este método al final de la función
    requestAnimationFrame(comprobar);
}
```

Haciendo esto podríamos quitar los eventos mouseenter de los meteoritos.

También podríamos comprobar si colisiona con la cápsula dentro de la función comprobar.