



Universidad Internacional de Valencia

# Juego del solitario creado con HTML, CSS y Javascript

**Titulación:**
Máster Universitario en Desarrollo de Aplicaciones y Servicios Web

**Curso académico:** 2022-2023

**Alumno/a:** Gómez Olivencia, Rubén
**D.N.I.:** 78910013-A

**Asignatura:** Desarrollo de aplicaciones web II: lado del cliente (front-end) y multimedia

# Índice general

# 1. Introducción

A lo largo de este documento se van a explicar las decisiones tomadas, tanto en el ámbito de programación como de diseño, durante el desarrollo de un juego basado en el popular juego de cartas “el solitario”.

Para la realización de este juego se ha hecho uso del lenguaje de marcado de hipertexto **HTML**, junto con el sistema de hojas de estilo en cascada **CSS** y el lenguaje de programación **Javascript**.

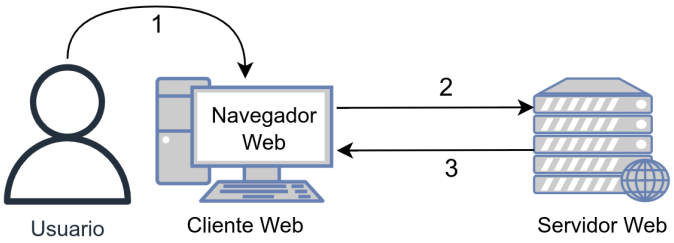
# 2. Arquitectura cliente-servidor

A la hora de realizar una aplicación web, o como en este caso un juego, tenemos que tener en cuenta cómo funciona la arquitectura que hace posible que podamos interactuar con él.

Cuando navegamos por internet se hace uso de la conocida como arquitectura **Cliente - Servidor**, en la que podemos diferenciar, como su nombre indica, dos apartados:

- **Servidor**: Un servidor es un ordenador potente que recibe peticiones, las procesa y devuelve una respuesta dependiendo de la petición recibida. En el caso de una aplicación web, el servidor contará con un servicio que recibirá peticiones **HTTP** y responderá con respuestas en código HTML, imágenes, ficheros ... Entre los servidores web más conocidos están: Apache HTTP Server, Nginx, IIS, ...
- **Cliente**: A la hora de interactuar con una aplicación web el cliente es quien va a realizar las peticiones al servidor. Normalmente el cliente es un navegador web utilizado por un usuario. Entre los navegadores web más conocidos están: Firefox, Chrome, Edge, ...

A la hora de interactuar, un usuario hará uso del cliente (navegador web), con el que interactuará para realizar, normalmente, peticiones al servidor en busca de información. A continuación se puede ver un dibujo de cómo funciona la interacción descrita:




Tal como se puede ver en la imagen, los pasos que se realizan son:

1. **Usuario interactúa con navegador web**. Tal como hemos dicho, un navegador web es el cliente de nuestra arquitectura, que es utilizado por el usuario. Cuando un usuario accede a una web, hace clic en un enlace o realiza una búsqueda, está comenzando el proceso del siguiente paso.
2. **Realizar petición**. El navegador detecta lo que el usuario quiere realizar, y en ese momento lanza una petición (o varias) pidiendo al servidor web lo solicitado (una nueva web, una búsqueda, ...).
3. **Devolver respuesta**: El servidor web recibe la petición, la procesa, y devuelve el resultado al cliente (navegador web) que mostrará por pantalla.

## 2.1. Back-end vs. Front-end

En la arquitectura mostrada previamente, a la hora de diseñar el software, entran en juego también dos partes, que se encargarán por separado de procesar la entrada y realizar una salida:

- El ***back-end*** es el software que se ejecuta en la parte de **servidor**, que recibe las peticiones del cliente, las procesa (gestión de autenticación, acceder a base de datos, realizar cálculos, obtener información...) y que posteriormente el servidor devolverá.

  En este apartado de ***back-end*** se hace uso de lenguajes de programación como PHP o Java, frameworks como Ruby on Rails... , por poner sólo unos ejemplos.

- El ***front-end*** es la parte que se ejecuta en la parte **cliente** (navegador web) de la arquitectura anterior. Mostrará la información atendiendo al diseño recibido, teniendo en cuenta la información HTML y las hojas de estilos CSS.

También existe la posibilidad de ejecutar código que alterará, o que nos permitirá interactuar con la web. Este código está escrito en lenguaje **javascript** y que se ejecuta enteramente en el lado cliente (navegador web).

El juego realizado, que se va a detallar a continuación, ha sido realizado para que pueda ser ejecutado enteramente en la parte ***frontend***. Esto hace que no sea necesario un servidor con el que interactuar, y por tanto sólo el navegador web ejecutará todas las funciones necesarias para poder jugar.

# 3. Normas del juego

Aunque el juego del solitario es conocido, a continuación se va a detallar brevemente las normas del juego, ya que es importante para entender el desarrollo realizado.

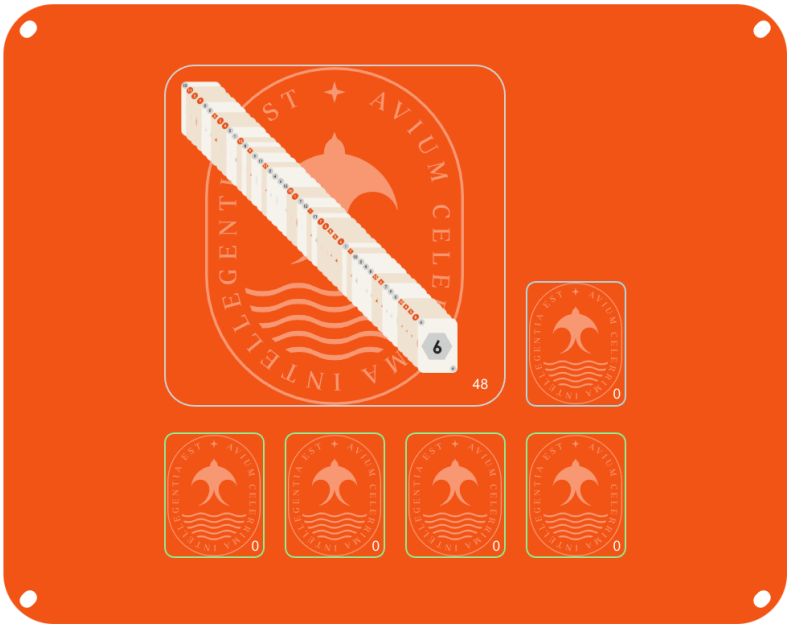
Las reglas del juego son las siguientes:

- La baraja de cartas consta de:
  - 4 palos nombrados como “viu”,”cuadrados”, “hexágonos” y “círculos”. Los dos primeros tienen color naranja y los dos últimos color gris.
  - Cada palo tiene 12 cartas numeradas del 1 al 12.
- La mesa de juego contará con seis tapetes (donde se colocarán mazos de cartas).
- Los tapetes tienen las siguientes funciones:
  - **Tapete inicial**: donde se sitúa el mazo completo ya barajado al inicio de la partida. Este tapete nunca podrá recibir cartas mientras haya cartas en él.
  - **4 tapetes receptores**: donde se sitúan las cartas en orden decreciente (empezando siempre por una carta con el número 12) y alternando los colores. No se puede saltar ningún número y nunca podrán ir dos cartas del mismo color seguidas.
  - **Tapete sobrantes/descartes**: donde se pondrán las cartas de manera temporal desde el tapete inicial ya que no es posible ponerlas sobre ningún tapete receptor.

Es posible mover una carta de este tapete a uno receptor en cualquier momento.

- Cuando no queden cartas en el tapete inicial, se cogerán las cartas del tapete sobrantes, se barajarán y se pondrán sobre el tapete inicial.
- El juego termina cuando no haya ninguna carta sobre el tapete inicial ni el sobrantes.

La mesa de juego tiene la siguiente forma, donde se pueden ver los distintos tapetes: en la fila superior el inicial (más grande) y el de cartas sobrantes y abajo los cuatro de recepción:



# 4. Desarrollo realizado

Teniendo en cuenta las reglas previas, se ha tenido que realizar el desarrollo siguiendo dichos criterios. En este apartado no se va a detallar el aspecto visual del mismo, y nos vamos a centrar en la parte programada en **javascript**.

A la hora de realizar el desarrollo existen distintos aspectos y etapas que tiene el juego. A continuación se exponen y serán detallados más adelante:

1. Variables globales

2. Comienzo del juego
    - Barajar y cargar el tapete inicial
    - Control del tiempo y contadores
3. Mover carta entre tapetes
4. Fin del juego

## 4.1. Variables globales

A la hora de realizar el juego se han creado distintas variables globales que pueden ser accedidas desde cualquier parte del juego. Esto facilita el realizar cambios en los valores de las mismas.

Sin entrar en las variables de los palos y los números, existen tres variables globales principales que son las siguientes:

- **Tapetes**: Los tapetes existentes en el juego, que son objetos DOM del HTML.

Variable para los tapetes, con el DOM correspondiente

```
// Tapetes
let tapetes = [
    document.getElementById("inicial"),
    document.getElementById("receptor1"),
    document.getElementById("receptor2"),
    document.getElementById("receptor3"),
    document.getElementById("receptor4"),
    document.getElementById("sobrantes")
]
```

- **Mazos**: Los mazos que habrá durante el juego, cada uno encima de un tapete distinto.

Variable para los mazos

```
// Mazos
let mazos = [
    [], // inicial
```

```
    [], // receptor1
    [], // receptor2
    [], // receptor3
    [], // receptor4
    [], // sobrantes
]
```

- **Contadores**: Los contadores que controlan cuántas cartas hay en cada tapete. También son elementos del DOM para poder ser actualizados.

Cada una de estas tres variables son un array de seis posiciones, siguiendo el orden:

- **Inicial**: Tiene que ver con el tapete/mazo inicial. Posición cero del array.
- **Tapetes receptores**: Hay cuatro tapetes receptores, y por tanto cuatro posiciones en el array para cada uno de ellos (1..4).
- **Sobrante**: La última posición (5) es para el tapete sobrante, su mazo y el contador del mismo.

## 4.2. Comienzo del juego

Tal como se ha podido ver previamente, el comienzo del juego se ha diferenciado en dos apartados, que se realizan de manera consecutiva.

Para que comience el juego, primero debemos hacer clic sobre el botón situado en la parte inferior con el aspecto:


Lo que llamará a la función **comenzar_juego**:

Evento onclick del botón para comenzar el juego

```
document.getElementById("reset").onclick = comenzar_juego;
```

### 4.2.1. Barajar y cargar el tapete inicial

Este apartado es el principal del juego, ya que se encarga de crear el mazo inicial juntando las cartas de cada uno de los palos de la baraja.

Para ello se recorren los palos y las cartas y se añaden al array de mazos (posición "0", para el mazo inicial). Se crean objetos imágenes, indicando cuál es el origen de la imagen a utilizar, indicamos que los objetos se pueden mover ("*draggable*") y añadimos ciertos atributos que después necesitaremos para aceptar o denegar el movimiento entre tapetes.

Este mazo inicial se baraja en la función **barajar**, que recibe el mazo como parámetro. Esta función también se utilizará al pasar las cartas del tapete sobrantes al inicial.

Y por último, se colocan las cartas sobre el tapete inicial a través de la función **cargar_tapete_inicial** haciendo un efecto en cascada diagonal.

### 4.2.2. Control del tiempo y contadores

Este apartado no es estrictamente necesario para el juego, pero da valor a la hora de que sea más ameno si queremos tratar de batir nuestro propio récord, ya sea de tardar menos o realizar el menor número de movimientos posibles.

El juego cuenta con un control de tiempo, que es un ***interval*** que se llama cada segundo, realizando un incremento de tiempo para posteriormente ser visualizado en el formato "HH:MM:SS" (donde "HH" son horas, "MM" minutos y "SS" segundos).

Existe un contador para cada tapete, situado en su esquina inferior derecha, para que el usuario sepa cuántas cartas tiene el mazo que está sobre él.

Y por último, existe el contador de movimientos, que se incrementa cada vez que movemos una carta sobre cada uno de los tapetes.

Tanto el tiempo como el contador de movimientos se puede ver en la parte superior de la página, encima de la mesa de juego:


Tras iniciar el juego, el botón que daba comienzo al juego se cambia de color y pondrá “Reiniciar”, lo que reiniciará el juego, limpiando tapetes, contadores, volviendo a barajar y situando las cartas de nuevo en el tapete inicial.


### 4.2.3. Mover carta entre tapetes

Este es el apartado más importante dentro de la programación del juego, ya que contiene la lógica de lo que se debe realizar al tratar de mover una carta entre tapetes.

Toda esta funcionalidad está basada en la API *Drag and Drop* añadida en el primer borrador de HTML5 en el 2008, que nos permite arrastrar y soltar elementos de la web generando un evento.

A continuación se detallan cuáles son los pasos que se realizan en esta funcionalidad:

1. **Al mover carta**: Hay que tener en cuenta que sólo se puede mover la carta que está más arriba del montón del tapete inicial y del tapete sobrantes, quedando el resto “bloqueadas” (incluidas las de los tapetes receptores).

   Al intentar mover una de estas cartas superiores, se lanza el evento y se obtiene información de la carta y del mazo origen.

2. **Soltar carta**: Este es el momento en el que la carta se suelta sobre uno de los tapetes receptores o sobrantes. El tapete inicial no permite que se suelten cartas sobre él.

   Si se mueve la carta sobre el tapete de sobrantes, no hay que realizar ninguna comprobación, por lo que se pasa al siguiente paso.

   En cambio, si se intenta soltar sobre un tapete receptor, hay que comprobar lo siguiente:

- Si no hay cartas en el tapete, sólo se permite soltar una carta con el número 12.
- Si hay alguna carta sobre el tapete, la carta que se suelte encima tiene que ser de una única unidad inferior y de color distinto.

Si cualquiera de estas dos opciones no es válida, la carta vuelve al tapete de origen. A continuación parte de la función **soltar_carta**:

Controlamos si la carta se puede soltar (parte de función **soltar_carta**)

```
if (mazo_receptor.length == 0 && numero == 12){
  // es una carta con número 12 y va a ser la primera carta del tapete
  carta.draggable = false;
  // movemos la carta de un mazo a otro
  mover_carta(mazo_origen,mazo_receptor,tapete_receptor,cont_origen,cont_receptor);
} else if (mazo_receptor.length != 0) {
  // no es la primera carta del tapete. Tenemos que ver
  // el número de la carta que ya está en ese tapete y su color.
  carta_mazo_num = mazo_receptor[mazo_receptor.length-1].getAttribute("data-numero");
  carta_mazo_col = mazo_receptor[mazo_receptor.length-1].getAttribute("data-color");
  if (carta_mazo_num-1 == numero && carta_mazo_col != color ) {
    // aceptamos la carta porque el número es uno menos que la que ya está
    // y es de distinto color/palo
    carta.draggable = false;
    mover_carta(mazo_origen,mazo_receptor,tapete_receptor,cont_origen,cont_receptor);
  }
}
```

3. **Mover carta**: Si las comprobaciones del paso anterior son correctas, la carta se mueve al nuevo tapete. Tal como se puede ver en el código anterior, existe la función **mover_carta** que recibe varios parámetros.

   En esta función también se actualizan los **contadores** llamando a otras funcione para restar o incrementar el contador, según sea el caso.

Hay que recordar que cuando se mueve la última carta del tapete inicial (ya sea sobre uno de los tapetes receptores o el de sobrantes) se cogerán todas las cartas del tapete sobrantes, se barajarán de nuevo y se colocarán en el tapete inicial para continuar la partida.

Para limpiar el tapete correspondiente, lo que se ha realizado es la eliminación de los nodos hijos que son “img”, y para ello se ha creado una función:

Función que limpia un tapete de imágenes

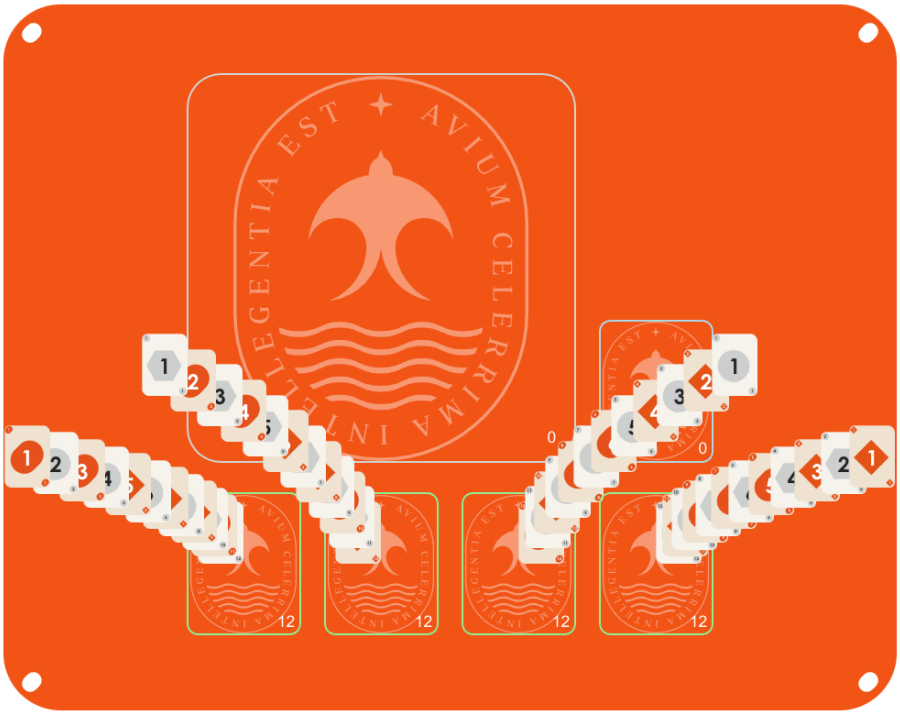
```
// función para limpiar un tapete de cartas
function limpiar_tapete(tapete) {
    while (tapete.getElementsByTagName('img').length>0) {
        tapete.getElementsByTagName('img')[0].remove();
    }
}
```

### 4.2.4. Fin del juego

La comprobación de ver si el juego ha finalizado se realiza cada vez que se mueve una carta de manera correcta.

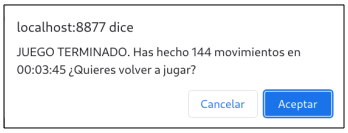
Esta comprobación es sencilla, y lo que hace es comprobar que el número de cartas de los mazos inicial y sobrantes es igual a cero. En este caso el juego se da por terminado.

A la hora de finalizar el juego se para el contador y las cartas realizarán un efecto de desplazamiento por la mesa de juego, terminando como la imagen siguiente:



Después saldrá un aviso con el número de movimientos realizados, junto con

tiempo empleado y preguntando al usuario si quiere volver a jugar:




En caso de aceptar el aviso, el juego se reiniciará. En caso de cancelar, podremos ver cómo ha quedado el tapete, el tiempo y el número de movimientos. En todo caso, podremos volver a reiniciar el juego con el botón de “Reiniciar”.

# 5. Aspecto visual

Para mejorar el aspecto visual del juego, se han realizado unas pequeñas modificaciones a la plantilla propuesta para el juego:

- **Colores corporativos**: El tapete se ha modificado para que cuente con el color corporativo de la VIU, haciendo uso de una variable dentro del CSS.

  Variable global en CSS para el color

```css
/* Variable global de color */
:root {
    --corporativo: #F25416;
}
```

- **Redondear tapetes**: Gracias a la funcionalidad “border-radius” de CSS se ha hecho que los tapetes tengan un aspecto más redondeados en lugar de cuadrados.

- **Personalizar tapetes**: Para personalizar la mesa de juego, se ha añadido el sello de la VIU a cada uno de los tapetes, que junto con el borde, hace que se vea más claramente dónde se pueden colocar las cartas.

- **Borde de la mesa**: Se ha añadido un borde, que sólo aparece en las esquinas de la mesa de juego, con el fondo del logotipo de la VIU en color blanco.

- **Uso de Bootstrap**: Se ha hecho uso del sistema *grid* de Bootstrap para dar tamaño a la mesa y colocar el temporizador y el contador de movimientos.

# 6. Dificultades del proyecto

Como todo proyecto de programación, durante el desarrollo nos podemos encontrar con ciertas dificultades que deben ser subsanadas para llegar a cumplir los requisitos planteados al comienzo del proyecto.

En este caso, la lógica del juego hace que sea un desarrollo sencillo, en el que no ha habido que tener demasiadas cosas en cuenta, por lo que la lógica del juego ha resultado una tarea sencilla.

Ahora bien, eso no quita que haya habido dificultades al realizar el proyecto, ya que se han usado ciertos aspectos nunca vistos previamente por el programador, que son:

- **API drag and drop**: Aunque la mecánica de la API de HTML5 es sencilla, se ha tenido que profundizar en cómo funcionan las distintos funciones que se pueden utilizar con la API. Con ello, también se ha investigado la información que se puede enviar a través de los eventos generados en cada una de esas funciones.

  Las funciones de la API utilizadas han sido "**ondragstart**", al comenzar el movimiento de la carta, y "**ondrop**", para cuando se suelta sobre un tapete. Otras de las funciones existentes en la API ("onstart", "ondragend", "ondragleave") no han sido necesarias, pero se han inicializado para que no hagan nada.

- **Colocación de las cartas**: Debido a la flexibilidad de CSS, y la colocación de imágenes teniendo en cuenta su posición relativa respecto al contenedor, ha sido necesario entender cómo funciona los estilos "float" y "position", sobre todo para el efecto final.

- **Intervalos**: Aunque ya habían sido utilizados por parte del programador, debido a que no los utiliza de manera habitual, ha sido interesante recordar su funcionalidad.
- **Promesas**: Igual que el caso anterior, aunque se conocía el uso de "***Promise***" para realizar **programación asíncrona**, se ha hecho uso de ellas para el efecto final de movimiento de las cartas. De esta manera se realiza una pequeña pausa al mover cada una de las cartas, y hay que esperar para que salga el aviso de finalización del juego.

# 7. Conclusiones

A la hora de afrontar un proyecto, aunque el algoritmo generado pueda ser sencillo, no quita que exista un proceso de aprendizaje a la hora de utilizar ciertas herramientas no conocidas.

Este ha sido el caso durante la programación en **Javascript** y el uso de ciertas funcionalidades de **HTML5** y **CSS** versión 3 no conocidas hasta ahora.

Este desarrollo nos va a permitir que las aplicaciones generadas a partir de ahora para el lado del cliente (conocido como programación ***frontend***) tengan más funcionalidades, y de esta manera podamos realizar aplicaciones más complejas.