# Juego “Cartas de memoria”

## Objetivo:

El objetivo es realizar el clásico juego de cartas de memoria, donde el jugador debe elegir dos cartas, que se mostraran por un breve instante, con el objetivo de encontrar el par igual (en ese caso quedan ya “boca arriba”).

El juego lo haremos con HTML y JavaScript.

## Conceptos e ideas que se trabajarán:

- Array de objetos.
- Ordenamiento randomico.
- Manejo del DOM.
- Funciones CALLBACK con tiempo.

## Implementación:

### Creación de las cartas:

Debemos crear una array que contenga todas las cartas que vamos a utilizar. En este caso serán 8. Tenemos que tener 4 diseños diferentes que se repitan de a pares.

```
const arrayCartas = [
        {
                dibujo: 'Daredevil',
                imagen: 'img/daredevil.jpg'
        },
        {
                dibujo: 'Daredevil',
                imagen: 'img/daredevil.jpg'
        }
]
```

En este ejemplo, se muestra una de las cartas cargadas al array. La “clave” será el atributo “dibujo”, que nos servirá para comparar si tenemos “match”. Podría tener otras resoluciones este aspecto. Vemos que se repite, porque como habíamos mencionado, crearemos el par de cartas. A la hora de diferenciarlas, tomaremos como referencia si índice en el array.

## La tabla de HTML

Manejaremos una tabla, donde se mostrarán las cartas. Para ello, será importante y de gran ayuda, determinar cada celda con un id, comenzando desde 0, lo que nos permitirá tenerlas como referencia hacia las posiciones del array.

```
<td id=0></td>
```

## Las imagines de las cartas, y el cambio

En cuanto a las imágenes, tenemos las que ya declaramos en el array, que veremos cómo usarlas, y el reverso de las cartas. Vamos a declarar dos estilos en CSS para las cartas:

```
.carta{
        display: inline-block;
        height: 250px;
        width: 190px;
}

.cartaReverso{
        background: url("img/cartaReverso.png");
        background-size: cover;
}
```

Estos estilos serán de vital importancia en este ejemplo, ya que no utilizaremos el elemento <img> en esta oportunidad, sino que, para las celdas, utilizaremos “background”.

Dicho esto, para poder asignar la imagen inicial a la carta, haremos lo siguiente:

```
document.getElementById(idCelda).setAttribute("class", "carta cartaReverso");
```

Utilizamos una de las propiedades del DOM, que es setAttribute.

## setAttribute

El método setAttribute () agrega el atributo especificado a un elemento y le da el valor especificado. Recibe dos parámetros, el atributo y el valor.

Si el atributo especificado ya existe, solo se establece / cambia el valor.

En este ejemplo, recordamos que cada celda tiene un id, debemos sustituir ***idCelda*** por cada celda que creamos.

Cambiando la imagen:

```
document.getElementById(idCelda).style.backgroundImage =
"url('"+arrayCartas[i].imagen+"')";
```

Lo que hacemos en este ejemplo, es utilizar otra de las propiedades de DOM, que es style. Podíamos haber utilizado también setAttribute, pero no es recomendado en estos casos.

## style

La propiedad de estilo devuelve un objeto CSSStyleDeclaration, que representa el atributo de estilo de un elemento.

La propiedad de estilo se usa para obtener o establecer un estilo específico de un elemento usando diferentes propiedades de CSS.

# Funciones

## Mezclar las cartas

```
function mezclarCartas(){
        arrayCartas.sort(() => Math.random() - 0.5);
        for (var i =0; i < arrayCartas.length ;i++) {
              document.getElementById(i).setAttribute("class", "carta
cartaReverso");
        }
 }
```

Esta función se llamará al inicio y cada vez que reiniciemos el juego.

Hace dos cosas fundamentalmente:

- Ordena nuestro array que declaramos al comienzo de forma randomica
- Le asigna los estilos de cartas y reverso de cartas a las celdas.

## Mostrar y ocultar carta

Estas dos sencillas funciones, simplemente cambiaran el estilo del fondo de la celda.

```
function mostrarCarta(i){
       document.getElementById(i).style.backgroundImage =
"url('"+arrayCartas[i].imagen+"')";

 }

 function ocultarCarta(i){
       document.getElementById(i).style.backgroundImage =
"url('img/cartaReverso.png')";
 }
```

Reciben como atributo el id de la celda (también sería la posición en el array de la carta)

# Jugada

Aquí se centrará el algoritmo principal del juego:

```
function elegirCarta(i){
        mostrarCarta(i);
        if (seleccionadas == 0){
                cartaAnterior = i;
                seleccionadas = 1;
        }else{
                if(arrayCartas[cartaAnterior].dibujo === arrayCartas[i].dibujo){
                        aciertos++;
                }else{
                        setTimeout(() => {  ocultarCarta(cartaAnterior);
ocultarCarta(i); }, 500);
                }
        seleccionadas = 0;
        movimientos++;
        }

        document.getElementById("movimientos").innerHTML = movimientos;

        if (aciertos == 4){
                document.getElementById("ganador").innerHTML = "GANASTE!!!!";
        }

 }
```

## *Descripción de la función:*

Recibe como parámetro la ubicación de la carta, que como ya dijimos, cuenta tanto para la ubicación en la tabla HTML como en el array de cartas.

Lo primero es “mostrar la carta”, utilizando la función que ya habíamos creado.

Luego resuelve el algoritmo para ver si hay “match”, apoyándose en variables que deberíamos tener declaradas de forma global.

Si las cartas son iguales, las deja fijas mostrando su imagen, de lo contrario, las vuelve a poner de reverso. Para esto, utilizamos una función por tiempo.

# setTimeOut()

El método setTimeout () llama a una función o evalúa una expresión después de un número específico de milisegundos.

Recibe dos parámetros, la o las funciones o expresiones, y el tiempo en milisegundos.

En este ejemplo, recibe dos funciones. Son las llamadas funciones CALLBACK, ya que se ejecutan cuando termina la función setTimeOut().