

2024

Proyecto de Desarrollo Web en Entorno Cliente

Gabriela Barton

Javier Melendo

**Juego del Solitario**

**Índice**

[Introducción 3](#_Toc158397605)

[Desarrollo 3](#_Toc158397606)

[Diario de bitácora 3](#_Toc158397607)

[Métodos desarrollados 5](#_Toc158397608)

[Conclusiones 5](#_Toc158397609)

[Bibliografía/Webgrafía 5](#_Toc158397610)

[Anexos 6](#_Toc158397611)

**(recordar poner contenido indice en arial 11 cuando terminemos)**

# Introducción

La presente memoria documenta el desarrollo de un juego interactivo implementado mediante las tecnologías web fundamentales: HTML, CSS y JavaScript. Este proyecto tiene como objetivo principal aplicar los conocimientos adquiridos en las clases teóricas y explorar la versatilidad de las tecnologías web en la creación de experiencias interactivas.

El juego propuesto (“El Solitario”) está diseñado utilizando las bibliotecas JavaScript, Bootstrap y jQuery, con el fin de aprovechar sus capacidades para mejorar la estética y funcionalidad de la interfaz de usuario. Todos los recursos del juego están contenidos en la carpeta del proyecto, garantizando que funcione correctamente con independencia de su ubicación en el sistema de archivos.

En cuanto al código, hemos prestado especial atención para asegurar su compatibilidad en diversos navegadores, como Chrome y Firefox. También nos hemos enfocado en optimizar su limpieza y eficiencia, además de elaborar una memoria detallada que profundiza en los aspectos técnicos de su desarrollo.

# Desarrollo

En esta sección detallaremos el proceso lógico de elaboración del código, los obstáculos que aparecieron y el proceso de toma de decisiones y resolución de problemas.

## Diario de bitácora

Día 1:

El primer paso es estudiar el material proporcionado. Tras ver el código y las indicaciones sugeridas, **colocamos todas las variables globales en la cabeza de nuestro archivo** javascript: los palos y los números de las cartas, los tapetes, mazos, contadores de cartas y las variables de tiempo (el número de segundos, el temporizador que los incrementa y el elemento del DOM que los muestra formateados en hh:mm:ss).

Para poder hacer pruebas más cómodamente, **reducimos el array del número de cartas al rango de 9-12** (en lugar de 1-12 como es la versión finalizada).

Lo primero que hay que hacer al empezar a jugar a las cartas es barajarlas, así que empezamos a pensar en el método “barajar”. Se nos da pistas al respecto: el método sugerido consiste en cambiar cartas de posición tantas veces como número de cartas haya. Sin embargo, aún no tenemos mazo (el array asignado al mazo está vacío), así que el primer paso es [**llenar el mazo (1)**](#llenarMazo). Ahora tenemos un método que carga el array de “cartas” (líneas de código que referencian imágenes de una carpeta) dentro del mazo inicial.

Día 2

Hoy nos ha costado más comenzar a desarrollar el juego porque no sabíamos muy bien por dónde continuar. Decidimos que nuestro próximo objetivo es poner las cartas sobre el tapete inicial, así que nos lanzamos a desarrollar nuestro método [**cargar\_tapete\_inicial (2)**](#cargar_tapete_inicial). Al principio sólo nos salía una carta en el visualizador, pero no tardamos en darnos cuenta de que en realidad sí estaban todas… unas encima de otras. Para poder verlas en “cascada”, es necesario aplicar la variable “paso” que nos da unos píxeles de diferencia abajo y a la derecha de una carta con otra. La mayor dificultad: darnos cuenta darnos cuenta de que había que multiplicar ‘paso’ por el iterador y no sumarlo.

Seguidamente nos decantamos por crear un **método que actualice los contadores** de cada tapete, de forma que aparezca el número de cartas que tenemos en el tapete inicial, cuarenta y ocho.

Hemos pasado un largo rato sin que nos funcionase nada hiciéramos lo que hiciéramos, y entonces es cuando hemos visto que había un error: no nos funcionaba el script porque los contadores de cartas NO estaban bien definidos. El código provisto de script definía los contadores con un nombre como *cont\_sobrantes,* mientras que en el html los contenedores equivalentes eran “*contador\_sobrantes*”

(donde ponía *let = document.getElementById("cont\_sobrantes")*; el Id en realidad era)

Avanzamos quitando la propiedad ‘draggable’ a todas las cartas para que sólo podamos modificar la última del tapete inicial.

Día 3

Hoy hemos estado mirando los métodos de evento para soltar imágenes de un lado a otro. Gracias al ejemplo dado, hemos seguido esas funciones y las hemos modificado ligeramente para poder usarlas en nuestro proyecto.

La función dragStart() la hemos implementado añadiendo una línea más, y hemos creado otra igual pero para los movimientos del tapete de sobrantes, de tal forma que la función pase información extra cuando hacemos el drop().

function dragStart(event) {

    event.dataTransfer.setData("Text", event.target.id);

}

function dragStartSobrantes(event) {

    event.dataTransfer.setData("Text", event.target.id);

    event.dataTransfer.setData("fromSobrantes", "true");

}

Simplemente le hemos añadido la línea event.dataTransfer.setData(“fromSobrantes”,”true”) para que cuando hagamos un drop en uno de los 4 tapetes receptores sepan que esa carta viene del tapete de sobrantes y que hagan el método desdeSobrantes() en vez de la jugada normal (hemos hecho dos tipos de jugada, una si la carta la cogemos desde el tapete de sobrantes y otra si la cogemos del tapete inicial).

La función allowDrop() la hemos dejado exactamente igual a como estaba.

Para el método drop() hemos hecho que pille el target de forma que si lo sueltas encima del tapete 1 o en la ultima carta de ese tapete haga una cosa u otra. También hemos hecho que sepa si la carta viene de sobrantes (con la función de dragStartSobrantes()).

function drop(event) {

    event.preventDefault();

*//comprobamos que soltemos la carta o en un tapete o encima de la ultima carta de ese tapete*

    if (event.target == tapete\_receptor1 || event.target == tapete\_receptor1.lastChild) {

        if (event.dataTransfer.getData("fromSobrantes") === "true") {

            desdeSobrantes(mazo\_receptor1, tapete\_receptor1, cont\_receptor1);

        } else {

            jugada(mazo\_receptor1, tapete\_receptor1, cont\_receptor1);

        }

    } else if …(lo mismo para los otros 3 tapetes y el tapete de sobrantes)

<https://www.w3schools.com/html/html5_draganddrop.asp>

// event.preventDefault() se utiliza para permitirte soltar el elemento que arrastras

// allowDrop(event) se llama cuando se está arrastrando algo sobre un elemento

// event.dataTransfer.setData("text", event.target.src) establece los datos que se deben transferir durante el arrastre (la URL de la imagen)

// drop(event): Esta función se llama cuando se suelta algo sobre un elemento

Día 4

Método para obtener la última carta

(método .pop() de un array separa el elemento y lo elimina)

Para evitar repetir el código por cada jugada, hemos creado un array de tapetes receptores y aplicado el mismo método a todos ellos mediante un bucle.

Ahora tenemos problemas con el método reset.

Cuando intentamos reiniciar el juego, aunque las imágenes se ponen en su sitio, algo se desbarajusta en los tapetes receptores y en vez de poder poner el 12 de nuevo sólo nos deja continuar sobre “el fantasma” de la última carta que pusimos. Como no sabemos dónde está el error después de muchas pruebas, vamos a peinar el código y a reasignar las variables de forma adecuada, diferenciando variables que no cambian su referencia (const) de las que sí (let).

Tras dos horas de rediseño, nos hemos dado cuenta de que estábamos trabajando en el archivo equivocado y la previsualización no coincidía con nuestras modificaciones.

Día 5

Perfilamos detalles (temporizador, estructura del código) y añadimos CSS y animaciones como confeti.

Día 6

Nos damos cuenta de que animar el confeti tomaría mucho tiempo para entender cómo hacerlo bien, bonito y estético sin importar librerías ajenas, así que decidimos decantarnos por una forma mucho más sencilla: utilizando un gif.

Suprimimos el método “verificar victoria” cuya única función era comprobar si el contador inicial y de sobrantes estaba a 0 (lo que significaría que se han colocado todas las cartas en el resto de tapetes), para incluirlo en el método [**victoria (X)**](#victoria).

Modificaciones en CSS

Día 7

Hoy nos hemos dedicado a hacer pruebas y hemos encontrado una serie de errores:

**Error 1**: Si tomabas una carta del tapete de sobrantes y la soltabas sobre el mismo tapete, robaba una carta del tapete inicial (esto no debería ocurrir). Es porque en el código habíamos dado por supuesto que todo lo que soltásemos sobre el tapete de sobrantes vendría del tapete inicial y no de sí mismo. Solucionado.

**Error 2:** Al minimizar la ventana quedaban raros los tapetes al ser responsive y no pillaban bien las cartas cuando las soltábamos, lo hemos solucionado cambiando cosas del css y poniendo estilos a las cartas.

## Métodos desarrollados

**1 -- llenarMazo()**

Con esta función, llenamos el array “mazo\_inicial” con rutas a las imágenes de las 48 cartas de la baraja. La nomenclatura que tienen todas es “ruta/numero-palo.png”, así que hacemos dos bucles que recorran todos los números de cada palo y así conseguimos generar las 48 rutas.

**2-- cargar\_tapete\_inicial()**

Con esta función, recorremos el array del “mazo\_inicial” y vamos creando un elemento img para cada carta, poniéndoles sus atributos y haciendo un append.Child al tapete inicial para que aparezcan en pantalla.

**X—jugada (mazo, tapete, cont)**

Con esta función, realizamos la jugada que ocurre cuando coges la última carta del mazo inicial y la pones en uno de los tapetes. Existen dos casos: para los 4 tapetes receptores o para el tapete de sobrantes. La función coge la última carta del mazo inicial y con un pop la quita y la pone al mazo que queremos introducirla. Después realizamos un proceso parecido al de cargar\_tapete\_inicial(), creamos un elemento imagen y le meteremos los datos necesarios para que en pantalla aparezca la última carta del mazo que coincide con su tapete.

Aquí es donde los dos casos son importantes, ya que si la carta es soltada en el tapete de sobrantes se le aplicara el atributo draggable=true y ondragstart=”draggStartSobrantes(event)” (hacemos que la carta se pueda seguir moviendo). En cambio, si la carta es soltada en un tapete receptor se le pondrán el atributo draggable=false (para que esa carta no se pueda mover de ese tapete) y se verificara que si el mazo esta vacío la carta que pongas sea rey o que el palo y el numero sean correctos según la carta que ya este puesta en ese tapete.

Para terminar, incrementamos la cuenta del tapete en el que dejamos la carta, incrementamos la cuenta de movimientos y le quitamos uno al contador del tapete inicial. También borramos la última carta del tapete inicial (pues es la que hemos colocado en otro tapete) y llamamos a 3 funciones: draggeable(), verificarMazoInicial() y victoria().

**X—draggeable()**

Esta función ocurre cada vez que realizamos una jugada quitando la última carta del tapete inicial. Coge la utima carta del tapete inicial y le incluye los atributos ondragstart = dragStart(event), draggable = true y id=”carta” (este ultimo hace una animación de zoom cuando pones el raton encima de la ultima carta del tapete).

**X—verificarMazoInicial()**

Esta función hace que cuando el contador de cartas del mazo inicial llegue a 0 se pongan las cartas del mazo sobrantes en el inicial ya barajeadas y saliendo en pantalla. Esto ocurre llamando a las funciones barajar() (para cambiar el orden de las cartas), cargar\_tapete\_inicial() (para que aparezcan en pantalla), reserTapete() (para limpiar el tapete de sobrantes), set\_contador() (para poner el contador de cartas sobrantes a 0) y draggeable() (para que podamos mover la ultima carta del tapete principal).

**X—arrancar\_tiempo() y parar\_tiempo()**

Para contabilizar el tiempo utilizamos una función del objeto Window llamada *setInterval()*, la cual configura un temporizador que ejecuta una función cada X milisegundos ([Kantor, 2022](#Bibliografia)). Nos han dado la función “hms()” que trunca los segundos para mostrarlos en formato hh:mm:ss, incrementa un segundo cada vez que se ejecuta y los manda a una variable.

**X—desdeSobrantes()**

Básicamente realiza la misma función que jugada() pero en vez de quitarle la ultima carta al mazo inicial se lo quita al mazo de sobrantes. **(esta función se puede quitar y cambiar cosas en la de jugada(), creo)**

**X—barajar()**

Esta función recorre el mazo desde la última carta hasta la primera, y saca un número aleatorio entre el número de cartas que todavía no han sido cambiadas en el mazo (si el primer Math.random es entre 0 y i, el siguiente será entre 0 y i-1, y así sucesivamente). Una vez sacado el número, intercambiamos la carta[i] del mazo por la carta [número aleatorio].

**X—obtenerUltimaCarta()**

Esta función se encarga de coger la última carta del mazo que le pongas y consigue sacar su número y su palo. Como el valor de las cartas es el nombre de la imagen que tiene cada una (y todas tienes su número y el palo en su nombre), dividiremos el valor de la carta en partes para conseguir el número y el palo.

**X—comprobarRey()**

Esta función guarda en una variable los valores que devuelve la función obtenerUltimaCarta(), de tal forma que si el número de la carta es 12 (rey) devuelve true y si es otro número devuelve false.

**X—compatibilidadCarta()**

A esta función le pasamos los dos mazos que se verán afectados (el mazo del cual viene la carta que hemos cogido y el mazo en el cual vamos a soltar la carta). Obtenemos el número y el palo de la ultima carta de los dos mazos (con la función obtenerUltimaCarta()) y creamos dos variables más que son los palosRojos[“cua”,”ova”] y palosGrises[“cir”,”hex”]. Primero comprobamos que el número de la carta que hemos cogido sea uno menor que el de la carta ya puesta en el mazo. Si se cumple eso, comprobaremos que si una de las cartas tiene palos grises, la otra debe tener palos rojos y viceversa. Si cumple con el numero y los palos devuelve true, sino devuelve false.

**X—resetTapete()**

Esta función se encarga de quitar todos los elementos img del tapete que le pases, es decir que borra todas las cartas puestas en pantalla de ese tapete.

**X—inc\_contador()**

Le pasas un contador y se encarga de sumarle 1.

**X—dec\_contador()**

Le pasas un contador y se encarga de restarle 1.

**X—set\_contador()**

Le pasas un contador y el valor que quieres que tenga.

**X—dragStart()**

Se encarga de mandar información del objeto que mueves al objeto que lo recibe.

**X—dragStartSobrantes()**

Manda información al tapete receptor de que la carta que vas a soltar viene del tapete de sobrantes.

**X—allowDrop()**

Permite que se puedan soltar elementos.

**X—drop()**

En esta función comprobamos sobre que tapete estamos dejando la carta y realizamos una jugada según el tapete y la información correspondiente. Por ejemplo si la carta viene del tapete sobrantes se realizara desdeSobrantes(), pero si viene del tapete inicial se realizara jugada().

**X-- victoria()** y **pararVictoria()**

Si los contadores del tapete inicial y del tapete de sobrantes están a 0 significa que hemos colocado todas las cartas y que hemos ganado. En este caso: paramos el tiempo, mostramos nuestra ventana emergente de victoria junto con nuestro gif de confeti y reproducimos un sonido. No nos hemos olvidado de crear otro método inverso a este que oculte la ventana emergente, el gif y detenga el sonido.

**X—comenzar\_juego()**

Es una compilación de llamadas a funciones usadas para resetear los contadores a 0, parar el mensaje de victoria(si esta activo), reiniciar los mazos y los tapetes, llenar el mazo inicial, barajearlo y mostrarlo en pantalla, etc.

# Conclusiones

texto

# Bibliografía y recursos

texto

Audio de javascript - <https://lenguajejs.com/javascript/multimedia/api-multimedia-nativa/>

Gif del confeti - <https://en.picmix.com/stamp/confetti-gif-deco-1434843>

Imagen tapete - <https://www.pokerproductos.com/Tapete-antideslizante-bridge-Belote>

Kantor, I. (2022) *Planificación: setTimeout y setInterval*. El Tutorial de JavaScript Moderno. Consultado el XXXXXXX de 2024. <https://es.javascript.info/settimeout-setinterval>

# Anexos

