

2024

Proyecto de Desarrollo Web en Entorno Cliente

Gabriela Barton

Javier Melendo

**Juego del Solitario**

**Índice**

[Introducción 3](#_Toc158829562)

[Desarrollo 3](#_Toc158829563)

[Diario de bitácora 3](#_Toc158829564)

[Métodos desarrollados 5](#_Toc158829565)

[Conclusiones 6](#_Toc158829566)

[Bibliografía y recursos 6](#_Toc158829567)

[Anexos 7](#_Toc158829568)

**(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 (“*drag and drop*”). Gracias al ejemplo provisto por nuestro profesor y a la información en la página web de [W3Schools](#Bibliografia), hemos sabido utilizar las mismas funciones con adaptaciones sutiles para poder usarlas en nuestro proyecto.

Hemos añadido una línea a la función dragStart() y hemos creado otra similar para que responda a un evento específico (movimientos del tapete de cartas sobrantes) para que ambas puedan transferir la información al efectuar un “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");

}

Al añadir la línea event.dataTransfer.setData(“fromSobrantes”,”true”) es posible que los tapetes receptores obtengan información sobre el origen de los datos que se intentan transferir, y de esta forma podemos ejecutar un tipo de jugada u otra. Hemos decidido crear dos tipos de jugada: la estándar (una carta llega desde el tapete inicial) y la que viene desde el tapete de cartas sobrantes. En ambos casos, los datos se transfieren a uno de los cuatro tapetes receptores, que a través del método adecuado obtendrán la información transferida desde su correspondiente mazo.

// 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.

## 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 este método, insertamos elementos

**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-- 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.

# 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>

W3Schools. (s. f.) *HTML Drag and Drop API*. Consultado el 25 de enero de 2024 en <https://www.w3schools.com/html/html5_draganddrop.asp>

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

# Anexos

