

Solitario

23/02/2023

—

Iván Mosteo Lop, Raúl Acín Herrero y Julián Aragón Guerrero

Grupo 3

CPIFP Los Enlaces

2º DAW Vespertino



ÍNDICE

Introducción	2
Desarrollo: HTML	4
Desarrollo: CSS	4
Desarrollo: JAVASCRIPT (declaración de variables)	5
Desarrollo: JAVASCRIPT (drag and drop)	6
Desarrollo: JAVASCRIPT (otros métodos)	9
Conclusiones	11
Bibliografía/Webgrafía	12

Introducción

A la hora de comenzar el proyecto decidimos examinar las partes de este con antelación y preparar un servidor de respaldo con una estructura predeterminada donde subiríamos todo el proceso que realizamos a lo largo del mes que teníamos de plazo.

Esto se traduce a un repositorio de github en el que todos tendríamos permisos para subir nuestro proyecto a la rama develop y después de asegurarnos de su correcto funcionamiento el encargado del repositorio podría realizar un merge entre main y develop puesto que nadie salvo propietario del repositorio podía editar main como medida de seguridad.

Al subir los cambios en la rama main se envía una actualización a un servidor de respaldo en vercel donde la aplicación es lanzada, permitiendo el acceso a esta en cualquier momento sin necesidad de ejecutar en local.


Link a github:

<https://github.com/Ivan1676/Solitario.git>

Después de decidir la organización del proyecto nuestro siguiente paso fue realizar un estudio detenido del código proporcionado, sobre todo de los elementos html ya que debíamos comprender su estructura a la hora de relacionarlos con el css y el Javascript. Cuando ya teníamos una idea relativamente firme de los componentes del proyecto decidimos comenzar con los pasos más simples de Javascript, como por ejemplo el contador del tiempo.

Al principio surgieron una serie de problemas debido a que al tener un tipo de fuente especial y por cómo le dimos los márgenes a los contadores los span se desplazaban en función del tamaño que tuviera el contador de tiempo. Al final conseguimos que este no diera ningún problema de desplazamiento y lo incluimos todo en una navbar que creamos con antelación para dar lugar a una nueva interfaz de aspecto visual más llamativo.

Para proseguir decidimos centrarnos en el desarrollo de los tapetes basándonos en el código previamente proporcionado. Al terminar con esto comenzó la ardua tarea de instanciar las cartas y hacerlas aparecer en el mazo inicial como hijas del parent. Poco después de hacerlas aparecer se nos presentó el problema de mostrar las cartas en diagonal lo cual no supuso una cantidad de trabajo y esfuerzo ingente pero aun así consiguió retrasarnos durante un tiempo. Con las cartas finalmente en diagonal era el momento de pasar al siguiente paso.



El siguiente paso consistirá en hacer que las cartas puedan ser seleccionadas y arrastradas para su posterior introducción en los tapetes designados. Esto fue posible por medio de los métodos `drop`, `drag` y `allowDrop` que definimos posteriormente para este propósito. Por supuesto se debe dar a las cartas el atributo `draggable true` para que esto funcione correctamente.

Con las cartas ya disponibles y su función `drag` habilitada el siguiente paso fue hacer el contador de movimientos y la posibilidad de colocar las cartas en los tapetes. Las reglas de estos serían implementadas más adelante.

La idea del contador de movimientos se basa en que cada vez que una carta sea colocada en un tapete este debe incrementarse en uno. De un modo distinto al empleado para el contador de movimientos conseguimos que los contadores de los tapetes nos indiquen el número de cartas las cuales pertenecen a este. Esto se conseguía por medio de arrays.

Acercándonos al desenlace de nuestro proyecto el siguiente paso que teníamos que realizar sería uno de los más complicados, las reglas. Estas especifican que en los tapetes receptores sólo pueden incluirse determinados números en función de si ya había uno antes o no o en función al color del número anterior ya que no pueden ponerse dos cartas de un mismo color seguidas.

Con las reglas también implementamos la opción de reiniciar la partida y de rellenar el tapete inicial con las cartas de sobrantes en caso de que todas las cartas estuvieran colocadas en los tapetes siempre y cuando la partida no esté finalizada. Al pulsar reiniciar volvemos a barajar y comenzamos de cero en todos los contadores.

Como último paso realizamos la implementación de efectos de sonido a la hora de barajar, introducir una carta de forma errónea o ganar. Ganar también incluye dos gifs los cuales serán eliminados junto con la música si decidimos reiniciar la partida.

Con esto nos gustaría finalizar la introducción y dar lugar al siguiente paso de nuestra documentación en el que hablaremos de una forma más extensa y detallada sobre la parte del código del proyecto para proporcionar un entendimiento o con un enfoque lógico de este. Sin más preámbulos pasemos pues al desarrollo de nuestro proyecto basado en el solitario.

Desarrollo: HTML

La página se compone de una serie de elementos los cuales describiremos a continuación.

La navbar hecha con bootstrap actúa como el encabezado de la página, en la que podemos encontrar el logo del centro cargado por medio de un elemento `img`, el título con tamaño `h1` y dos `span` en los que introducimos el tiempo y los movimientos que llevamos según se va desarrollando el juego.

El resto de elementos son los que componen la mesa (el `div`). Estos se basan en una serie de `divs` con unos `span` para los contadores. Los elementos son inicial que será el recuadro donde aparecen las cartas al comenzar, los receptores donde poner las cartas cuando se pueda y sobrantes para dejar las cartas. Para terminar contamos con un botón de reiniciar.

Desarrollo: CSS

En el `css` definimos los estilos básicos que queremos que tenga la página, como por ejemplo la foto de la navbar y la imagen de fondo por medio de las propiedades `background-image`. También definimos los tamaños de los elementos en `em`, la propiedad `z-index` para que la navbar se superponga en caso de que sea necesario y gran cantidad de posiciones absolutas y relativas para la distribución de los elementos de la página.

La mayoría de los márgenes están hechos con `%` y con `left` o `right` sin poner `margin-left` o `right` debido a que en algunos casos como el contador de tiempo al intentar mantener siempre el mismo margen realizaba desplazamientos en el resto de elementos según aumentaba el tiempo. Esto es usado también en los contadores de los tapetes.

Para el tema de centrado además de la posición relativa o absoluta utilizamos también `transform: translate` la cual permite centrar los elementos sin que queden más desplazados de lo que deseamos (por ejemplo cuando le pides a un elemento que se coloque en medio de la página pero está en una posición que supera el 50%).

La fuente utilizada es la más adecuada para cada usuario por medio de la propiedad `font-family: system-ui`, la cual aplica la fuente recomendada para el sistema que la ejecuta.

Desarrollo: JAVASCRIPT (declaración de variables)

Una vez puestos dentro del código lo primero era implementar las variables que íbamos a utilizar, algunas ya venían implementadas en el propio enunciado base pero eso no quita que no tuviéramos que acostumbrarnos a ellas, debíamos entender cómo usarlas y cuál era su función. Veamos algunas de ellas:

Los arrays “palos” y “numeros” eran necesarios para crear el mazo de cartas inicial, ya que combinando un elemento de cada uno se consigue el nombre de la imagen a la cual hay que llamar. Ejemplo: 1-cua.png, 7-ova.png, etc.

```
let palos = ["ova", "cua", "hex", "cir"];  
let numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12];
```

Las variables “paso” y “tapete_origen” serían variables que usaremos más adelante en los distintos métodos de la tarea.

En el caso de la variable “paso” se usaría para incrementar la posición de las cartas conforme iban siendo mostradas en pantalla, de este modo daban un aspecto de estar extendidas en la mesa de forma diagonal, esta se incrementa cada vez que muestre una carta por pantalla y se reinicia cada vez que se muestre un nuevo tapete.

La variable “tapete_origen” simplemente guarda el tapete de donde proviene la carta que coja el usuario, solo se pueden dar dos casos, que provenga del tapete inicial o del sobrantes, esto se tendrá en cuenta a la hora de poder coger las cartas de nuevo desde el tapete sobrantes.

```
let paso = 1;  
let tapete_origen;
```

Los audios se implementan como variables para poder reproducirlos más tarde, simplemente indicamos su ruta en su valor.

Y por último otras variables que venían con el propio enunciado como los arrays de cartas de cada tapete, las variables que usa el temporizador y elementos del propio html como el contador de cartas, movimientos y tiempo y los propios tapetes donde próximamente soltaremos y recogeremos nuestra baraja.

Una vez listadas todas las variables, podemos comenzar a explicar el código.

Desarrollo: JAVASCRIPT (drag and drop)

Con el propósito de poder arrastrar las cartas de un tapete a otro, hubo que aplicar la técnica del “Drag and Drop”. Para ello se crearon varias funciones teniendo en cuenta que el elemento se pudiera agarrar, mover y soltar.

La primera que aparece en el código es “allowDrop()”. Su cometido es permitir que una carta se disponga sobre cualquier tapete. A esta función se le pasa el evento “ondragover”. Dicho evento ocurre cada vez que un elemento, posibilitado en el documento HTML para ser desplazado, se arrastra hasta otro elemento objetivo. Para que esto sea posible, la acción por defecto del navegador es bloqueada con el “preventDefault()”.

```
function allowDrop(ev) {  
    ev.preventDefault();  
}
```

En segundo lugar, nos encontramos con la función “drag()”. Su cometido es recoger el identificador en formato texto del elemento arrastrado. Al objeto del evento se le asigna dicha información a través del “setData()”. Por otra parte, en lo concerniente al evento proporcionado, en este caso es el: “ondragstart”. Este tiene lugar en el momento en el que se comienza a desplazar la carta.

En esta función es también donde hacemos uso de la variable antes comentada “tapete_origen”, ya que aquí es donde la rellenamos con el id del padre del elemento que está siendo agarrado, en este caso, una carta.

```
function drag(ev) {  
    ev.dataTransfer.setData("text", ev.target.id);  
    var data = ev.dataTransfer.getData("text");  
    tapete_origen = document.getElementById(  
        document.getElementById(data).parentElement.id  
    );  
}
```

Por último comentaremos la función “drop()”, el cual se inicia cada vez que una carta es depositada en un tapete, aquí es donde comienza el juego del solitario ya que dentro de esta función es donde se implementan las normas del solitario y todas las reglas que permiten que la siguiente carta pueda cogerse después de depositar otra.

Lo primero que indicaremos a parte de las funciones generales del drag and drop es este fragmento, el cual indica el tapete donde se tiene que introducir la carta aun teniendo una carta delante.

```
function drop(ev) {  
    ev.preventDefault();  
    var data = ev.dataTransfer.getData("text");  
    var carta = document.getElementById(data);  
    var movimientoValido = false;  
    var target_;  
    if (  
        document.getElementById(ev.target.id).parentElement.id ==  
        "padre_tapetes"  
    ) {  
        target_ = ev.target.id;  
    } else {  
        target_ = document.getElementById(ev.target.id).parentElement.id;  
    }  
    var tapeteSeleccionado = document.getElementById(target_);
```

Este código nos indica que si el padre del contenedor no es el div donde se encuentran todos los tapetes pequeños significa que está dentro de una carta, así que su padre será el tapete donde deba ir.

Sin este código una carta se introduciría dentro de otra dejándola inaccesible para seguir jugando con esa carta.

Principalmente para evitar esta situación hicimos uso de la función "notAllowDrop()" junto con la función "stopPropagation()", la cual resuelve que el evento no se extienda a otros elementos descendientes. Ya que conseguimos solucionar el problema a medida que avanzábamos ambas funciones fueron eliminadas.

Lo siguiente que necesitábamos era reservar el tapete al cual va dirigida la carta, el cual sacamos y guardamos mediante un switch.

Si la carta entra en el tapete sobrantes el movimiento sería automáticamente válido ya que no se le aplicaría ninguna regla del juego. En caso contrario comenzamos a aplicar las reglas del solitario propuesto, que consisten en ir alternando cartas entre naranja y gris y en una escala descendente desde el 12 hasta el 1.

Una vez determinado si el movimiento es válido insertamos la carta en el tapete indicado e incrementamos el contador de movimientos.

```
if (movimientoValido == true) {
    tapeteSeleccionado.appendChild(document.getElementById(data));
    carta.removeAttribute("style");
    if (tapeteSeleccionado != tapete_sobrantes) {
        carta.setAttribute("draggable", "false");
    }
    switch (tapete_origen.id) {
        case "inicial":
            tapete_inicial.lastChild.setAttribute("draggable", "true");
            mazo_inicial.pop();
            break;
        case "sobrantes":
            if (tapete_sobrantes.childElementCount != 1) {
                tapete_sobrantes.lastChild.setAttribute("draggable", "true");
            }
            mazo_sobrantes.pop();
            break;
    }
    mazoSeleccionado.push(carta.id);
    inc_contador(cont_movimientos);
}
```

Aquí es donde indicaremos cual es la siguiente carta que se puede coger a continuación ya que deberá ser la anterior dependiendo de si la carta viene del tapete inicial o del tapete sobrantes, esto es importante destacar para saber a qué elemento hay que cambiarle el atributo “draggable”, que indica si la carta puede arrastrarse o no.

Por último dentro de el drop también detectaremos el momento en que nos quedamos sin cartas para automáticamente recargar de nuevo el tapete con las cartas desordenadas del tapete sobrantes.

Desarrollo: JAVASCRIPT (otros métodos)

En lo que respecta a otras funciones empleadas ajenas a la técnica del “drag and drop”, tuvimos que tener en cuenta el hecho de que en varias situaciones del juego, todos los tapetes exceptuando el tapete donde aparece la baraja deben quedar vacíos. No solo cuando el jugador quiere reiniciar la partida de cero, sino también cuando no quedan cartas en el tapete inicial y, al no haber acabado el juego, se desea continuar con todas las cartas del tapete sobrantes colocadas en el inicial.

Por este motivo, en el supuesto de que se desee reiniciar el juego y hubiera cartas ya situadas en alguno de los tapetes; se requeriría eliminar estas cartas de su posición para volverlas a mostrar en el tapete inicial.

Esta es la intención de la función “limpiar”. Comprobará el número de cartas incluidas en un contenedor y para limpiar el tapete, es importante no borrar todos los elementos del tapete ya que podríamos eliminar el span con el contador de cartas, por ello nos ayudamos de la función “childNodes” para asegurarnos que borre todos los elementos menos el primero.

```
while (tapete_inicial.childNodes[2]) {  
    tapete_inicial.removeChild(tapete_inicial.childNodes[2]);  
}
```

Inicialmente esto se pensaba hacer con innerHTML pero los contadores se sobrescribían todo el rato a cero así que optamos por usar mejor esta opción.

Asimismo, otros de los encargos era el de llevar el número de cartas dispuestas sobre cada tapete. Para ello, se crearon las funciones de incrementar, decrementar y establecer el valor de los contadores.

Primero creamos un método que provocara que el contador fuera incrementándose de uno en uno. Luego otro que hiciera justo lo opuesto: que descontase también de uno en uno. A continuación, un método que estableciera el contador al valor que le corresponde. Y por último, un método que actualizase todos los contadores.

En el documento HTML ya se iniciaron los contadores a lo que se tenían que ajustar. Con el “innerHTML” se logra imprimir por pantalla el resultado de las operaciones. A la cifra existente ya como contador se le suma o se le resta el resultado de la cifra del contador más uno. En el caso donde se restablece el contador, se le asignará el valor: bien de 48 en el tapete inicial, bien de 0 en el resto de tapetes.

A parte añadimos las funciones de fallo y victoria, la primera recordaría al usuario las normas del solitario y la segunda indicará por pantalla el tiempo y los movimientos finales, con algún que otro añadido tanto sonoro como visual.

```
function hasGanado() {
    victoria.play();
    tapete_inicial.setAttribute(
        "style",
        "background-image: url('./imgs/gifs/chad.gif'); background-position:
center 25%; background-size: cover; "
    );
    tapete_sobrantes.setAttribute(
        "style",
        "background-image: url('./imgs/gifs/pingu.gif'); background-position:
center 25%; background-size: cover; "
    );
    alert(
        ";;;Enhorabuena!!!\nHas ganado la partida :D\n\nTiempo: " +
        cont_tiempo.textContent +
        "\nMovimientos: " +
        cont_movimientos.textContent +
        "\n\nPulsa el botón de Reiniciar para jugar otra partida :)"
    );
}
```

Conclusiones

A lo largo de todo el proceso realizado, se concluye que es primordial que exista una organización y un orden a la hora de establecer prioridades en cuanto a realizar un método de desarrollo. De igual manera, es importante conocer en qué funcionalidades se debe progresar primero y cómo.

Se comienza por las tareas un poco más sencillas y superficiales, pues se aprecian nada más cargar la página y se puede saber exactamente qué es lo que se ejecutó mal. Además sirven para tener en cuenta cómo se sitúa cada elemento HTML en el diseño de la página; después atendiendo a esto, se va ajustando a lo que requiera el código. Un ejemplo de estas tareas son los contadores: los de tiempo y los del número de cartas y movimientos.

Se continúa con funcionalidades más difíciles y, a pesar de que a veces también los errores se pueden observar al cargar la página, hay que ir más al fondo del código para saber específicamente por qué se ha causado un conflicto. Algunas de estas funcionalidades son: el método por el cual se muestran las cartas en el tapete inicial o la técnica del “drag and drop”.

Finalmente, se termina con aquello que no es tan imprescindible y también con los pequeños errores visuales. Los efectos de sonido se añaden una vez que nos hemos asegurado de que el juego es funcional completamente. También se pulen detalles visuales como los contadores o el logo.

Para concluir, durante todo este proceso de desarrollo se desechan muchas ideas y funciones por diferentes motivos: son demasiado arduas de implementar y posteriormente se encuentran otras que son más sencillas; por seguir una línea del concepto diferente al que se había planteado en un brainstorming; o simplemente porque se van probando opciones hasta que se encuentra una que no es tan problemática; o porque hay funcionalidades que se adaptan o se engloban en otras por optimización o practicidad.

Bibliografía/Webgrafía

MDN(28 de noviembre de 2022).*Drag and Drop* archivo.https://developer.mozilla.org/es/docs/Web/API/HTML_Drag_and_Drop_API/File_drag_and_drop. Recuperado el 23 de febrero de 2023.

MDN(20 de febrero de 2023).*Event.stopPropagation()*.<https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation>. Recuperado el 23 de febrero de 2023.

MDN(20 de febrero de 2023).*DataTransfer.setData()*.<https://developer.mozilla.org/en-US/docs/Web/API/DataTransfer/setData>. Recuperado el 23 de febrero de 2023.

MDN(12 de diciembre de 2022).*String.prototype.split()*.https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/split. Recuperado el 22 de febrero de 2023.

MDN(20 de febrero de 2023).*Node.childNodes*.<https://developer.mozilla.org/es/docs/Web/API/Node/childNodes>. Recuperado el 22 de febrero de 2023.

mrGcoding(29 de abril de 2020).*JavaScript - Randomize Items In An Array*.<https://www.youtube.com/watch?v=seApG3uwjAs>. Recuperado el 22 de febrero de 2023.

Programando el Destino(8 de abril de 2020).*Curso Javascript - DRAG and DROP*.<https://www.youtube.com/watch?v=UAKCVwhzaG4>. Recuperado el 22 de febrero de 2023.

Programando el Destino(8 de abril de 2020).*Curso Javascript - DRAG and DROP(2)*.<https://www.youtube.com/watch?v=wP-yu5cDtNc>. Recuperado el 22 de febrero de 2023.

SoundJay(s.f.).*Card Shuffle Sounds*.<https://www.soundjay.com/card-sounds-1.html>. Recuperado el 22 de febrero de 2023.

W3Schools(s.f.).*HTML Drag and drop*.https://www.w3schools.com/html/html5_draganddrop.asp. Recuperado el 22 de febrero de 2023.

W3Schools(s.f.).*HTML DOM Element childElementCount*.https://www.w3schools.com/jsref/prop_element_childelementcount.asp. Recuperado el 22 de febrero de 2023.



W3Schools(s.f.).*Ondragover*

event.https://www.w3schools.com/jsref/event_ondragover.asp. Recuperado el 23 de febrero de 2023.