Juego del Solitario

Proyecto de Desarrollo Web en Entorno Cliente

Javier Melendo Gabriela Barton

2024

Índice

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)**. 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 (2)** 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. Nos damos cuenta de que en realidad ese método va a ir incrementando y decrementando los contadores con las acciones que vayamos ejecutando, y que el juego termina cuando el contador de los tapetes superiores llegue a 0… Así nace la **jugada (3).** Terminamos de desarrollar el método **barajar (4)**.

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 de script provisto definía los contadores con el nombre de *cont_sobrantes,* mientras que en el html los contenedores equivalentes eran "*contador_sobrantes*". Por ello las funciones no podían obtener la información necesaria de los contenedores, puesto que sus valores no coincidían.

Una vez localizado el error avanzamos quitando la propiedad 'draggable' a todas las cartas para que sólo podamos modificar la última del tapete inicial, puesto que de otro modo podíamos arrastrar cualquier carta del tapete inicial y sólo nos interesaba que pudiera tocarse la última. Creamos el método **draggeable (5)**.

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, hemos sabido utilizar las mismas funciones con adaptaciones sutiles para poder usarlas en nuestro proyecto:

```
// event.preventDefault() se utiliza para anular el comportamiento predeterminado del navegador, lo que permitiría que el elemento arrastrado se suelte correctamente;
// allowDrop(event) se llama cuando se está arrastrando algo sobre un elemento y contiene lógica para determinar si se permite soltar el elemento en un momento y lugar dados;
// event.dataTransfer.setData("text", event.target.src) establece los datos que se deben transferir durante el arrastre (la URL de la imagen);
// drop(event) procesa lo que sucede después de soltar el elemento arrastrado, utilizando los datos configurados previamente con event.dataTransfer.setData;
```

Hemos añadido una línea a la función *dragStart* y hemos creado otra similar llamada ***dragStartSobrantes* (6)** que responde a un evento específico (movimientos del tapete de cartas sobrantes). Así ambas pueden transferir la información correcta al efectuar un "drop" de carta.

El motivo de crear dos funciones tan similares es que necesitamos conocer el origen del evento para que al transferir la carta de un array a otro, se escoja el array indicado. Puesto que sólo existen dos posibilidades, hemos creído oportuno utilizar un *boolean* que permita ejecutar diferentes tipos de jugadas según el origen de la carta, ya sea desde el tapete inicial o desde el tapete de cartas sobrantes. En ambos casos, los datos se transfieren a uno de los cuatro tapetes receptores mediante el método adecuado para obtener la información desde su respectivo mazo.

Día 4

Puesto que teníamos que comprobar varias veces la última carta de distintos mazos, hemos creado un método sencillo para obtener la última carta de un mazo: **obtenerUltimaCarta (7)**.

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

Intentando desarrollar el método de **reseteo (8)** nos surgen problemas: aunque las imágenes se ponen en su sitio, algo se desbarajusta en los tapetes receptores y en vez de poder poner la carta 12 de nuevo sólo nos deja continuar sobre "el fantasma" de la última carta que pusimos (es decir, sólo podemos poner la carta 11). Como no sabemos dónde está el error después de muchas pruebas, peinamos el código, reasignamos las variables diferenciando las que no cambian su referencia (*const*) de las que sí (*let*), testeamos las salidas por consola… y 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

Cuando el contador del tapete inicial llega a 0 pero seguimos teniendo cartas en el mazo sobrante, tendremos que transferir los elementos de un mazo a otro y barajarlos por el camino. Creamos un método que compruebe si es el caso llamado **verificarMazoInicial (9)**.

Perfilamos detalles (temporizador, estructura del código) y añadimos CSS. Intentamos aplicar una animación con CSS para hacer confeti, pero es muy costoso y, después de intentar importar librerías ajenas sin éxito, nos decantamos por utilizar un *gif* con el mismo efecto.

También hemos creado un método **parar_tiempo (10)** basado en el ya existente arrancar_tiempo().

Día 6

Vamos limpiando código innecesario y fusionando métodos que pueden funcionar como uno solo. Por ejemplo, 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 (11)**.

También modificamos el CSS para que sea *responsive* y pueda verse en pantallas grandes y pequeñas manteniendo su funcionalidad, así como adición de cosas como efectos visuales y sonido de victoria.

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 captaban bien las cartas cuando las soltábamos. El problema era que el *span* de contadores se superponía al *div* de las cartas, así que hemos ajustado su tamaño y posición para que no interfiera. Solucionado.

Día 8

Elaboración de la memoria. Correcciones menores de CSS. Revisión de bibliografía.

Día 9

Vamos a añadir un podio al juego que almacene las mejores 5 jugadas que se hagan en un navegador determinado. Para ello, utilizaremos el objeto *LocalStorage* para que almacene los datos que nos interesan en un archivo propio del navegador, de forma que no se pierdan cuando cerremos la ventana.

El obstáculo que nos surge es el de almacenar los datos como un array, puesto que todos los objetos de tipo Storage almacenan únicamente cadenas de texto plano. Es por esto que hemos tenido que investigar si había algún equivalente al 'serialize' que utilizamos en PHP, y efectivamente, nos encontramos con JSON y sus métodos explicados en W3Schools, que nos permiten convertir los arrays en cadenas y después reconvertir esas cadenas en arrays para poder manipularlas. Así es como hemos decidido crear el método **guardarPuntuacion (12)**, para que podamos incluirlo cuando haya una victoria, pasándole por parámetro los segundos almacenados y el número de movimientos.

Finalmente, desarrollamos los métodos para mostrar y manipular las puntuaciones que vamos almacenando en una tabla en nuestro HTML. Estos son **obtener** y **actualizarPodio (13).** Con ello terminamos el proyecto.

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()

Esta función está diseñada para cargar un conjunto de imágenes (cartas) en un elemento HTML (tapete) en función del mazo proporcionado. 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.

3-- jugada (mazo, tapete, cont)

Una jugada ocurre cuando se arrastra la última carta de un mazo superior y se coloca en uno de los tapetes.

Esta función es troncal y determina qué ocurre en los distintos escenarios de interacción que puede tener el usuario con el juego. Cada actuación del jugador actualizará los datos de forma acorde permitiendo la resolución final del problema: conseguir que los contadores de cartas del tapete superior (tanto el mazo inicial como el mazo de cartas sobrantes) llegue a 0.

Existen dos tipos de jugada: aquella donde soltamos la carta en un tapete inferior o aquella donde pongamos la carta en el mazo de sobrantes. Si la carta es soltada en el tapete de sobrantes se le aplicará 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 verificará que si el mazo está vacío la carta que pongas sea rey (número 12), o que el palo y el numero sean correctos según la carta que ya este puesta en ese tapete (nos serviremos del método **compatibilidadCarta (14)**).

Después de cada movimiento, asegura que la última carta en el tapete sea "arrastrable", es decir, que pueda moverse. También verifica si el mazo inicial está vacío (y las acciones en consecuencia, pues puede significar que se haya ganado el juego). En caso de haber una victoria, llama a la función correspondiente.

4-- 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. Una vez sacado el número, intercambiamos la carta[i] del mazo por la carta [número aleatorio].

5-- draggeable()

Esta función se llama cada vez que realizamos una jugada quitando la última carta del tapete inicial. Coge la última carta del tapete inicial y le incluye los atributos *ondragstart = dragStart(event)*, *draggable = true* y *class="carta"* para poder aplicar los estilos CSS correspondientes.

6-- dragStart(), dragStartSobrantes() y drop()

Se encarga de mandar información del objeto que mueves al objeto que lo recibe, diferenciando su origen (boolean *false* si viene de tapete inicial, *true* si viene desde sobrantes).

Al hacer *drop()* comprobamos sobre qué tapete estamos dejando la carta y realizamos una jugada según el tapete y la información correspondiente. Si la carta viene del tapete sobrantes se realizará la jugada *desdeSobrantes()*, pero si viene del tapete inicial se realizará la *jugada()* normal.

7-- obtenerUltimaCarta(mazo)

Esta función se encarga de coger la última carta del mazo que pasemos por parámetro 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 tienen su número y el palo en su nombre), dividiremos el valor de la carta en partes para conseguir el número y el palo. Utilizamos el método .pop() de Array para almacenar el último elemento de un array en una variable y eliminarlo de su colección original.

8-- comenzar_juego()

Es una compilación de llamadas a funciones usadas para resetear el juego. Antes que nada actualiza el podio con los datos de las últimas 5 mejores victorias (el criterio es el menor tiempo posible), reinicia los contadores a 0, elimina todas las cartas de los tapetes, vacía todos los mazos, vuelve a cargar el mazo inicial y a barajarlo, carga las cartas sobre el tapete grande y arranca de nuevo el tiempo.

9-- verificarMazoInicial()

Esta función verifica si el mazo inicial se ha quedado sin cartas, lo que significa que su contador está a 0. Llegado a este punto, es necesario comprobar si el mazo sobrante tiene alguna carta, pues si no la tuviera tampoco significaría que todas las cartas están debidamente jugadas y que ha terminado la partida.

Cuando quedan cartas en el mazo de sobrantes, necesitamos "pasarlas" al tapete inicial de nuevo. Para ello, copiamos ese mazo, lo barajamos y lo colocamos en el mazo inicial. Además, actualiza los contadores de cartas de ambos tapetes y convierte en 'draggable' o movible la última carta del mazo inicial, de forma que sea posible continuar jugando.

10-- 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). 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, mostrada en su contenedor de tiempo de la página web.

11-- 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.

12-- guardarPuntuacion()

Esta función toma una nueva puntuación junto con un nombre opcional, la agrega al podio existente (con un máximo de 5 posiciones), ordena los ítems y luego guarda los datos del podio actualizado en un localStorage.

13-- obtenerPodio() y actualizarPodio()

El método *obtenerPodio()* nos devuelve datos sobre victorias anteriores en caso de que las hubiera. Si no, nos devolverá un array vacío. El método actualizarPodio() es el que se encarga de actualizar visualmente la tabla del podio en la interfaz de usuario, para lo cual elimina todas las filas menos el título y el encabezado, obtiene las puntuaciones actuales y las inserta en la tabla.

14-- compatibilidadCarta()

A esta función le pasamos los dos mazos que se verán afectados (el mazo del cual viene la carta que estamos arrastrando y el mazo en el cual vamos a soltar la carta). Obtenemos el número y el palo de la última carta de los dos mazos 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 número y los palos devuelve true, sino devuelve false.

Esta función es imprescindible para la jugada, puesto que determinará si es posible soltar la carta que estamos arrastrando sobre otro mazo en función de su compatibilidad con éste.

Conclusiones

El objetivo del proyecto era aplicar los conocimientos adquiridos sobre JavaScript en un proyecto práctico y con funcionalidad, y hemos sido capaces de realizarlo en el tiempo estimado, trabajando en equipo y aprendiendo cosas nuevas sobre la marcha, como algunas funciones de JavaScript o cómo trabajar con Git.

Si bien hemos apostado por el formato de "diario", no ha sido completamente riguroso porque a veces uno se "pierde" en el trabajo que está haciendo y nos olvidábamos de registrar los cambios y actualizaciones de forma metódica. Aun así nos ha venido bien hacer este ejercicio para adquirir rutina para más adelante. Nos ha venido muy bien utilizar un sistema de control de versiones (Git) comparar código y llevar ese registro. En un momento dado pudimos dar marcha atrás cuando guardamos unos cambios que no nos convenían.

Podríamos haber organizado mejor las tareas y el rumbo en la toma de contacto con el proyecto, pero hemos tenido la fortuna de haber fluido bien como grupo, haber estado de acuerdo en la toma de decisiones y resolución de problemas y haber aprendido el uno del otro durante este tiempo. La comunicación abierta y continuada ha contribuido innegablemente al éxito del proyecto.

En cuanto al aprendizaje más técnico, hemos podido:

- **Desarrollar habilidades prácticas** de programación para crear funciones que tienen una utilidad real y tangible, como un juego de cartas. Podemos llevar a cabo proyectos prácticos de forma efectiva.
- **Comprender nuevos conceptos** de programación, relativos al DOM, al manejo de arrays con JSON y otros objetos, aplicando la lógica de programación.
- **Solucionar problemas en tiempo real**, enfrentando desafíos prácticos e imprevistos que surgían durante el desarrollo del código ("*ahora no funciona esto, ahora se desconfigura aquello…*").
- **Aplicar buenas prácticas** como la modularización del código en funciones, el uso de parámetros y la organización lógica del flujo de ejecución, manteniendo un código limpio y fácil de entender.
- **Practicar la aplicación de estilos** y diseños con CSS para crear una experiencia de usuario agradable.

En conclusión, agradecemos la oportunidad de haber podido llevar a cabo este proyecto y de entender un poco mejor la utilidad y el alcance de las tecnologías en el entorno cliente.

Bibliografía y recursos

Apuntes y webgrafía:

García, M. (2024). *U.D. 5: Utilización del modelos de objetos del documento (DOM - Document Object Model)* [manuscrito no publicado]. Desarrollo web en entorno cliente, CPIFP Los Enlaces.

Kantor, I. (2022) *Planificación: setTimeout y setInterval*. El Tutorial de JavaScript Moderno. Consultado el 19 de enero de 2024 en https://es.javascript.info/settimeout-setinterval

Román, J. (2021) *API de Audio Javascript - Javascript en español*. Consultado el 7 de febrero de 2024 en https://lenguajejs.com/javascript/multimedia/api-multimedia-nativa/

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

W3Schools. (s. f.) *JSON.stringify()*. Consultado el 20 de febrero de 2024 en https://www.w3schools.com/js/js_json_stringify.asp

Recursos:

confetti gif deco. (2019) PicMix. En https://en.picmix.com/stamp/confetti-gif-deco-1434843

Crowntes (2021)*Victory Sound Effect* [Vídeo]. YouTube. https://www.youtube.com/watch?v=ksN6oqzwdaU

García, M. (2024). *Proyecto: Juego del Solitario* [manual]. Desarrollo web en entorno cliente, CPIFP Los Enlaces.

PokerProductos.com (2020) *Tapete antideslizante bridge Belote* [imagen]. https://www.pokerproductos.com/Tapete-antideslizante-bridge-Belote

Anexos

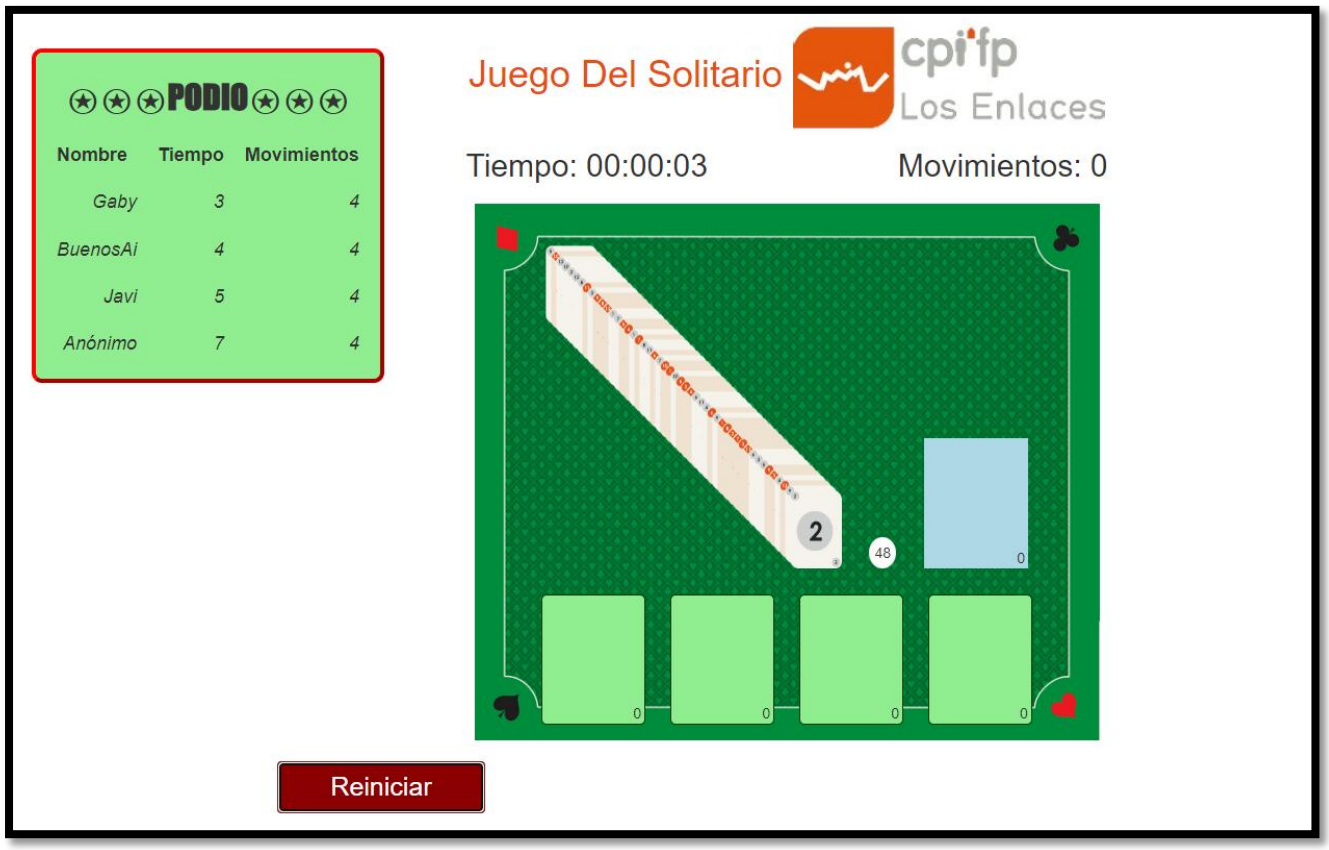

Ilustración 1. Vista normal del juego

Ilustración 2. Vista en móvil

Ilustración 3. Vista de victoria