

BINGO

Vamos a comenzar el desarrollo del juego del Bingo versión española. En este sprint, nos centraremos en el desarrollo de la maquetación inicial de los cartones.

1. Los Cartones del juego están compuestos de 3 líneas con 9 celdas por línea, en total 27 celdas.
2. Vamos a crear tres clases:
 - a. Carton
 - b. Linea
 - c. Celda
3. Utilizaremos un layout HTML para cada objeto construido con las clases, serán constantes en el desarrollo:

```
const layoutHTMLCarton = {
  tipo: 'section',
  atributos: {
    name: "carton",
  },
  estilos: {
    backgroundColor: 'lightgrey',
    width: '600px',
    height: '240px',
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
    justifyContent: 'center'
  }
};

const layoutHTMLLinea = {
  tipo: 'div',
  atributos: {
    name: "linea"
  },
  estilos: {
    width: '600px',
    height: '70px',
    backgroundColor: 'dodgerblue',
    border: '1px solid white',
    display: 'flex',
    flexDirection: 'row',
    justifyContent: 'space-around',
    alignItems: 'center'
  }
};

const layoutHTMLCelda = {
  tipo: 'div',
  atributos: {
    name: 'celda'
  },
  estilos: {
    height: '50px',
    width: '50px',
    backgroundColor: 'whitesmoke',
    border: '1px solid snow',
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
    fontSize: '2em',
    fontFamily: 'Arial',
    fontWeight: 'bold'
  }
};
```

Este desarrollo consta de 2 SPRINTS, el primer SPRINT vale 4 puntos y el segundo SPRINT vale 6 puntos.

Utilizaremos el siguiente html, con el src = "bingo.js"

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>BINGO</title>
</head>
<body>
</body>
<script src="bingo.js"></script>
</html>
```

SPRINT 1. – Maquetación del cartón.	4 puntos
Función generarElementoHTML	2 puntos
Esta función, similar al método estático que utilizamos en el blog, tomará un layout y devolverá un elemento HTML construido con la especificación dada: <ol style="list-style-type: none">1. tipo, Recoge el tipo de elemento a crear2. atributos, es un objeto con todos los atributos del elemento3. estilos, es un objeto con todos los estilos a aplicar al elemento	

CLASES Carton/Linea/Celda		2 puntos
Para abreviar, en este desarrollo no utilizaremos setters o getters		
CLASE Carton		
Propiedades		
Constructor (parámetros entrada que pasan a propiedades)	numeroBolasJuego	Parámetro, por defecto 90 (Bingo español)
	numerosPorCarton	Parámetro, por defecto 15
	numeroDeLineas	Parámetro, por defecto 3
	numeroDeColumnas	Parámetro, por defecto 9
Constructor (otras propiedades)	ElementoHTML	Llamará a la función generarElementoHTML con el layout correspondiente al cartón y devolverá el elemento que quedará asignado a la propiedad.
	numeros	Array vacío que recogerá los números del cartón
	lineas	Array vacío que recogerá un array con cada línea que generemos para el cartón
Constructor - Proceso		
<p>El constructor de la clase Carton, generará todas las líneas requeridas en el parámetro de entrada.</p> <p>Se llamará al método generarLineas() descrito a continuación, para generar todas las líneas del cartón.</p> <p>Desplegaremos el layout del cartón en el document.body, en la posición 'beforeend'</p>		
Método	generarLineas()	<p>En un bucle, generará para cada línea un objeto de la clase Linea, al que le pasará 3 parámetros:</p> <ul style="list-style-type: none"> El propio cartón El primer número de la línea, se calculará en función del índice del bucle, por ejemplo para la primera línea, en el caso de que tenga 9 columnas, irá del 1 al 9, la segunda del 10 al 18, y la tercera del 19 al 27 El último número de la línea, tal como se explica en el punto anterior. <p>Una vez generada el objeto línea, este se añadirá al array líneas. De esta forma tendremos referenciadas todas las líneas del cartón.</p>

CLASE Linea		
Propiedades		
Constructor (parámetros entrada que pasan a propiedades)	carton	Parámetro, recoge el objeto cartón
	desde	Número inicial de la línea
	hasta	Número final de la línea
Constructor (otras propiedades)	ElementoHTML	Llamará a la función generarElementoHTML con el layout correspondiente a la línea y devolverá el elemento que quedará asignado a la propiedad.
	numeros	Array vacío que recogerá los números de la línea
	celdas	Array vacío que recogerá un array con cada celda que generaremos para la línea

Constructor - Proceso		
Se invocará al método generarCeldas() descrito a continuación para generar todas las celdas. Desplegaremos el layout de la línea en el elementoHTML del cartón en la posición 'beforeend'.		
Método	generarCeldas()	<p>En un bucle se generarán todas las celdas correspondientes de la línea, el bucle se iniciará en el valor desde y finalizará en el hasta.</p> <p>El bucle generará un objeto de la clase Celda, al que se le pasarán los siguientes parámetros:</p> <ul style="list-style-type: none">• El propio objeto línea• El numero a generar. <p>Una vez generado el objeto celda, este se añadirá al array de celdas de la línea. A su vez el número procesado se incorporará al array de números del cartón y al array de números de la línea.</p> <p>.</p>

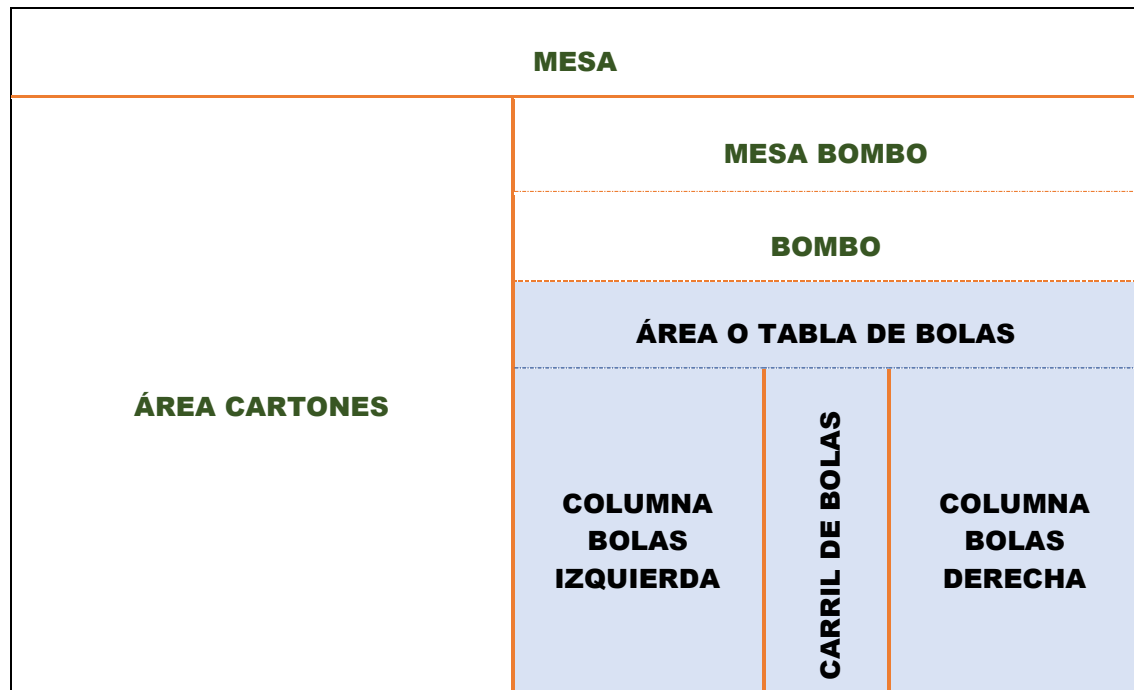
CLASE Celda		
Propiedades		
Constructor (parámetros entrada que pasan a propiedades)	linea	Parámetro, recoge el objeto linea
	numero	Numero de la celda
Constructor (otras propiedades)	ElementoHTML	Llamará a la función generarElementoHTML con el layout correspondiente a la celda y devolverá el elemento que quedará asignado a la propiedad.
Constructor - Proceso		
Al innerHTML del elemento HTML generado se le asignará el número Desplegaremos el layout de la celda en el de la línea, en la posición 'beforeend'.		
SALIDA PREVISTA		
Construyendo 3 cartones con las siguientes llamadas: <pre>let miCarton1 = new Carton(90,15,3,9); let miCarton2 = new Carton(90,15,3,9); let miCarton3 = new Carton();</pre>		

SPRINT 2. – Ajustando la numeración de los cartones						6 puntos																																																																																																				
En el primer sprint hemos realizado la maquetación del cartón, en este sprint nos centraremos en generar la adecuada numeración de este.																																																																																																										
Vamos a centrarnos en el caso del Bingo en España:																																																																																																										
<div>1. El bingo tiene 90 números</div> <div>2. Cada cartón tiene 3 filas y 9 columnas, en total 27 celdas</div> <div>3. Cada línea contiene 5 números y 4 celdas vacías que se distribuyen de forma aleatoria.</div> <div>4. En total el cartón, por tanto, dispone de 15 números y 12 celdas vacías.</div> <div>5. Cada columna sólo admite números correspondiente a una decena que le corresponde, es decir: la columna 1, números del 1 al 9, la 2 del 10 al 19, la 3 del 20 al 29, y así sucesivamente, hasta la última que debe incluir del 80 al 90 (en vez del 89).</div>																																																																																																										
EJEMPLOS (3 CARTONES GENERADOS)																																																																																																										
<table><tr><td></td><td>11</td><td>26</td><td>35</td><td></td><td></td><td>64</td><td></td><td>84</td></tr><tr><td></td><td></td><td></td><td>37</td><td>43</td><td>54</td><td>65</td><td>72</td><td></td></tr><tr><td>6</td><td>12</td><td></td><td></td><td>46</td><td></td><td>66</td><td></td><td>86</td></tr><tr><td colspan="9"></td></tr><tr><td></td><td>15</td><td></td><td></td><td>47</td><td>54</td><td>68</td><td>75</td><td></td></tr><tr><td></td><td>19</td><td></td><td></td><td></td><td>57</td><td>61</td><td>76</td><td>90</td></tr><tr><td></td><td>14</td><td>25</td><td></td><td>43</td><td></td><td>62</td><td>71</td><td></td></tr><tr><td colspan="9"></td></tr><tr><td>4</td><td>19</td><td>29</td><td>32</td><td></td><td></td><td>69</td><td></td><td></td></tr><tr><td>3</td><td>12</td><td>28</td><td>33</td><td>42</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td>36</td><td>46</td><td></td><td></td><td>73</td><td>82</td></tr></table>									11	26	35			64		84				37	43	54	65	72		6	12			46		66		86											15			47	54	68	75			19				57	61	76	90		14	25		43		62	71											4	19	29	32			69			3	12	28	33	42					1			36	46			73	82
	11	26	35			64		84																																																																																																		
			37	43	54	65	72																																																																																																			
6	12			46		66		86																																																																																																		
	15			47	54	68	75																																																																																																			
	19				57	61	76	90																																																																																																		
	14	25		43		62	71																																																																																																			
4	19	29	32			69																																																																																																				
3	12	28	33	42																																																																																																						
1			36	46			73	82																																																																																																		
Función numeroAleatorio						0,5																																																																																																				
Esta función tomará como parámetros, dos números (desde, hasta) entre los que deberá devolver un entero aleatorio cuyo valor podrán estar incluidos estos dos valores (desde, hasta) Así, la llamada a numeroAleatorio(10,19), devolverá un número entre 10 y 19, pudiendo incluir estos 2.																																																																																																										
Modificaciones CLASE Carton						0,25																																																																																																				
Propiedades																																																																																																										
Constructor, nuevas propiedades	numerosPorLinea	No lo calcularemos en esta versión, lo dejaremos establecido en el valor 5.																																																																																																								
	huecosPorLinea	Lo dejamos establecido en el valor 4.																																																																																																								
Método	generarLineas()	Aunque en esta versión no necesitaremos pasar los parámetros primero y último, los dejaremos. A la llamada, le añadimos huecosPorLinea y numeroDeColumnas para procesarlos en la propia línea																																																																																																								

Modificaciones CLASE Linea		4
Propiedades		
Constructor (nuevos parámetros entrada que pasan a propiedades)	numeroHuecos	Huecos que tendrá que recoger cada línea
	numeroDeColumnas	Número de celdas totales que tiene cada línea
Constructor (otras propiedades)	huecos	Array vacío que recogerá los números de la celdas que estarán vacía. (serán cuatro elementos con valores del 1 al 9)
Método	generarHuecos()	<p>Este método va a calcular las cuatro (numeroHuecos) celdas que estarán vacías en la línea.</p> <p>En un bucle para procesar todos los huecos (1 al numeroHuecos), buscaremos un valor aleatorio entre 1 y el número de columnas. Como no puede repetirse, lo buscaremos en el array huecos para comprobar que no exista, antes de añadirlo al array huecos.</p> <p>Una vez que el array contiene el índice de las celdas vacías, el método lo devolverá.</p>
	generarCeldas()	<p>Comentar el código del Sprint 1. Este método debe cambiarse completamente para poder adaptarlo a la especificación del Bingo.</p> <p>Lo primero será llamar al método generarHuecos, que nos devuelve la posición de los huecos que estarán vacíos en la línea.</p> <p>En un bucle, para cada columna (i), si el índice (i) de la columna está entre los huecos devueltos en la función anterior, quiere decir que esa celda estará vacía por lo que al array celdas del objeto, le añadiremos un objeto Celda al que le pasaremos el valor 0.</p> <p>En otro caso debemos encontrar un número aleatorio entre los valores de decena relativos al índice (i) de la columna.</p> <p>Índice = 1, del 1 al 9, Índice = 2, del 10 al 19. ... Índice = 8, del 70 al 79 Índice = 9, del 80 al 90</p> <p>Ese valor aleatorio no puede estar entre los valores del array numeros del cartón, por lo que buscaremos el valor adecuado hasta que no se repita.</p> <p>Una vez encontrado, lo incluiremos en el array numeros del cartón y a continuación generaremos un objeto Celda con el valor del número aleatorio procesado y lo incluiremos en el array celdas de la línea.</p>
Modificaciones CLASE Celda		0,25
Constructor - Proceso		
Modificaremos la asignación del innerHTML, si el número es 0, le asignaremos una cadena vacía "", en otro caso el número.		

SPRINT 3. – Tablero, Bombo y bolas

El primer paso es reorganizar el escenario. Partiremos de una hipotética mesa, cuyo diseño será más o menos el siguiente:



```
const layoutHTMLMesa = {
  tipo: "div",
  atributos: {
    mesa: "mesa",
  },
  estilos: {
    display: "flex",
  },
};

const layoutHTMLAreaCartones = {
  tipo: "div",
  atributos: {
    name: "areaCartones",
  },
  estilos: {
    display: "flex",
    flexDirection: "column",
  },
};

const layoutHTMLMesaBombo = {
  tipo: "div",
  atributos: {
    mesa: "mesaBombo",
  },
  estilos: {
    display: "flex",
    flexDirection: "column",
  },
};

const layoutHTMLAreaBolas = {
```

```
    tipo: "div",
    atributos: {
      name: "areaBolas",
    },
    estilos: {
      display: "flex",
      width: "600px",
      flexDirection: "row",
      flexWrap: "wrap",
      justifyContent: "center",
    },
  },
};

const layoutHTMLColumnaBolas = {
  tipo: "div",
  atributos: {
    name: "columnaBolas",
  },
  estilos: {
    display: "flex",
    width: "275px",
    flexDirection: "row",
    flexWrap: "wrap",
    justifyContent: "center",
  },
};

const layoutHTMLCarrilBolas = {
  tipo: "div",
  atributos: {
    name: "carrilBolas",
  },
  estilos: {
    display: "flex",
    width: "50px",
    flexDirection: "row",
    flexWrap: "wrap",
    justifyContent: "center",
  },
};

const layoutHTMLBombo = {
  tipo: "section",
  atributos: {
    name: "bombo",
  },
  estilos: {
    backgroundColor: "lightgrey",
    width: "600px",
    height: "240px",
    display: "flex",
    flexDirection: "column",
    alignItems: "center",
  },
};
```



```
    justifyContent: "center",
  },
};
```

Los layouts especificados como constantes, corresponden a cada una de las áreas del tablero de juego, cuya raíz será la MESA.

También se especifican los huecos de las bolas del tablero de bolas:

```
const layoutHTMLHuecoBola = {
  tipo: "div",
  atributos: {
    name: "huecoBola",
  },
  estilos: {
    display: "inline-block",
    background: "#cccccc",
    borderRadius: "100px",
    color: "#fff",
    width: "40px",
    height: "40px",

    lineHeight: "1.6em",
    marginRight: "15px",
    fontSize: "1.5em",
    fontFamily: "Arial",
    fontWeight: "bold",
    textAlign: "center",

    boxShadow: "7px 7px 5px 0px rgba(50, 50, 50, 0.75)",
  },
};
```

Y las propias bolas:

```
const layoutHTMLBola = {
  tipo: "div",
  atributos: {
    name: "bola",
  },
  estilos: {
    display: "inline-block",
    background: "#fd4d3f",
    borderRadius: "100px",
    color: "#fff",
    width: "40px",
    height: "40px",
    lineHeight: "1.6em",
    marginRight: "15px",
    fontSize: "1.5em",
    fontFamily: "Arial",
```

```

fontWeight: "bold",
textAlign: "center",
boxShadow: "7px 7px 5px 0px rgba(50, 50, 50, 0.75)",
bottom: "50px",
position: "relative",
transitionProperty: "bottom",
transitionDuration: "2s",
},
};

const layoutHTMLBotonBingo = {
  tipo: "button",
  atributos: {
    name: "botonBombo",
    fontSize: "4em",
  },
  eventos: []
}

```

Clase Bingo

Esta clase va a gestionar el juego del bingo, contiene las propiedades para realizar la distribución del tablero, los cartones, el bombo y las bolas y la gestión de los eventos relacionados con el juego.

Propiedades

Constructor, parámetros que pasan a propiedades	numeroDeBolas	Valor por defecto 90		
	numeroDeCartones	Valor por defecto 3		
	numerosPorCarton	Valor por defecto 15		
	lineas	Valor por defecto 3		
	columnas	Valor por defecto 9		
Constructor otras propiedades	cartones	Array de cartones		
	bolas	Array de bolas		
	numeros	Array de números		
propiedades layout	mesaHTML	bomboHTML	columnaBolasDerechaHTML	
	areaCartonesHTML	botonBingoHTML	carrilBolasHTML	
	mesaBomboHTML	areaBolasHTML	columnaBolasIzquierdaHTML	

Constructor - Proceso

- ❖ GENERAR LA MESA
 - Con la función generarElementoHTML y el layout de la mesa generamos la propiedad del objeto bingo mesaHTML
 - Añadimos al document.body el elemento mesaHTML generado anteriormente
- ❖ GENERAR AREA DE CARTONES
 - Con la función generarElementoHTML y el layout del área de Cartones generamos la propiedad del objeto bingo areaCartonesHTML
 - Añadimos al elemento mesaHTML el layout generado anteriormente

- Invocamos al método generarCartones() que generará los cartones con las clase y método utilizados en los anteriores sprints.
- ❖ **GENERAR MESA DEL BOMBO**
 - Con la función generarElementoHTML y el layout de la mesa del Bombo generamos la propiedad del objeto bingo mesaBomboHTML
 - Añadimos al elemento mesaHTML el layout generado anteriormente
 - Invocamos al método generarCartones() que generará los cartones con las clase y método utilizados en los anteriores sprints.
- ❖ **BOMBO**


La mesa del Bombo contiene el propio Bombo y el tablero donde se colocarán las bolas después de deslizarse por el carril.

El Bombo consta de una imagen del Bombo y un botón que accionará o detendrá el giro de este.

- Con la función generarElementoHTML y el layout del Bombo generamos la propiedad del objeto bingo bomboHTML
- Añadimos al elemento mesaBomboHTML el objeto bomboHTML.
- También generamos el botón de girar y parar con la función generarElementoHTML y la definición: layoutHTMLBotonBingo (de momento sin eventos).
- Añadimos al objeto bomboHTML el botón anterior.

❖ TABLA DE BOLAS

Invocamos el método generarTablaDeBolas que se describe a continuación. Este método genera la tabla con los huecos de las bolas donde deberán colocarse según salgan estas del bombo.

Método	generarTablaDeBolas ()	<div data-bbox="699 1012 1329 1339">  </div> <p>Lo primero es generar el layout de la tabla, tiene una zona izquierda, una derecha y el carril central por donde bajan las bolas del bombo.</p> <p>Se utilizarán los siguientes layouts:</p> <ul style="list-style-type: none"> • layoutHTMLAreaBolas => recoge todo el layout, y se despliega en mesaBomboHTML. (areaBolasHTML) • layoutHTMLColumnaBolas => con este layout generaremos tanto la columna izquierda como la derecha del tablero. (columnaBolasIzquierdaHTML, columnaBolasDerechaHTML) • layoutHTMLCarrilBolas => con él, diseñamos el carril central por el que bajarán las bolas. (carrilBolasHTML) <p>Huecos de las bolas</p> <p>Los huecos de las bolas se generarán mediante un bucle, para el total de bolas. Cada columna (izquierda y derecha) donde se desplegarán los huecos, contendrán 5 bolas por fila, o sea, los huecos para las bolas del 1 al 5 irán en la primera fila de la columna izquierda y del 6 al 10 en la primera fila de la columna derecha.</p> <p>Cada hueco se generará con el layout: layoutHTMLHuecoBola y se desplegará en la columna que corresponda. A cada elemento se le</p>
---------------	-------------------------------	--

		asignará el atributo id correspondiente al número del hueco que le corresponda.
	generarCartones()	Este método se invocará en el constructor del Bingo para generar los cartones establecidos en la propiedad numeroDeCartones.

SPRINT 4. – Sacar las bolas del bombo			
Vamos a girar el bombo y sacar las bolas			
Clase Bola			
Esta clase nos permite gestionar el objeto bola.			
Propiedades			
Constructor, parámetros que pasan a propiedades	bingo	Recogerá el objeto bingo (el propio juego)	
	numero	El número correspondiente a la bola.	
propiedades layout	elementoHTML		
Constructor - Proceso			
El proceso consistirá en tomar el HTML correspondiente a la bola y generar el elementoHTML y añadir este al carril por donde bajará la bola: bingo.carrilBolasHTML			
Modificaciones CLASE Bingo			
Creamos el método del juego que permite extraer una bola del bingo de forma aleatoria.			
Método	nuevaBola()	<p>Extraemos un número aleatorio entre 1 y el número total de bolas del bingo, que no deberá estar incluido en el array de números, a donde iremos añadiendo los números que salgan.</p> <p>Con el número aleatorio válido y el objeto (this) bingo, creamos un objeto bola con la clase Bola.</p> <p>Añadimos la bola al array de bolas y el número al array de números.</p> <p>Sustituimos el hueco en el tablero de bolas con la bola recién extraída.</p> <p>Para simular la bajada de la bola por el carril y su colocación en el tablero, vamos a utilizar dos temporizadores</p> <pre>let tempo = setTimeout(function () { bola.elementoHTML.style.bottom = "-500px"; let tempo = setTimeout(() => { let casilla = document.getElementById(aleatorio); let bottomPosition = casilla.style.bottom; bola.elementoHTML.style.bottom = bottomPosition; casilla.replaceWith(bola.elementoHTML); bola.elementoHTML.style.boxShadow = "7px 7px 5px 0px rgba(50, 50, 50, 0.75)"; bola.elementoHTML.style.transitionProperty = "bottom"; bola.elementoHTML.style.transitionDuration = "2s"; }, 1000); }, 1000);</pre>	
Añadir Evento al Botón (girar/parar)			
	<p>En el array de eventos de layoutHTMLBotonBingo, añadimos un objeto que permite gestionar el evento click:</p> <pre>{ evento: 'click', funcion: { (evento, bombo) => {.....} } }</pre>		

El parámetro evento, lo proporcionará el gestor del evento cuando se lance, el bombo es el objeto que se le ha pasado al añadir la función cuando generamos el elementoHTML.

El cuerpo de la función recogerá el siguiente proceso:

1. Para el bombo gestionaremos una nueva propiedad (girando), que será true o false, en función de si el usuario ha pulsado el botón de girar o parar.
2. Si bombo.girando es undefined, significa que no se ha pulsado nunca el botón, por lo que lo inicializamos a false, sino es undefined dejamos el estado que tuviera.
3. Si el bombo no está girando:
 - a. Ponemos a true bombo.girando
 - b. Cambiamos la etiqueta del botón a “Parar”
 - c. Añadimos un temporizador que cada 3 segundos procesará una nueva bola. setTimeinterval(**procesarBola()**, 3000); Guardamos el temporizador en una propiedad nueva del bombo: bombo.temporizador.
4. Si el bombo está girando:
 - a. Ponemos a false bombo.girando
 - b. Cambiamos la etiqueta del botón a “Girar”
 - c. Eliminamos el temporizador: clearInterval(bombo.temporizador).

La función **procesarBola()** cuya definición podemos incluirla dentro del bloque anterior), realizará los siguientes pasos:

1. Invocará al método nuevaBola() del bingo, en este caso le hemos llamado bombo (bombo.nuevaBola())
2. Si se han extraído todas las bolas:
 - a. Eliminamos el temporizador
 - b. Cambiamos la etiqueta del botón a “Finalizado”
 - c. Inhabilitamos el botón: disabled = true;

SPRINT 5. – Cantar línea y bingo

Para cantar la línea y el bingo, el jugador ha de ir marcando las celdas, para lo que se requiere que cada celda tenga su manejador de evento el usuario interactúe para marcarlas o desmarcarlas.

En el layout correspondiente a la celda, añadimos la función manejadora del evento 'click'.

```
const layoutHTMLCelda = {
  tipo: "div",
  atributos: {
    name: "celda",
  },
  estilos: {
    height: "50px",
    width: "50px",
    backgroundColor: "whitesmoke",
    border: "1px solid snow",
    display: "flex",
    justifyContent: "center",
    alignItems: "center",
    fontSize: "2em",
    fontFamily: "Arial",
    fontWeight: "bold",
    userSelect: "none",
  },
  eventos: [
    {
      evento: "click",
      funcion: (evento, celda) => {
        if (celda.numero !== 0) {
          // Gestionamos el estado de la celda.
          // sino tiene estado o es falso (no pulsada)

          if (celda.estado == undefined || !celda.estado) {
            // cambiamos estado a true (pulsada)
            celda.estado = true;
            // invertimos el color.
            evento.target.style.filter = "invert(100%)";
            // Comprobamos Bingo, este método comprobará línea y bingo.
            Bingo.comprobarBingo(
              celda.linea.carton.juego, // Pasamos el juego (bingo)
              celda.linea,              // Pasamos la línea
              celda.linea.carton        // Pasamos el cartón
            );
          } else { // Si el estado era true (pulsada)
            // cambiamos estado a false (no pulsada)
            celda.estado = false;
            // invertimos el color.
            evento.target.style.filter = "";
          }
        }
      },
    },
  ],
};
```

Cada vez que el jugador marca una de las celdas del bingo, debemos comprobar si se ha “cantado línea” y, en su caso, si se ha “cantado bingo”.

El proceso será el siguiente, y lo recogeremos en un método estático de la clase Bingo, que se llamará comprobarBingo.

1. Comprobamos si el jugador ha marcado todas las celdas de la línea, si es así, estará reclamando “Línea”.
2. Si ha cantado Línea:
 - Si el bombo estaba girando debemos pararlo.
 - Emitir la alerta de que el jugador ha cantado Línea. “Cantada línea”
 - Comprobar que realmente todos los números de la línea han salido del bombo.
 - Si la línea es correcta, etiquetamos a true su estado y:
 - i. Cambiamos el color de todas las celdas de esa línea, y eliminamos todos los eventos de las celdas para que no se puedan reprocesar.
 - ii. Debemos comprobar si todas las líneas ya están marcadas, ya que si fuera así estaría reclamando “Bingo”. Se emite la alerta “Cantado Bingo”.
 - iii. Si todas las líneas están ya marcadas revisamos si los números del cartón son correctos. En ese caso el juego del Bingo finaliza.
 - Si el juego no ha finalizado, si el Bombo se había parado, se reinicia.

Modificaciones CLASE Bingo		
Creamos los métodos estáticos para comprobar la línea y el bingo		
Método	comprobarBingo	<p>Tal como se puede ver al comienzo de la descripción del sprint, este método se invocará desde el evento ‘click’ de cada celda de la tarjeta:</p> <pre>// Comprobamos Bingo, este método comprobará línea y bingo. Bingo.comprobarBingo(celda.línea.cartón.juego, // Pasamos el juego (bingo) celda.línea, // Pasamos la línea celda.línea.cartón // Pasamos el cartón);</pre> <p>El método recibe tres parámetros, el propio juego del bingo, el objeto línea y el objeto cartón.</p> <p>Inicializamos los siguientes controladores:</p> <ul style="list-style-type: none">• cantaLinea => será true si todas las celdas de la línea, que contienen un número, tienen el estado a true.• cantaBingo => se inicializa a false;• líneaOk => se inicializa a false• bingoOk => se inicializa a false <p>Si cantaLinea es verdadero (el jugador ha cantado línea):</p> <p>Debemos parar el bombo:</p> <p>Generamos un objeto, evento ‘click’ para poder disparar el botón de arranque parada: evento = new Event(‘click');</p>

		<p>Inicializamos un control (arrancarBingo = false) para saber si el Bingo estaba parado o girando, de tal forma que podamos devolver su estado una vez finalizado el proceso de comprobación.</p> <p>Si el bombo estaba girando (bingo.girando), paramos el bombo utilizando el evento generado: bingo.botonBingoHTML.dispatchEvent(evento)</p> <p>Una vez parado ponemos a true nuestra variable arrancarBingo , para reiniciar el bombo cuando finalizamos la comprobación.</p> <p>lineaOk recogerá el resultado de la comprobación de la línea, que se realizará con el método estático comprobarLinea, al que le pasaremos el bingo y la propia línea.</p> <p>Si la línea es OK, se marcará la propiedad estado del objeto línea a true y se cambiará el color de fondo de cada una de sus celdas al deseado.</p> <p>Sólo si la línea es OK, cantaBingo recogerá el resultado de comprobar si todas las líneas del cartón son OK también y en el caso que sea verdadero, se procederá a revisar el bingo con el método estático revisarBingo al que se le pasará el cartón.</p> <p>Una vez finalizadas las comprobaciones, si el bingo estaba arrancado se reinicia utilizando el evento que habíamos generado inicialmente.</p>
Método	comprobarLinea	<p>Recibe como parámetros los objetos bingo y línea (que queremos comprobar)</p> <p>Recorremos todos los números de cada celda de la línea y comprobamos si existen en nuestro array de números del Bingo, en ese caso, si todos están en el array, devolvemos true, en otro caso false.</p>
	revisarBingo	<p>Recibe como parámetro el cartón que queremos comprobar</p> <p>Recorremos todas las líneas del cartón y realizamos la comprobación de línea. Si todas las comprobaciones son ok devolvemos true, en otro caso false.</p>