***Índice***

[**Función *IniciaJuego()***](#_7tpscnjyh2dh) **2**

[**Función *cargarPuzzle()***](#_tr200zocnh84) **2**

[**Función *comprobarPuzzle()***](#_xz51zje2be18) **4**

[**Función *desordenarPiezas(numeros)***](#_q06vooujweod) **5**

### **Función *IniciaJuego()***

Esta función es la función principal del script y engloba dos de ellas que tiene en su interior: *cargarPuzzle()* que es el groso del funcionamiento y *comprobarPuzzle()* que también es muy importante. De esta cabe destacar que tiene las variables *“contenedorPuzzle”* la cual es el elemento “div” con *display grid* que tendrá las piezas para el *drag and drop*, un array *“piezas”* para guardar los correspondientes elementos div que tendrá el contenedor y una variable *”intentos”*. Esta se hace cargo de llamar a la función *cargarPuzzle()*. La razón por la que esta función tiene otras dos funciones en su interior se debe a que estas deben poder acceder a las variables de la principal en todo momento para cualquier comprobación y manipulación de los datos, por lo que es una buena opción.

#### **Función *cargarPuzzle()***

Esta función tiene dos variables: Un array *“numeros”* con valores del 1 al 16 para corresponder los valores a cada elemento “div” del *“contenedorPuzzle”* y un objeto *“imagenes”* que contiene claves valor cuyas claves van del 1 al 16 para relacionar los valores con los del array de números y cuyos valores son la ruta de las imágenes troceadas indexadas en orden. En primer lugar se desordena el array convirtiendo el array con la función *desordenarPiezas(numeros)*, luego se recorre el array de números y por cada vuelta:

1. Se crea un elemento “div” al que se le da un valor con la propiedad “textContent” obtenido del array *“numeros” desordenado*, se activa la propiedad “draggable” para que el elemento pueda ser arrastrado y se le aplica una clase CSS con la propiedad className. Por último se obtiene la imagen correspondiente a la clave del objeto donde la clave es igual al valor asignado a esa pieza y mediante la propiedad style.backgroundImage se le asigna el valor correspondiente del objeto a la pieza. (Para no visualizar también el valor asignado con textContent en la pieza se ha usado un truquito CSS).
2. Para la primera carga de las piezas en el puzzle se necesita dar formato de colores al borde de cada pieza para indicar su situación en el tablero. Por lo que se obtiene la *“filaActual”* y *“columnaActual”* con una operación para la pieza y su valor con textContent para comprobar que esté situada en su casilla correspondiente para añadirle la clase CSS “pieza\_verde”, que esté en su fila o columna pero no en su casilla correspondiente para asignarle la clase CSS “pieza\_naranja” o la clase CSS “pieza\_gris” si no se cumple ninguna de las comprobaciones.
3. Asignamos los *eventos “dragstart”,* *“dragover”* *y* *“drop”* para cada *“pieza”*. Para la lógica de arrastrar y soltar obtendremos el valor de la pieza con la propiedad “dataTransfer” y su función *setData()* con el valor de la propiedad “textContent” del elemento. Mediante este valor podremos comprobar la posición de una pieza. Para el *evento “drop”* se obtiene el valor del elemento que estábamos arrastrando con la propiedad “dataTransfer” y su función *getData()* y también del elemento donde estamos soltando con *ev.target* para referirnos al mismo y la propiedad “textContent” para su valor. Creamos una variable *“índicePiezaArrastrada” y otra “indicePiezaSoltada”* y las inicializamos con un valor no válido. Recorremos el array de piezas en busca de la pieza que coincida con ambos valores para obtener sus índices para realizar el intercambio de piezas. Primero obtenemos los valores de las propiedades “textContent” y “backgroundImage” de la primera pieza y los guardamos en una variable de respaldo. Luego intercambiamos dichos valores con los de la pieza donde arrastramos y a esta se le asignan los valores de respaldo de la pieza arrastrada. Por cada vez que se produzca el intercambio entre piezas en un *evento “drop”* se llamará a la función *comprobarPuzzle()* para asignar las clases CSS oportunas mediante las mismas comprobaciones del principio.
4. Por último se añade la *“pieza”* al array *“piezas”* de la función *iniciaJuego()* y se añade dicho elemento al *contenedor grid* que contendrá todas las piezas al final del *bucle for*.

#### **Función *comprobarPuzzle()***

Esta función se ejecuta cada vez que se arrastra un elemento. Creamos una variable *“todasCorrectas”* seteada a true e incrementamos la variable *“intentos”* de la función principal. La lógica es la misma que en las comprobaciones en un *bucle for* para asignarle una clase CSS buscando la casilla correspondiente de la pieza junto a su fila y columna. En esta función lo que se diferencia es que si no todas las piezas entran en el primer if, quiere decir que hay alguna que no está en su posición correcta, por lo que se setea la variable *“todasCorrectas”* a false.

Al final de la función, fuera del *bucle for* se comprueba que la variable *“todasCorrectas”* sea true. Si es así se eliminan todas las clases CSS de los elementos *“pieza”* y los juntamos un poco más, por último centramos el contenedor con márgenes para mostrar un poco más atractivo el resultado del puzzle. Además se crea un elemento “h2” con un mensaje indicando el número de movimientos de la variable *“intentos”* en que hemos completado el puzzle.

### **Función *desordenarPiezas(numeros)***

Esta es la única función que está fuera de la función principal, por lo que debe recibir el array *“numeros”* en el momento del intercambio. En esta se recorre el array *“números”* mediante un *bucle for* y se crea la variable *“posicionAleatoria”* para obtener un valor aleatorio con *Math.random()* en cada vuelta. Se crea un respaldo para el valor de la variable *“posicionActual”* del array y se intercambia su valor por el de *“posicionAleatoria”*. Por último se devuelve el array *“numeros”* desordenado.