Universidad Internacional de Valencia

Solitario

Actividad 1

Rafael Calixto Ruas Fragozo

Ingrid Mayerly Bustos Robayo

Ángel López Arias

Máster Universitario en Desarrollo de Aplicaciones y Servicios Web

Lado del Cliente (Front-End y multimedia)

12/02/2023

Resumen

En este documento se recoge la implementación de la primera actividad de la asignatura Lado del Cliente Front-End.

Inicialmente se comenta la estructura aportada por el docente, para el comienzo del desarrollo de la actividad. Tras esto, se incluyen planteamientos y cambios en la estructura, que se llevaron a cabo para poder trabajar más cómodamente. Por último, se aportan comentarios y fragmentos de código de las partes más relevantes del código final.

En la parte final, se incluyen unas conclusiones sobre el trabajo realizado.

Índice

1. Introducción

La primera actividad de esta asignatura consiste en la construcción de una versión del juego del solitario (juego de cartas) utilizando una página web en html, un fichero de estilos y un script de JavaScript, donde se pretende alojar toda la lógica de este.

El objetivo principal del desarrollo del juego es practicar y demostrar los conocimientos adquiridos sobre el lenguaje de programación JavaScript, así como sus posibles usos en el ámbito de la programación web en el lado del cliente.

2. Base inicial

Como base para comenzar el proyecto, se han proporcionado una serie de archivos:

- Una hoja de estilos
- Un fichero solitario.html
- Un fichero solitario.js
- Una carpeta con 48 imágenes de cartas de una baraja

En el fichero HTML se encuentra una vista en la que ya están creados algunos elementos del escenario del juego, como lo son el contador de tiempo, el contador de movimientos, los tapetes en los que habrá que situar las cartas y el botón de reinicio del juego. En la hoja de estilos, se incluyen los colores de los tapetes y los estilos que tiene la vista principal.

El archivo JavaScript contiene una estructura básica que aporta la columna vertebral, alrededor de la cual, se puede construir la lógica restante para completar el juego. Se encuentran declarados todos los elementos ya creados en el documento HTML como variables discretas, una función *arrancarTiempo*, que pone en marcha el temporizador y un conjunto de funciones planteadas como *comenzarJuego*, *barajar*, *cargarTapeteInicial*, *incContador*, *decContador*, *setContador.*

Todo esto conforma la base sobre la que se ha desarrollado la actividad.

3. Cambios en la estructura

La implementación de la lógica del juego del solitario se ha desarrollado durante dos semanas, de forma intermitente. A medida que se iba avanzando en el desarrollo, se planteó un cambio de estructura frente a la planteada inicialmente en el material aportado: cambiar la forma que se tenía de interactuar con los tapetes, sus mazos y sus contadores.

Inicialmente, se tenía una variable por cada tapete, mazo y contador, haciendo necesaria la escritura manual de estos objetos cada vez que se quisiera modificar alguno de ellos o cambiar alguna de sus propiedades.

```
let tapeteInicial    = document.getElementById("inicial");
let tapeteSobrantes  = document.getElementById("sobrantes");
let tapeteReceptor1  = document.getElementById("receptor1");

let mazoInicial    = [];
let mazoSobrantes  = [];
let mazoReceptor1  = [];

let contInicial     = document.getElementById("contador_inicial");
let contSobrantes   = document.getElementById("contador_sobrantes");
let contReceptor1   = document.getElementById("contador_receptor1");
```

Fragmento Código 1: Variables iniciales.

Para poder acceder de forma intuitiva a cada uno de los mazos, contadores y tapetes del juego, sin tener que escribir los nombres de las variables cada vez que se quería acceder a ellos, se cambió lo anteriormente mencionado por un array de diccionarios, que permitiese tener todos los tapetes reunidos bajo una variable y, además, permitiese relacionar cada mazo con su tapete y su contador.

```
const tapetesMazos = [
    {
        id: "inicial",
        tapete: document.getElementById("inicial"),
        mazo: [],
        contador: document.getElementById("contador_inicial")
    },
    {
        id: "sobrantes",
        tapete: document.getElementById("sobrantes"),
        mazo: [],
        contador: document.getElementById("contador_sobrantes")
    },
    {
        id: "receptor1",
        tapete: document.getElementById("receptor1"),
        mazo: [],
        contador: document.getElementById("contador_receptor1")
    },
    ...];
```

Fragmento Código 2: Array con objetos tapetes.

De esta forma, se puede acceder a todos los tapetes, si se desea ejecutar una función sobre todos ellos, solamente iterando el array **tapetesMazos.** Para acceder a un tapete en particular, se construyó una función que devuelve el objeto deseado, recibiendo como parámetro el id del mismo.

```
function getTapeteObject(idTapete) {
	return tapetesMazos.find((tapete) => tapete.id === idTapete);
}
```

Fragmento Código 3: Función para devolver el objeto tapete deseado.

Como cambios nimios, se puede considerar la construcción de numerosas funciones más, a parte de las incluidas en el planteamiento inicial, a fin de que el código permanezca limpio y ordenado.

4. Implementación final

A continuación se comentan algunas de las funciones más relevantes que se han implementado en el desarrollo de la lógica del juego requerido. Cabe mencionar que, en el código fuente, cada método tiene un pequeño comentario que indica para qué se utiliza.

Al final del archivo solitario.js se encuentra una pequeña línea de código que hace que todo el juego comience, nada más se abre la página.

```
document.body.addEventListener('onload', comenzarJuego());
```

Fragmento Código 4: Iniciador del juego

En ella, se configura el body del documento, para que cuando se cargue por completo, se ejecute la función *cargarJuego().* A continuación, esta función, se encarga de preparar toda la lógica para comenzar.

```
function comenzarJuego() {
	botonReset.disabled = true;
	ponerACeroTapetes();
	setContador(contMovimientos, 0);

	cargarBaraja();
	configurarTapetes();

	const tapeteInicial = getTapeteObject('inicial');
	tapeteInicial.mazo = barajar(tapeteInicial.mazo);
	cargarTapeteInicial();

	arrancarTiempo();
}
```

Fragmento Código 5: Función comenzarJuego.

Lo primero que se hace es desactivar el botón de reinicio. Esto es importante ya que se le ha dado cierto efecto asíncrono a la carga de las cartas en el tapete, para que parezca que van apareciendo dinámicamente una sobre otra. Si se permitiese el uso del botón de reinicio mientras se cargan las cartas en el tapete, se mezclarían mazos y no funcionaría.

El siguiente paso es poner a cero todos los tapetes, dejando sus mazos como arrays vacíos y eliminando todos los posibles hijos que puedan tener sus componentes en el HTML, evitando borrar los componentes "contadores". Para esto se hace uso de la función *vaciarTapete()*.

```
function vaciarTapete(tapeteObjeto) {
    while (tapeteObjeto.tapete.firstChild &&
                !tapeteObjeto.tapete.lastChild.id.includes('contador')) {
        tapeteObjeto.tapete.removeChild(tapeteObjeto.tapete.lastChild);
    }
    tapeteObjeto.mazo = [];
    setContador(tapeteObjeto.contador, 0);
}
```

Fragmento Código 6: Función vaciarTapete.

Para continuar, se pone el contador de movimientos a cero. Después se crea, por cada carta de las 48 que se aportan, un objeto *<img>* con las propiedades necesarias y los datos que se requerirán posteriormente para las comprobaciones de movimientos, y se meten en el mazo del tapete inicial.

```
function cargarBaraja() {
    for (let palo = 0; palo < palos.length; palo++) {
        for (let numero = 0; numero < numeros.length; numero++) {
            const temporalImage = crearCarta(numeros[numero], palos[palo]);
            getTapeteObject('inicial').mazo.push(temporalImage);
        }
    }
}

function crearCarta(numero, palo) {
    const temporalImage = document.createElement("img");
    temporalImage.src = "./imagenes/baraja/" + numero + "-" + palo + ".png";
    temporalImage.width = 75;
    temporalImage.height = 100;
    temporalImage.id = palo + numero;
    temporalImage.setAttribute("data-palo", palo);
    temporalImage.setAttribute("data-numero", numero);
    temporalImage.setAttribute("data-tapete", "inicial");
    return temporalImage;
}
```

Fragmento Código 7: Funciones cargarBaraja y crearCarta.

Tras esto, se configuran los tapetes receptores y el tapete de cartas sobrantes, con la lógica necesaria para aceptar los eventos **drag&drop** necesarios para que puedan recibir cartas en ellos. Para todos ellos, se setean los eventos *ondragenter*, *ondragover* y *ondragleave* de la misma forma:

```
function (e) { e.preventDefault(); };
```

Sin embargo, el evento *ondrop* se configura de diferente manera para cada uno de los tipos de tapetes:

- Tapete de cartas sobrantes

  Para este tapete, se configura una función que permite el movimiento de cartas desde el tapete inicial, sin ninguna restricción. Simplemente se llama a la función *moverCartaTapete*.

- Tapetes receptores

  Estos tapetes requieren un poco más de control. Solo se permite el movimiento a estos tapetes si la carta proviene de los tapetes inicial o sobrantes y además se valida el movimiento. Es entonces cuando se llama a la función *moverCartaTapete*.

```
function moverCartaTapete(carta, origen, destino) {
	origen.tapete.removeChild(carta);
	decContador(origen.contador);
	origen.mazo.pop();

	carta.style.top = "50%";
	carta.style.left = "50%";
	carta.style.transform = "translate(-50%, -50%)";
	carta.dataset["tapete"] = destino.id;

	destino.tapete.appendChild(carta);
	incContador(destino.contador);
	destino.mazo.push(carta);

	incContador(contMovimientos);
	const tapeteInicial = getTapeteObject('inicial');
	if (origen.id === 'inicial') {
		if (tapeteInicial.mazo.length === 0) {
			moverCartasSobranteAInicial();
		} else {
			tapeteInicial.mazo[
				tapeteInicial.mazo.length - 1
			].draggable = true;
		}
	}
}
```

Fragmento Código 8: Función moverCartaTapete.

Si, cuando se está moviendo una carta desde el tapete inicial, este se queda vacío, automáticamente se rellena con las cartas que estén en el montón de sobrantes barajadas de nuevo, vaciándose el tapete de sobrantes para quedar de nuevo vacío.

El penúltimo paso antes de empezar el juego, es barajar el mazo con las 48 cartas, para lograr que estén en orden aleatorio y no secuencial. Para ello, al principio se optó por la implementación de una función *sort* que ordenase el array que contenía las cartas en función de un número aleatorio.

```
const mazoBarajado = mazo.sort(() => Math.random() - 0.5);
```

Fragmento Código 9: Función ordenación aleatoria.

Sin embargo, dado que el número de elementos dentro del array es algo elevado, y que muchas veces, en repetidas pruebas, no se conseguía la aleatoriedad buscada, se optó por la implementación del algoritmo de Fisher-Yates [1]. Este método se basa en permutaciones y consigue desordenar de forma efectiva un conjunto de objetos dado.

```
function barajar(mazo) {
	for (var carta = mazo.length - 1; carta > 0; carta--) {
		var cartaCambio = Math.floor(
			Math.random() * (carta + 1));
		[mazo[carta], mazo[cartaCambio]] =
				[mazo[cartaCambio], mazo[carta]];
	}
	setContador(getTapeteObject('inicial').contador, mazo.length);
	return mazo;
}
```

Fragmento Código 10: Función barajar.

El último paso es ejecutar la función *cargarTapeteInicial* para después arrancar el contador de tiempo y comenzar el juego. Dicha función, posee una particularidad: coge cada una de las cartas del mazo del tapete inicial y las va añadiendo como nodos al componente "tapeteInicial" de forma asíncrona, y con una diferencia de tiempo de 50ms. De esta forma, se consigue un efecto visual que se asemeja al que se puede apreciar cuando se esparce una baraja sobre un tapete.

```
function cargarTapeteInicial() {
	const inicial = getTapeteObject('inicial');
	for (let carta = 0; carta < inicial.mazo.length; carta++) {
		inicial.mazo[carta].dataset["tapete"] = "inicial";
		setTimeout(cargarCarta, 50 * carta, inicial.mazo[carta],
				carta, inicial.mazo.length);
	}
}
```

Fragmento Código 11: Función cargarTapeteInicial.

Cuando el juego termina de forma exitosa, se muestra por pantalla una alerta informando del tiempo que se ha tardado y los movimientos que se han llevado a cabo para conseguirlo. El juego permanecerá en pausa hasta que se pulse sobre el botón de reinicio.

5. Conclusión

Esta práctica nos ha permitido explorar un poco las posibilidades que ofrece JavaScript en el desarrollo web en el lado del cliente. Nos ha resultado muy versátil y útil combinarlo con HTML, ya que hemos podido puede alterar la estructura del documento, añadir nuevos elementos, mover los nodos de sitio y añadir lógica, sincronía, asincronía y memoria física. Hemos disfrutado realizando esta actividad.

6. Bibliografía y Referencias

a. Recursos de búsqueda de información

i. [1] Algoritmo Fisher-Yates:

https://www.delftstack.com/es/howto/javascript/shuffle-array-javascript/

b. Fragmentos Código

(Todos extraídos del fichero solitario.js, en su versión inicial y en su versión final)

i. Frag.Cod. 1: Variables iniciales
ii. Frag.Cod. 2: Array con objetos tapetes
iii. Frag.Cod. 3: Función para devolver el objeto tapete deseado
iv. Frag.Cod. 4: Iniciador del juego
v. Frag.Cod. 5: Función comenzarJuego
vi. Frag.Cod. 6: Función vaciarTapete
vii. Frag.Cod. 7: Funciones cargarBaraja y crearCarta
viii. Frag.Cod. 8: Función moverCartaTapete
ix. Frag.Cod. 9: Función ordenación aleatoria
x. Frag.Cod. 10: Función barajar
xi. Frag.Cod. 11: Función cargarTapeteInicial