Programar un buscaminas en JavaScript con tu hijo

Este artículo es sobre todo para los hijos, sobrinos y demás gentes que no saben nada, nada de programación, y quiere ver cómo se hace. En este artículo voy a enseñar a hacer un juego muy simple de cero. No necesitas saber nada. Sólo un ordenador, un editor de textos y un navegador. Algo al alcance de cualquiera.

El otro día meditaba sobre cómo me había metido en esto de la informática y fue a raíz de que me regalaran un Amstrad CPC 6128. Me juntaba con otros dos amigos de clase que tenían también Amstrad. Ninguno destacábamos en ningún deporte, y preferíamos dedicar nuestros recreos a contarnos los avances que habíamos hecho. Poco a poco fuimos haciendo nuestros primeros programas en un BASIC muy básico. Todo suficiente para empezar a programar. Muchos desarrolladores se reconocerán en este párrafo.

El hecho es que a mi me gratifica mucho crear cosas. Y de esto va este artículo. El otro día nos sentamos juntos mi hija de 9 años y yo, y me puse a enseñarle cómo hacer un buscaminas de cero. Haciéndolo paso a paso, y respondiendo a sus preguntas. Hay que tener en cuenta que ella no sabe nada de HTML, CSS ni JavaScript. Pero iba haciendo pequeños cambios y entendiendo cómo funcionaba, que de eso se trata.

Y se me ocurrió escribir el camino que seguimos, pues a lo mejor pica a otros con el gusanillo de la programación. Espero que eso sirva al neófito para que, haciendo pequeñas modificaciones, aprenda los rudimentos.

1. Introducción.

¿por qué un Buscaminas? es lo suficientemente conocido para que todo el mundo sepa cómo funciona, y su simplicidad no oculta los elementos básicos: habrá que pintar una pantalla y habrá eventos que desencadenan acciones que tendrán consecuencias sobre la pantalla.

Los ladrillos básicos que emplearemos serán HTML, CSS y JavaScript.

HTML es la maquetación. Es como en un periódico: defines las columnas, los titulares, el la entradilla, los pies de foto, imágenes, etc...

Con CSS le das estilos: tipo de letra, colores, márgenes, etc...

Con JavaScript coges esos elementos básicos y haces cosas con ellos: los mueves, los transformas, les cambias su aspecto, etc...

En este tutorial veremos cómo:

- dibujar en pantalla el tablero con HTML y CSS
- implementar la lógica del juego en JavaScript
- vincular eventos del ratón a acciones concretas

2. Nuestra página HTML

En un directorio, creamos un archivo de texto llamado buscaminas.html

```
<!DOCTYPE html>
<html>
<head>
	<title>Buscaminas</title>
	<link href="css/estilos.css" type="text/css" rel="stylesheet">
	<meta charset="UTF-8">
	<script language="JavaScript" src="js/funciones.js"></script>
</head>
<body>

</body>
</html>
```

Esto es una página web vacía.

En la cabecera vemos que tenemos un título llamado Buscaminas

Indicamos una hoja de estilos que está en un subdirectorio “css/estilos.css” y un archivo de JavaScript en otro subdirectorio “js/funciones.js”

De momento esos archivos no existen, pero ahora los crearemos.

Si abre el archivo y lo abre con un navegador verá una página en blanco y la pestaña tendrá el nombre del título. Le animo a que lo cambie, grabe el fichero, y recargue el navegador. Verá cómo cambia.

3. Dibujar el tablero.

El buscaminas tiene un tablero con un número de filas y columnas. Tiene forma de tabla. Lo primero que vamos a hacer es añadir en el <body> un contenedor donde pintar el tablero.

```
<!DOCTYPE html>
<html>
<head>
 ...
</head>
<body>
	<div id="tablero">
	</div>
</body>
</html>
```

Bien, vemos que un es un contenedor, que se cierra con . Esta tiene un id=”tablero”, para poder identificarlo luego y llamarlo por su nombre. Cada casilla del buscaminas podría ser un div ¿no?

```
<div id="tablero">
  <div></div> <div></div> <div></div> <div></div>
  <div></div> <div></div> <div></div> <div></div>
```

```
  <div></div> <div></div> <div></div> <div></div>
</div>
```

Aquí hay 12 casillas. Son pocas pero nos sirve para ilustrar nuestro ejemplo.

Si guardamos y refrescamos la pantalla, no vemos nada. Sigue todo blanco. Es normal, no he hemos indicado a cada div un fondo, ni un borde ni nada. Tampoco le hemos indicado al tablero si los debe pintar en fila, uno debajo del otro o en columna. Eso se hace en css.

Vamos a crearnos el subdirectorio css y dentro un archivo de texto llamado estilos.css

En él indicamos que el tablero se mostrará en forma de rejilla (grid) y que debe tener 3 filas y 4 columnas de 32 píxeles de ancho y alto.

```
#tablero{
	display:grid;
	grid-template-columns: repeat(4, 32px);
	grid-template-rows: repeat(3, 32px);
}
```

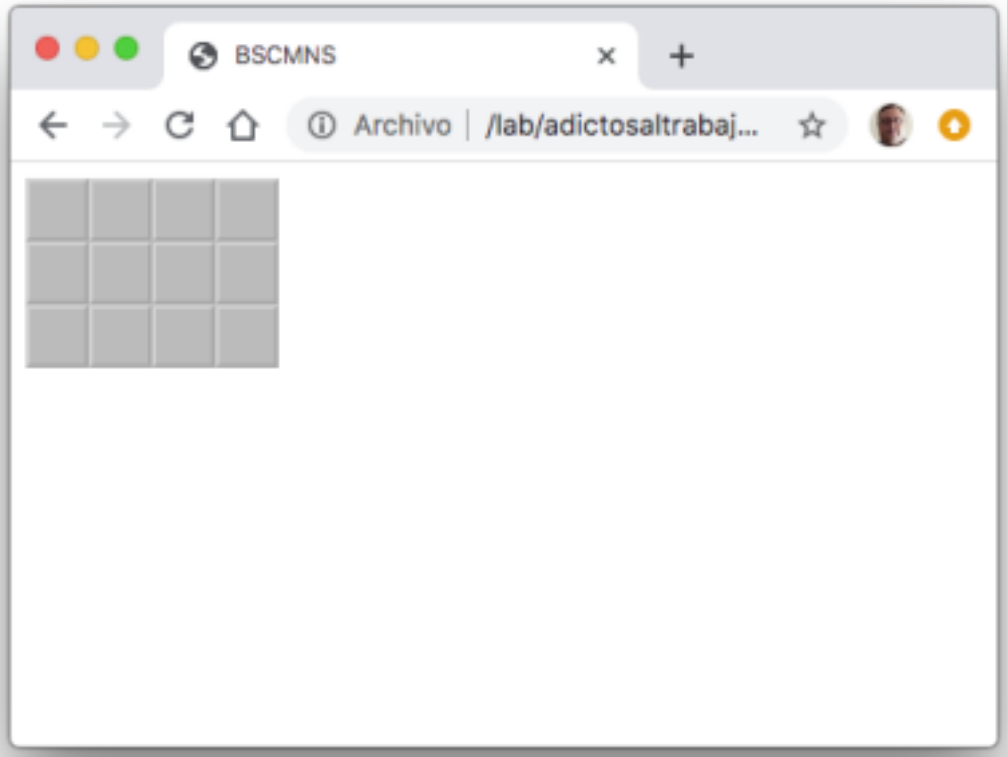
Además, de indicamos que todos los div dentro de tablero tendrán un color de fondo gris, y con un borde de 2 píxeles, arriba y a la izquierda un poco más claro y abajo y a la derecha un poco más oscuro, para dar sensación de bisel, de relieve.

```
#tablero div{
	background-color: #BBBBBB;
	border-left: 2px solid #D3D3D3;
	border-top: 2px solid #D3D3D3;
	border-right: 2px solid #A9A9A9;
	border-bottom: 2px solid #A9A9A9;
	text-align: center;
	line-height: 32px;
}
```

Esos `#A9A9A9` son colores RGB.

Además le hemos indicado que el texto tiene que estar centrado y que la altura de la línea es 32 píxeles también.

Si guardáis los archivos de texto y recargáis el navegador deberías ver:



Bien, y si en vez de una rejilla de 3x4 quisiéramos una rejilla de cualquier dimensión... Deberíamos poder pintar tantos div como casillas quisiéramos que tuviera nuestro buscaminas ¿no?

Para eso nos vamos a nuestro fichero HTML y vaciamos de divs el tablero. Ahora debemos rellenarlos programaticamente con los que queramos cada vez. Nos vamos al fichero js/funciones.js y creamos la función dibujarTablero.

```
function dibujarTablero(numFilas, numColumnas){
    let tablero = document.querySelector("#tablero");

    for(let f=0; f<numFilas; f++){
        for(let c=0; c<numColumnas; c++){
            let newDiv = document.createElement("div");
            tablero.appendChild(newDiv);
        }
    }
}
```

Esta función recibe el número de filas y el número de columnas por parámetro. Vemos que lo primero que hace es obtener el elemento tablero por su id y lo guarda en una variable

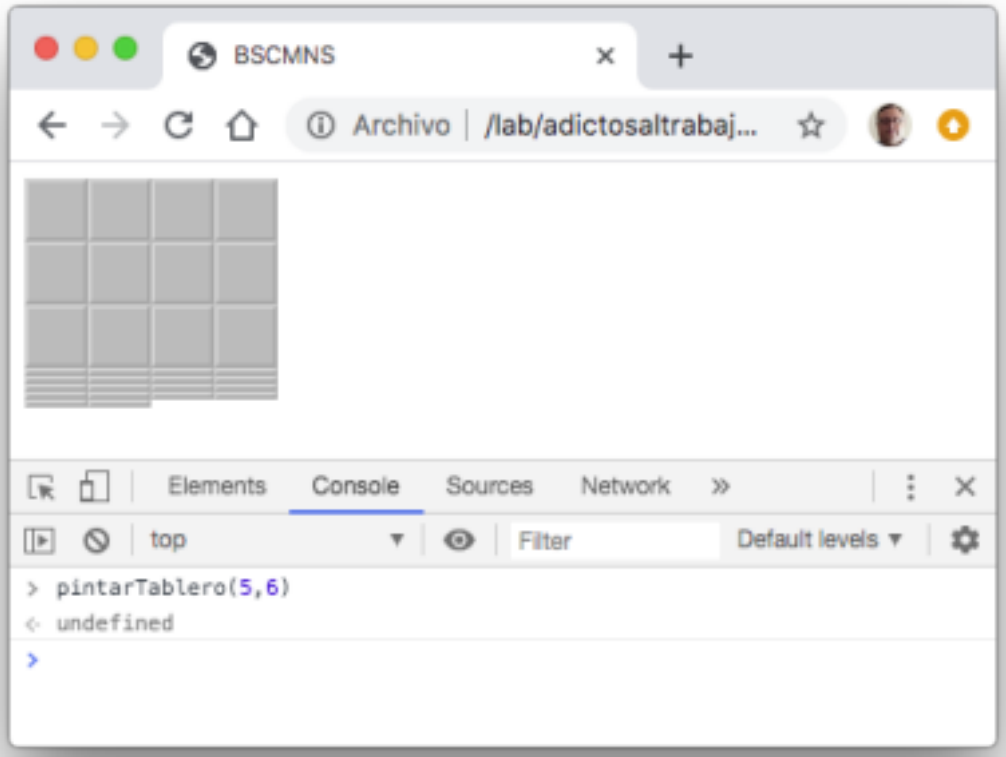
```
let tablero = document.querySelector("#tablero");
```

Luego para cada fila, para cada columna crea un div y se lo añade al tablero.

NOTA: vemos que el bucle *for* empieza en 0 y es que en JavaScript, los arrays, los vectores, empiezan en 0. Esta es propio de muchos lenguajes de programación.

Guardamos y recargamos el navegador. Y abrimos la consola del navegador con F12 o con Alt + Cmd + I para macOs.

En la consola, escribimos dibujarTablero(5,6)



Pero se ve raro... Vemos un grid de 4x3 y unas pocas casillas debajo... ¿qué ha pasado? Realmente hemos creado 5x6 divs... 30 casillas, pero la hoja de estilos ha intentado repartirlas en una rejilla de 4x3. Lo que tenemos que cambiar es la definición de la rejilla.

Para eso usaremos variables de CSS y las inicializaremos a unos valores por defecto.

```
:root{
      --num-columnas: 10;
      --num-filas: 10;
      --size: 32px;
}

#tablero{
      display:grid;
      grid-template-columns: repeat(var(--num-columnas), var(--size));
grid-template-rows: repeat(var(--num-filas), var(--size)); }

#tablero div{
```

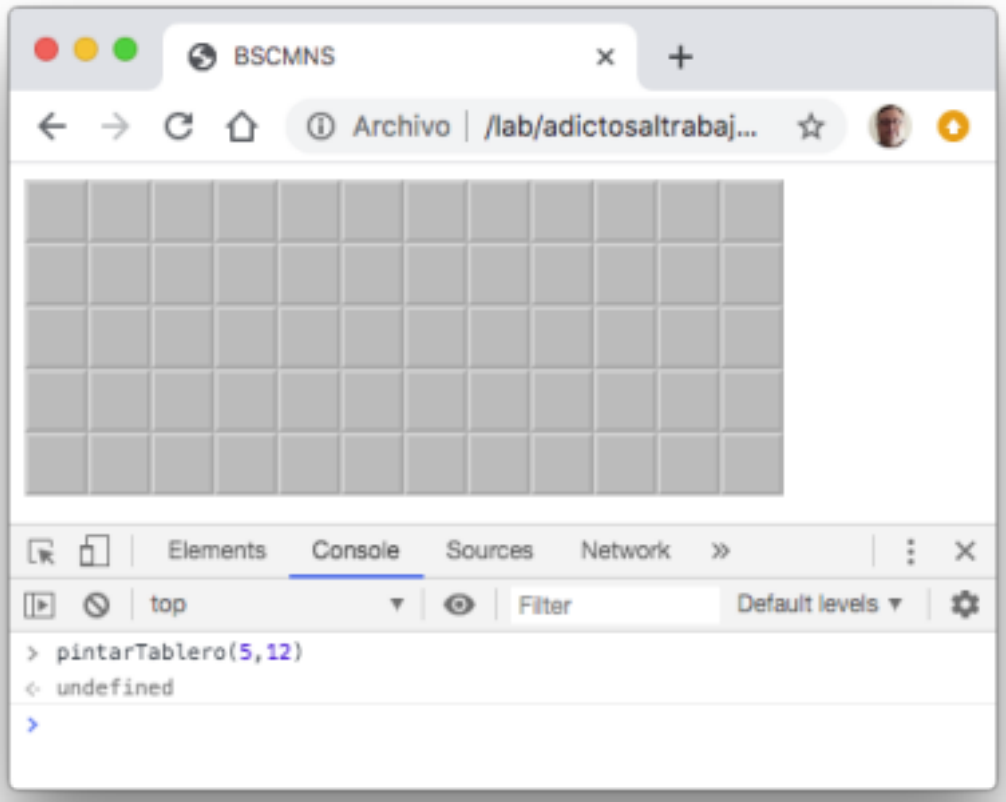
```
        background-color: #BBBBBB;
        border-left: 2px solid #D3D3D3;
        border-top: 2px solid #D3D3D3;
        border-right: 2px solid #A9A9A9;
        border-bottom: 2px solid #A9A9A9;
        text-align: center;
        line-height: var(--size);
}
```

Ahora, nuestra función de JavaScript deberá poder cambiar el valor de esas variables CSS.

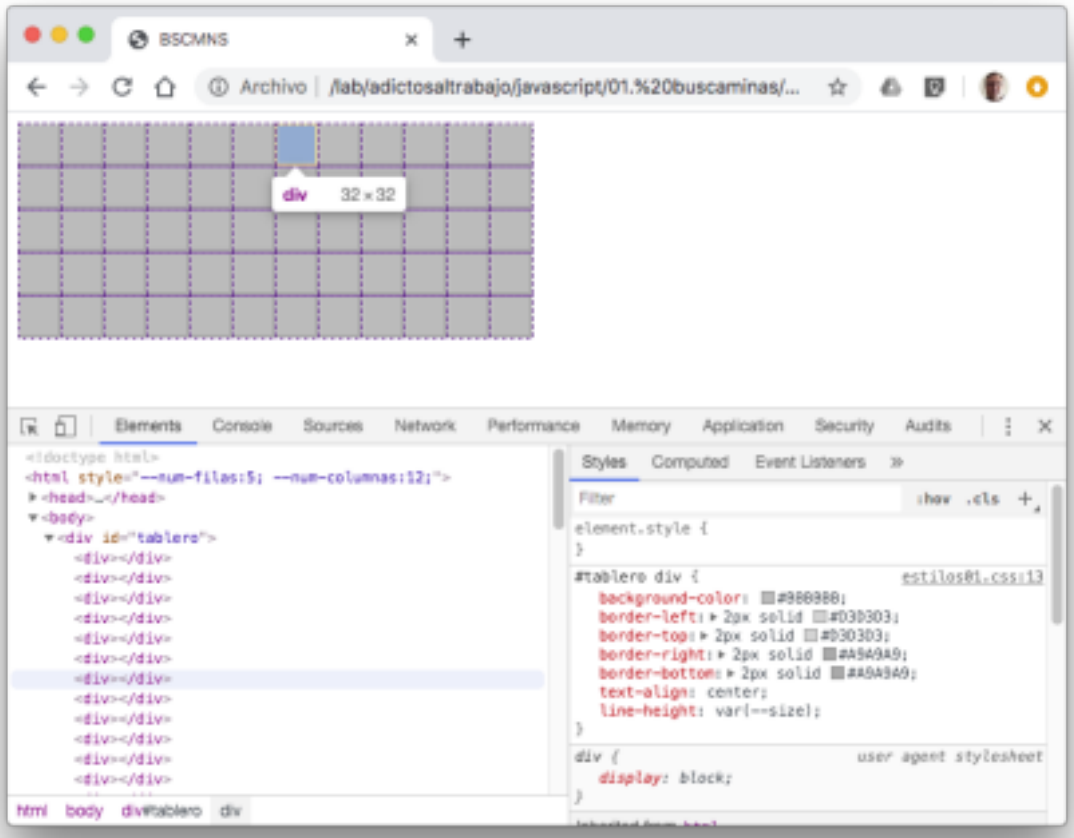
```
function dibujarTablero(numFilas, numColumnas){
    let tablero = document.querySelector("#tablero");


document.querySelector("html").style.setProperty("--num-filas",numFilas)
;

document.querySelector("html").style.setProperty("--num-columnas",numCol
umnas);

    for(let f=0; f<numFilas; f++){
        for(let c=0; c<numColumnas; c++){
            let newDiv = document.createElement("div");
            tablero.appendChild(newDiv);
        }
    }
}
```

con eso, colocamos en las variables CSS lo que recibimos como parámetros de entrada de la función.

Si recargamos el navegador y en la consola ejecutamos dibujarTablero(5,6) nos saldrá el tablero que le indiquemos



Si le dais con el botón derecho encima del tablero y le dais a “inspeccionar” os mostrará el HTML que ha generado dinámicamente y todos los divs.



Vemos que todos los divs son iguales, y no sabemos en qué fila y columna están. Para ello vamos a hacer uso de los atributos de datos personalizados. Les daremos un id propio a cada uno, y les vincularemos un escuchador de evento.

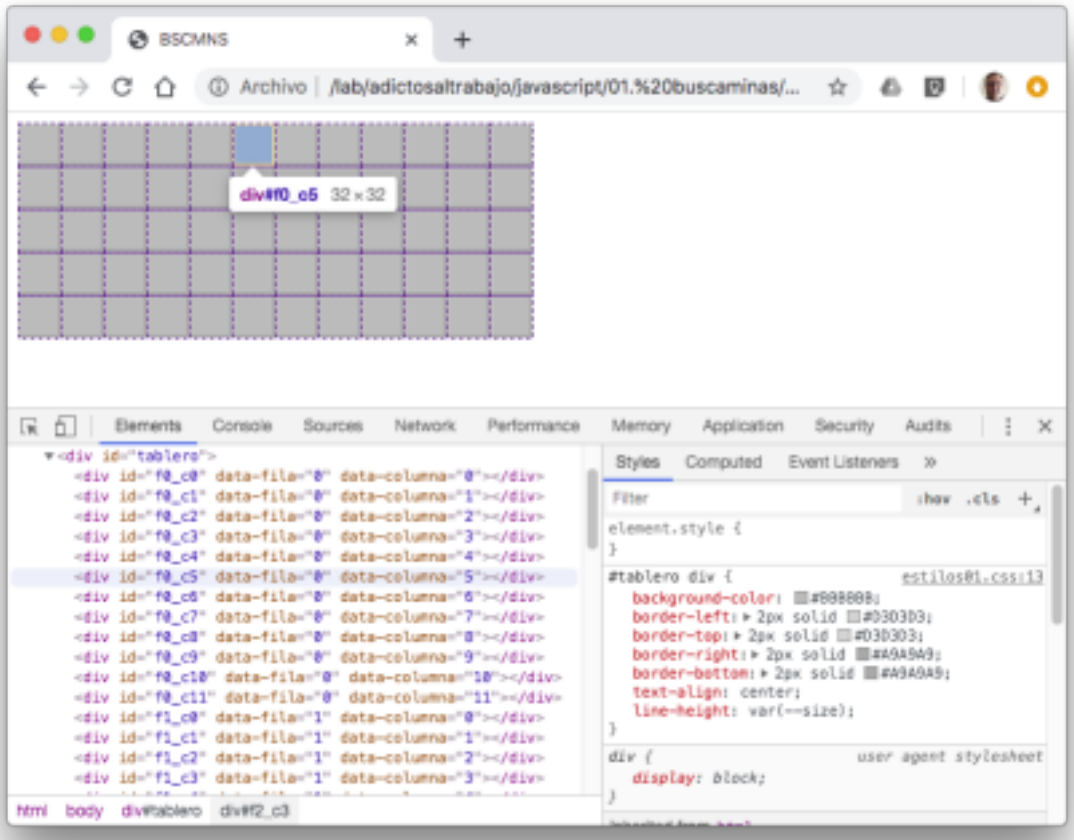
```
function dibujarTablero(numFilas, numColumnas){
 ...

    for(let f=0; f<numFilas; f++){
        for(let c=0; c<numColumnas; c++){
            let newDiv = document.createElement("div");
            newDiv.setAttribute("id","f" + f + "_c" + c );
            newDiv.dataset.fila = f;
            newDiv.dataset.columna = c;
            newDiv.addEventListener("contextmenu",marcar); //evento con
el botón derecho del raton
            newDiv.addEventListener("click",descubrir); //evento con
el botón izquierdo del raton

            tablero.appendChild(newDiv);
        }
    }
}
```

SI refrescamos la pantalla y pintamos el tablero de nuevo nos dará un error, porque los escuchadores de eventos estamos intentando vincularlos a funciones que aún no hemos definido: marcar y descubrir.

```
> pintarTablero(5,12)
▸Uncaught ReferenceError: marcar is not defined                funciones01.js:13
    at pintarTablero (funciones01.js:13)
    at <anonymous>:1:1
```

```
function marcar(evento){

}

function descubrir(evento){

}
```

Ya podemos recargar y comprobar el html generado.



Vemos que se ha generado un id concatenando fila, guión bajo, y columna. Además se han creado unos atributos llamados data-fila y data-columna. A estos atributos llamados data- se les llama atributos personalizados y son un estándar de HTML.

Otra cosa que debemos darnos cuenta es que si llamamos de nuevo a pintar añadirá divs a los que ya había. De alguna forma debemos vaciar el tablero cada vez que lo repintemos.

```
//borramos tablero actual
while (tablero.firstChild) {
        tablero.removeChild(tablero.firstChild);
}
```

NOTA: para los nóveles la solución anterior es suficiente, pero los más avezados se habrán dado cuenta que eso puede dar lugar a “pérdida de memoria”, llamado por los programadores *memory leak*.

Mientras el elemento tablero tenga hijo, lo borro. Esto lo saca del DOM, es decir lo quita de la pantalla, pero sigue ocupando espacio en la memoria del navegador. Porque antes hemos vinculado un escuchador de evento sobre ese objeto. A eso se le llama “bindar” un evento sobre ese objeto. Eso quiere decir que hay una referencia a ese objeto y por lo tanto no lo limpiará de la memoria.

Esta es una de las principales razones por las que las páginas web se degradan cuando llevan mucho tiempo funcionando. Y es que si no se hila fino, te vas dejando referencias a objetos. Ahora sí.

```
//borramos tablero actual
while (tablero.firstChild) {
        tablero.firstChild.removeEventListener("contextmenu",marcar);
        tablero.firstChild.removeEventListener("click",descubrir);
        tablero.removeChild(tablero.firstChild);
}
```

4. Generar un campo de minas vacío

Bien, un buscaminas tiene lo que se ve, el tablero, pero tiene una estructura interna que no se ve: que es dónde están las minas. Es una estructura en memoria, donde para cada casilla sabe algunas cosas:

- si hay una bomba o no en esa casilla
- el número de bombas que hay alrededor de esa casilla.

Para esta estructura en memoria vamos a una una matriz (es un vector de vectores, para los que aún no hayáis llegado a ese curso).

Podemos pasarle a todas las funciones los parámetros del número de filas y el número de columnas que tiene nuestro buscaminas, pero la verdad es que son propiedades del buscaminas, igual que el número de minas. Así que lo mejor es que nos creemos una variable (realmente es una constante) que almacene este objeto.

```
const buscaminas = {
    minasTotales: 30,
    numMinasEncontradas: 0,
    numFilas: 15,
    numColumnas: 15,
    aCampoMinas: []
}
```

Y podemos cambiar la función dibujarTablero() para que no reciba estos parámetros si no que los recoja del objeto buscaminas.

```
function dibujarTablero(){
 ...
    for(let f=0; f<buscaminas.numFilas; f++){
        for(let c=0; c<buscaminas.numColumnas; c++){
    ...
            tablero.appendChild(newDiv);
        }
    }
}
```

Ahora estamos en condición de generar el campo de minas vacío.

```
function generarCampoMinasVacio(){
    //generamos el campo de minas en el objeto buscaminas
    buscaminas.aCampoMinas = new Array(buscaminas.numFilas);
    for (let fila=0; fila<buscaminas.numFilas; fila++){
        buscaminas.aCampoMinas[fila] = new
Array(buscaminas.numColumnas);
    }
}
```

Con esto nos quedaría poner de una forma aleatoria las minas en el campo de minas y para cada casilla contar cuántas minas tiene alrededor.

5. Esparcir las minas

En el objeto buscaminas tenemos el número de minas totales. Así que mientras que el número de miras esparcidas sea menor que el número de minas totales, debemos elegir una fila aleatoria y una columna aleatoria y si no hay mina ya, colocar una, y sumar uno a las minas esparcidas.

```
function esparcirMinas(){
    //repartimos de forma aleatoria las minas
    let numMinasEsparcidas = 0;

        while (numMinasEsparcidas<buscaminas.minasTotales){
        //numero aleatorio en el intervalo [0,numFilas-1]
            let fila = Math.floor(Math.random() * buscaminas.numFilas);

        //numero aleatorio en el intervalo [0,numColumnas-1]
        let columna = Math.floor(Math.random() *
buscaminas.numColumnas);

        //si no hay bomba en esa posicion
        if (buscaminas.aCampoMinas[fila][columna] != "B"){
            //la ponemos
```

```
            buscaminas.aCampoMinas[fila][columna] = "B";

            //y sumamos 1 a las bombas esparcidas
            numMinasEsparcidas++;
        }
    }
}
```

6. Contar minas

Ahora dada una casilla en una fila y columna, tenemos que contar las minas que hay alrededor. Hay que contar, de la fila en la que estamos, la fila anterior (fila-1) y la fila en la que estoy fila, y (fila+1), y lo mismo para las columnas. En general hay que contar las 9 casillas, menos la central, que ya sabemos que no hay bomba (porque sólo debemos contar las bombas alrededor de las casillas donde no hay bomba)

```
function contarMinasCasilla(fila,columna){
    let numeroMinasAlrededor = 0;

    //de la fila anterior a la posterior
    for (let zFila = fila-1; zFila <= fila+1; zFila++){
        //de la columna anterior a la posterior
        for (let zColumna = columna-1; zColumna <= columna+1;
zColumna++){

            //si la casilla cae dentro del tablero
            if (zFila>-1 && zFila<buscaminas.numFilas && zColumna>-1 &&
zColumna<buscaminas.numColumnas){

                //miramos si en esa posición hay bomba
                if (buscaminas.aCampoMinas[zFila][zColumna]=="B"){

                    //y sumamos 1 al numero de minas que hay alrededor
de esa casilla
                    numeroMinasAlrededor++;
                }
            }
        }
    }

    //y guardamos cuantas minas hay en esa posicion
    buscaminas.aCampoMinas[fila][columna] = numeroMinasAlrededor;
}
```

Esto tenemos que hacerlo para cada casilla del tablero en la que no haya una bomba ya.

```
function contarMinas(){
    //contamos cuantas minas hay alrededor de cada casilla
    for (let fila=0; fila<buscaminas.numFilas; fila++){
        for (let columna=0; columna<buscaminas.numColumnas; columna++){
            //solo contamos si es distinto de bomba
            if (buscaminas.aCampoMinas[fila][columna]!="B"){
                        contarMinasCasilla(fila,columna);
            }
        }
    }
}
```

Con esto ya estaríamos en condición de jugar. Se podría hacer una función inicio() que fuese llamando a estas funciones para preparar el tablero, generar el campo de minas vacío, esparcir las minas, y contar cuántas hay alrededor de cada casilla.

```
function inicio(){
    buscaminas.numFilas = 10;
    buscaminas.numColumnas = 10;
    buscaminas.minasTotales = 12;
    dibujarTablero();
    generarCampoMinasVacio();
    esparcirMinas();
    contarMinas();
}
```

y al final del todo, le indicamos que cuando se cargue la ventana que llame a la función inicio(), para que se cargue todo.

```
window.onload = inicio;
```

Si recargamos la página, el tablero se pintará todo. Ahora sólo nos falta dotar de acciones a los eventos de ratón.

7. Acción con el botón derecho del ratón, Marcar

Ya vimos que cuando hacemos click con el botón derecho del ratón llamamos a una función llamada marcar que recibe el evento que acabamos de disparar como parámetro.

```
function marcar(miEvento){
    if (miEvento.type === "contextmenu"){
        console.log(miEvento);
        miEvento.stopPropagation();
        miEvento.preventDefault();
```

```
    }
}
```

Vemos si el evento es del tipo esperado, y si lo es, que nos lo muestre por la consola.

Además le decimos que detenga el burbujeo de eventos hacia arriba y que no nos muestre la acción por defecto (mostrar el menú contextual del navegador)

Ahora tenemos que marcar cíclicamente la casilla a “bandera” para indicar que ahí hay una mina, a “duda”, para indicar que puede que haya una mina, y finalmente quitar marca para volver al estado inicial

```
function marcar(miEvento){
    if (miEvento.type === "contextmenu"){
        console.log(miEvento);

        //obtenemos el elemento que ha disparado el evento
        let casilla = miEvento.currentTarget;

        //detenemos el burbujeo del evento y su accion por defecto
        miEvento.stopPropagation();
        miEvento.preventDefault();

        //obtenemos la fila de las propiedades dataset.
        let fila = casilla.dataset.fila;
        let columna = casilla.dataset.columna;

        if (fila>=0 && columna>=0 && fila< buscaminas.numFilas &&
columna < buscaminas.numColumnas) {
            //si esta marcada como "bandera"
            if (casilla.classList.contains("icon-bandera")){
                //la quitamos
                casilla.classList.remove("icon-bandera");
                //y la marcamos como duda
                casilla.classList.add("icon-duda");
                //y al numero de minas encontradas le restamos 1
                buscaminas.numMinasEncontradas--;
            } else if (casilla.classList.contains("icon-duda")){
                //si estaba marcada como duda lo quitamos
                casilla.classList.remove("icon-duda");
            } else if (casilla.classList.length == 0){
                //si no está marcada la marcamos como "bandera"
                casilla.classList.add("icon-bandera");
                //y sumamos 1 al numero de minas encontradas
                buscaminas.numMinasEncontradas++;
                //si es igual al numero de minas totales resolvemos el
tablero para ver si esta bien
                if (buscaminas.numMinasEncontradas ==
buscaminas.minasTotales){
                    buscaminas.resolverTablero(true);
```

```
                }
            }
        }
    }
}
```

Nos fijamos que el marcaje lo indicamos añadiendo o quitando una clase a la lista de clases. Esa lista, son los estilos que puede tener un elemento HTML. Nos

vamos a nuestra hoja de estilos y añadimos algunos que vamos a usar

Por un lado, le indicamos los iconos vectoriales, que son un tipo de letra. Por otro, indicamos cada icono que se corresponde con una “letra” de esa tipografía.

```
@font-face {
  font-family: 'fontello';
  src: url('./fontello.eot');
  src: url('./fontello.woff2') format('woff2'),
       url('./fontello.woff') format('woff'),
       url('./fontello.ttf') format('truetype'),
       url('./fontello.svg') format('svg');
  font-weight: normal;
  font-style: normal;
}

 [class^="icon-"]:before, [class*=" icon-"]:before {
  font-family: "fontello";
  font-style: normal;
  font-weight: normal;
  speak: none;
  display: inline-block;
  text-decoration: inherit;
  width: 1em;
  margin-right: .2em;
  text-align: center;
  font-variant: normal;
  text-transform: none;
  line-height: 1em;
  margin-left: .2em;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-shadow: 2px 2px 2px #999999;

}

.icon-bandera:before { content: '\e800'; color: #B22222;}
.icon-duda:before { content: '\e801'; }
.icon-bomba:before { content: '\f1e2'; }
```

8. Acción con el botón principal del ratón. descubrir

Cuando hacemos click con el botón principal, estamos “destapando” la casilla. Hemos vinculado en evento de hacer click con una función llamada descubrir que recibe un evento.

Vamos a trabajarla.

```
function descubrir(miEvento){
    if (miEvento.type === "click"){
        let casilla = miEvento.currentTarget;
        let fila = casilla.dataset.fila;
        let columna = casilla.dataset.columna;

        descubrirCasilla(fila,columna);
    }
}
```

Lo primero que hacemos es ver si el evento que recibe la función es del tipo que esperamos. Y obtenemos la casilla en base al “target” que ha disparado el evento.

Una vez sabemos la casilla podemos consultar su fila y columna en base a los dataset almacenados.

Cuando sabemos esa información, llamamos a una función que destapa la casilla y que recibe como parámetros la fila y columna a descubrir. Le hemos llamado descubrirCasilla(fila, columna)

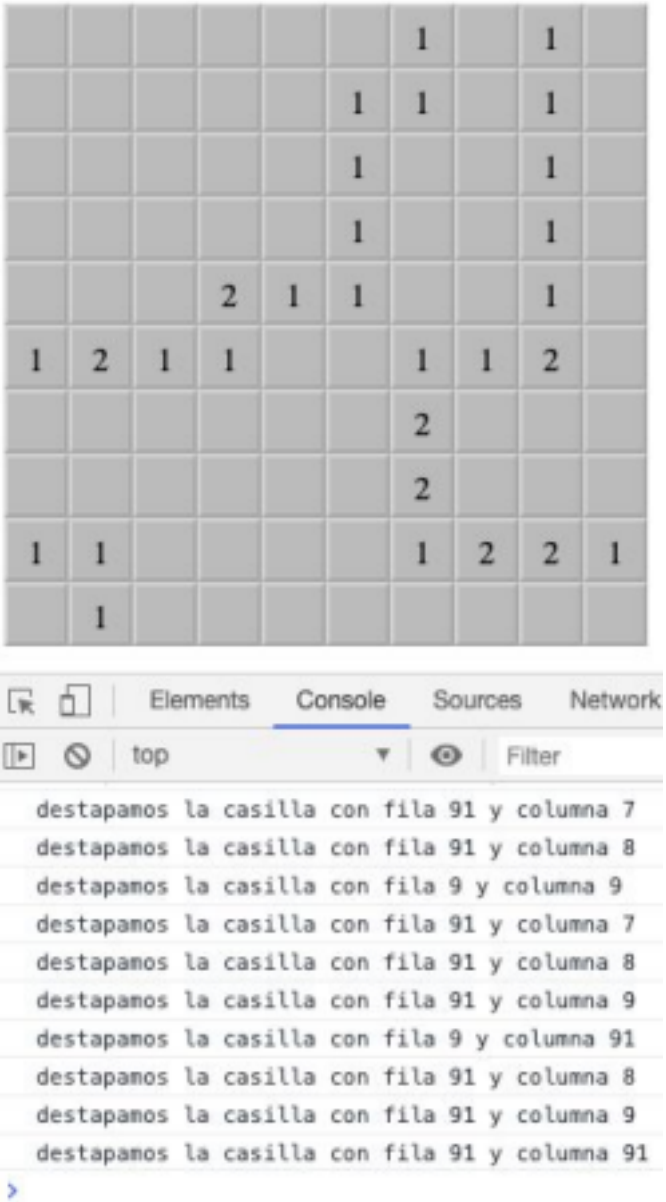
```
function descubrirCasilla(fila, columna){
    console.log("destapamos la casilla con fila " + fila + " y columna "
+columna );

    //si la casilla esta dentro del tablero
    if (fila > -1 && fila < buscaminas.numFilas &&
        columna >-1 && columna < buscaminas.numColumnas){

        //obtenermos la casilla con la fila y columna
        let casilla = document.querySelector("#f" + fila + "_c" +
columna);

        //si la casilla no esta destapada
        if (!casilla.classList.contains("destapado")){

            //si no esta marcada como "bandera"
            if (!casilla.classList.contains("icon-bandera")){

                //la destapamos
                casilla.classList.add("destapado");
```

```
                //ponemos en la casilla el número de minas que tiene
alrededor
                casilla.innerHTML =
buscaminas.aCampoMinas[fila][columna];

                //ponemos el estilo del numero de minas que tiene
alrededor (cada uno es de un color)
                casilla.classList.add("c" +
buscaminas.aCampoMinas[fila][columna])

                //si no es bomba
                if (buscaminas.aCampoMinas[fila][columna] !=="B"){

                    // y tiene 0 minas alrededor, destapamos las
casillas contiguas
                    if (buscaminas.aCampoMinas[fila][columna] == 0){
                       descubrirCasilla(fila-1,columna-1);
                       descubrirCasilla(fila-1,columna);
                       descubrirCasilla(fila-1,columna+1);
                       descubrirCasilla(fila,columna-1);
                       descubrirCasilla(fila,columna+1);
                       descubrirCasilla(fila+1,columna-1);
                       descubrirCasilla(fila+1,columna);
                       descubrirCasilla(fila+1,columna+1);

                        //y borramos el 0 poniendo la cadena vacía
                         casilla.innerHTML = "";
                    }
                                        }else if
                  (buscaminas.aCampoMinas[fila][columna]=="B"){ // si
                       por el contrario hay bomba quitamos la B
                    casilla.innerHTML = "";

                    //añadimos el estilo de que hay bomba
                    casilla.classList.add("icon-bomba");

                    // y que se nos ha olvidado marcarla
                    casilla.classList.add("sinmarcar");

                    // y resolvemos el tablero indicando (false), que
hemos cometido un fallo
                    resolverTablero(false);
                }
            }
        }
    }
}
```

Ponemos una traza para que nos pinte por consola qué casilla estamos destapando. Esto es útil para encontrar errores.

Lo siguiente que miramos es si la fila y columna están dentro de los límites del tablero, no sea que nos pasen una casilla que está fuera.

Recordemos que ahora no tenemos la casilla en sí, si no la fila y la columna. Como nos interesa tener información de la casilla la obtenemos en base a su fila y columna.



Miramos si la casilla no está destapada. Consideramos que está destapada cuando tiene la clase css “destapada”, que aún no hemos definido.

Luego miramos si la casilla está marcada como bandera de la misma manera, porque si hemos puesto una bandera en esa casilla, no se puede descubrir.

Si se dan todas estas circunstancias entonces sí podemos descubrir la casilla, y lo que hacemos es ponerla como destapada. Es decir, le añadimos el estilo de la clase CSS “destapada”, y en su interior ponemos el número de minas que tiene alrededor.

Ahora, puede ser una bomba o puede que no.

Si no es una bomba, y no tiene minas alrededor, podemos descubrir las casillas adyacentes. Y como tiene cero minas alrededor, reemplazamos el número cero por una cadena vacía para que no pinte un cero en el tablero.

Ahora si es una mina, hemos perdido. Borramos el número de minas que tiene alrededor, le ponemos el estilo de la clase icon-bomba para indicar que ahí hay una bomba, y añadimos un estilo de “sinmarcar”, para indicar que esa bomba se nos ha olvidado marcarla y salga resaltada. Y llamamos a la función resolverTablero(false). El false indica que hemos perdido.

Si recargamos la página y hacemos click podemos ver cosas curiosas, como que aún no hemos definido la clase “descubrir”, ni los colores. Pero además vemos en la consola cosas muy raras. Como “destapamos la casilla con fila 91 y columna 7”.

Si las filas van de 0 a 9 y las columna de 0 a 9 ¿como puede haber fila 91?

He dejado este error adrede por dos motivos.

El principal es porque me ha ocurrido de verdad, y si lo oculto, estoy haciendo el artículo más bonito, pero estoy ocultando una lección. El otro motivo es la razón por la que JavaScript ha sido tan denostado durante tanto tiempo, y es que el tipado es dinámico ¿y eso qué es?

Muy fácil, las filas y columnas son una variable numérica y cuando uso el signo + entre dos variables numéricas es otra numérica, con la suma de las anteriores.

Sin embargo, cuando recuperamos el valor de la fila y la columna así:

```
let fila = casilla.dataset.fila;
let columna = casilla.dataset.columna;
```

Estamos recuperando propiedades de un objeto HTML (de un HTMLElement). Estas propiedades son "strings", es decir, cadenas de texto, y el operador + entre dos cadenas de texto da lugar a la concatenación. Así que cuando estamos en la fila 9, y columna 6, cuando llamamos a `descubrirCasilla(fila+1,columna)`, estamos llamado realmente a `descubrirCasilla("9"+1,"6")`, que es `descubrirCasilla("91","6")`

Estas son esas cosas que sacan de quicio a programadores de otros lenguajes donde una variable no puede cambiar nunca de tipo.

Se soluciona fácilmente convirtiéndola a un número entero en base 10.

```
let fila = parseInt(casilla.dataset.fila,10);
let columna = parseInt(casilla.dataset.columna,10)
```

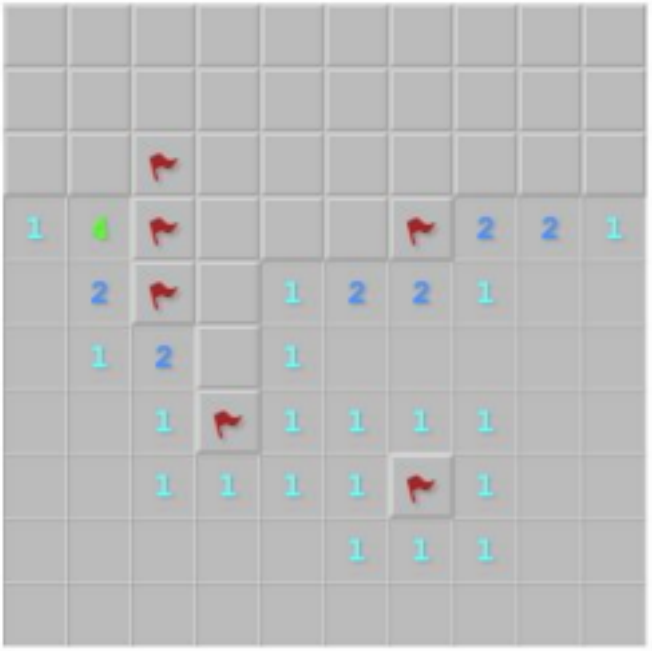
Bien, ya funciona, pero hemos marcado las casillas con estilos que aún no hemos definido, como "destapada", "sinmarcar", etc...

Añadimos las clases que definen esos estilos:

```
#tablero div.destapado{
	font-family: "Lucida Console","Courier New", Courier, monospace;
	font-weight: bold;
	border-left: 1px solid #A9A9A9;
	border-top: 1px solid #A9A9A9;
	border-right: 1px solid #D3D3D3;
	border-bottom: 1px solid #D3D3D3;
	text-shadow: 1px 1px 2px #999999;
}
#tablero div.sinmarcar{
	background-color: #FF0000;
}
#tablero div.banderaErronea{
	background-color: #AAAAAA;
	border: 1px solid #B22222;

}

//colores con número de minas alrededor
```

```
#tablero div.c1{ color: #00ffff;}
#tablero div.c2{ color: #0099ff;}
#tablero div.c3{ color: #0033ff;}
#tablero div.c4{ color: #00ff33;}
#tablero div.c5{ color: #ccff00;}
#tablero div.c6{ color: #ffcc00;}
#tablero div.c7{ color: #ff3300;}
#tablero div.c8{ color: #660000;}
```

Recargamos la página y jugamos un poco a ver qué pinta tiene.



Ya somos capaces de “descubrir”, de “marcar minas” y de saber cuántas hay alrededor de cada casilla destapada.
Nos quedaría saber cuántas minas nos quedan e implementa la función resolverTablero() cuando acaba la partida.

9. Contador de número de minas.

Vamos a añadir una barra de estado que nos diga el número de minas restante en cada momento. Para eso en el fichero html añadimos un contenedor bajo el tablero con el id “estado”

```
<body>
	<div id="tablero"></div>
	<div id="estado"><div>Nº de minas restante: <span
id="numMinasRestantes"></sapn></div></div>
</body>
```

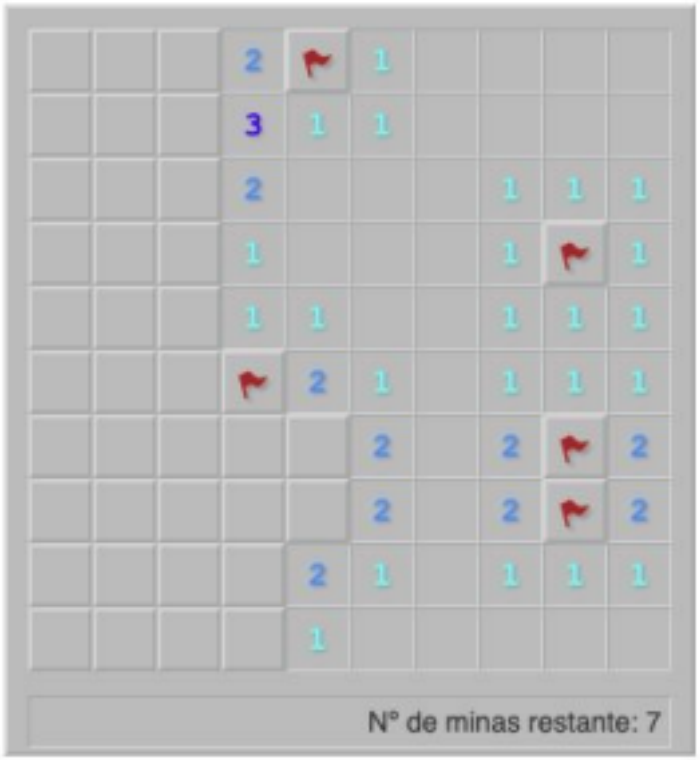
Cambiamos el CSS para darle un aspecto de continuación del tablero:

```
#estado{
	padding: 2px 10px 2px 10px;
```

```
        display:block;
        background-color: #BBBBBB;
        border-left: 2px solid #D3D3D3;
        /* border-top: 2px solid #D3D3D3; */
        border-right: 2px solid #A9A9A9;
        border-bottom: 2px solid #A9A9A9;
        width: calc(var(--num-columnas)*var(--size));
}
#estado div{
        text-align:right;
        font-family: sans-serif;
        font-size: 14px;
        color: #333333;
        border-left: 1px solid #A9A9A9;
        border-top: 1px solid #A9A9A9;
        border-right: 1px solid #D3D3D3;
        border-bottom: 1px solid #D3D3D3;
        padding: 4px;
}
```

Fijaron que al DIV del estado le quitamos el borde superior, para dar sensación de continuidad con el tablero.

Por otro lado el ancho de este contenedor debería ser igual al del tablero. Podemos hacerlo por JavaScript y redimensionar el ancho al que tenga el tablero, pero también
podemos hacerlo por CSS. Ya que el ancho sólo depende del número de columnas y del ancho de cada columna, así que lo calculamos en el propio CSS en función de esas variables.



```
width: calc(var(--num-columnas)*var(--size));
```

Ahora lo que debemos es actualizar el contenido dei id con “numMinasRestantes” con el número de minas que quedan en cada caso.

```
function actualizarNumMinasRestantes(){
    document.querySelector("#numMinasRestantes").innerHTML =
        (buscaminas.minasTotales - buscaminas.numMinasEncontradas); }
```

Y nos queda llamarlo al iniciar() el juego y al final de la función marcar()

```
function inicio(){
    [...]
    actualizarNumMinasRestantes();
}
function marcar(miEvento){
    if (miEvento.type === "contextmenu"){
        [...]
        if (fila>=0 && columna>=0 && fila< buscaminas.numFilas &&
columna < buscaminas.numColumnas) {
            [...]
            //actualizamos la barra de estado con el numero de minas
restantes
            actualizarNumMinasRestantes();
        }
    }
}
```

10. Resolver el tablero.

¿Cómo acaba la partida?

- o bien descubrimos dónde están todas las minas
- o bien destapamos una mina

En el primer caso ganamos, y en el segundo perdemos.

Pero reflexionemos un momento sobre descubrir una mina.

- puede ser por mala suerte
- puede ser fruto de una equivocación. Hemos marcado erróneamente una casilla como mina, que en realidad no lo es, y como consecuencia, acabamos descubriendo una casilla que sí tiene una mina que no nos esperábamos. En ese caso también queremos saber dónde hemos cometido el error.

Vamos a descubrir las casillas que quedan sin descubrir para ver lo que hay debajo. Para ello obtenemos todos los elementos hijos de tablero y los guardamos en una variable para recorrerla posteriormente.

```
let aCasillas = tablero.children;
```

Y ahora para cada casilla le quitamos los eventos. ¿por qué? Esto no se lo expliqué a mi hija, pero sabemos que si luego jugamos otra partida y borramos el tablero, aunque lo eliminemos del DOM, como las casillas tienen escuchadores de eventos, hay una referencia a ellas, y el recolector de basura no las limpia de la memoria. Si no has entendido este párrafo, no te preocupes, y sigue leyendo.

```
for (let i = 0 ; i < aCasillas.length; i++){
        //quitamos los listeners de los eventos a las casillas
       aCasillas[i].removeEventListener("click", descubrir);
        aCasillas[i].removeEventListener("contextmenu", marcar);

        [...]
}
```

Como ahora estamos recorriendo todas las casillas, no sabemos exactamente a qué fila y columna pertenece la casilla en cuestión que estamos leyendo. Pero lo podemos obtener fácilmente.

```
let fila = parseInt(aCasillas[i].dataset.fila,10);
let columna = parseInt(aCasillas[i].dataset.columna,10);
```

Ahora se trata de descubrir las casillas
Si la casilla es una bandera y debajo tiene una bomba, eso es que la bandera es correcta. Debemos descubrir, y cambiar la bandera por la mina.
Ahora bien, si en la casilla hay una bandera y debajo no hay una mina, se trata de una bandera errónea, e indicamos isOK a false.
Y así para todas las casillas.

Si al finalizar de descubrir todas las banderas estaban bien, hemos ganado.

```
function resolverTablero(isOK){
    let aCasillas = tablero.children;
    for (let i = 0 ; i < aCasillas.length; i++){
        //quitamos los listeners de los eventos a las casillas
       aCasillas[i].removeEventListener("click", descubrir);
        aCasillas[i].removeEventListener("contextmenu", marcar);

        let fila = parseInt(aCasillas[i].dataset.fila,10);
        let columna = parseInt(aCasillas[i].dataset.columna,10);

        if (aCasillas[i].classList.contains("icon-bandera")){
            if (buscaminas.aCampoMinas[fila][columna] == "B"){
                //bandera correcta
                aCasillas[i].classList.add("destapado");
                aCasillas[i].classList.remove("icon-bandera");
```

```
                aCasillas[i].classList.add("icon-bomba");
            } else {
                //bandera erronea
                aCasillas[i].classList.add("destapado");
                aCasillas[i].classList.add("banderaErronea");
                isOK = false;
            }
        } else if (!aCasillas[i].classList.contains("destapado")){
            if (buscaminas.aCampoMinas[fila][columna] == "B"){
                //destapamos el resto de las bombas
                aCasillas[i].classList.add("destapado");
                aCasillas[i].classList.add("icon-bomba");
            }
        }

    }

    if (isOK){
        alert("¡¡¡Enhorabuena!!!");
    }
}
```

11. Conclusiones

Lo ideal es que el chaval se ponga ahora a cambiar cosas en el código y vea qué pasa. ¿y se aumentamos el número de filas y columnas? ¿y las bombas?

A los que más sabéis os sorprenderá este código porque contraviene un montón de buenas prácticas. No costaría mucho convertir las funciones en métodos del objeto buscaminas. Pero pensad que se trataba de mantener la atención de una niña de nueve años mientras se programaba y se veían los resultados por pantalla.

Ha sido un ejercicio muy gratificante explicar a mi hija cómo se hace un buscaminas. No nos damos cuenta y solemos explicar a un nivel muy profundo, pues nuestros interlocutores suelen ser profesionales como nosotros. No estamos acostumbrados a explicar de cero conceptos sencillos como estos.

Estoy seguro que no lo ha entendido todo. Pero también sé que se lo pasó bien, y entendió los conceptos.

De vez en cuando viene y me dice:
— papá ¿puedo jugar una partida a mi buscaminas?