DESARROLLO DE APLICACIONES Y SERVICIOS WEB II: LADO DEL CLIENTE (FRONT-END) Y MULTIMEDIA

ACTIVIDAD 1

1. Introducción y objetivos

En la presente actividad vamos a utilizar JavaScript junto con HTML y CSS para diseñar un sencillo juego. Emplearemos adicionalmente varias librerías JavaScript con las que pretendemos aumentar la cantidad de recursos y herramientas a utilizar. Concretamente, emplearemos la librería Bootstrap y otra necesaria por esta: jQuery. Bootstrap tiene varias funcionalidades, entre ellas, mostrar botones con un aspecto estético mejorado respecto a la representación estándar.

En esta actividad vamos a utilizar lo estudiado en clases de teoría. Aunque las especificaciones pueden ser incompletas, deberían ser suficientes para conseguir un aspecto bastante similar a las referencias gráficas que aquí aparecen.

Durante el transcurso de la actividad, el alumno podrá apreciar que una misma idea puede implementarse de múltiples formas; por ello se pretende que se reflexione sobre estas alternativas y elija la opción que le parezca más conveniente, tratando de justificarla adecuadamente. Asimismo, es probable que deba consultar algunos detalles de implementación no impartidos en la asignatura, para que así el alumno ejercite la competencia de aprendizaje permanente.

2. Explicación del juego

La actividad consiste en implementar una versión elemental del juego del solitario. Respecto a las reglas del juego, hay que tener en cuenta las siguientes consideraciones:

- La baraja de cartas será una baraja VIU, que consiste en:
  - 12 números: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 y 12.
  - 4 palos: “vius”, cuadrados, hexágonos y círculos.

  Importante: inicialmente se debe trabajar con un subconjunto de cartas, para no hacer el juego demasiado denso o largo y poder hacer pruebas de forma rápida y ágil, como por ejemplo: las cartas 9-12 de cada palo.

Una vez que el juego esté mínimamente validado, debería ser algo extremadamente fácil conmutar al juego completo de cartas, por lo que todo el software debe preconcebirse para este cambio de circunstancias.

- Existirán 6 tapetes (superficies que albergarán mazos de cartas): el tapete inicial, el tapete de sobrantes y cuatro tapetes receptores, tal y como puede observarse en la Figura 1, que albergarán sendos mazos de cartas a lo largo de la partida.
- El juego consiste en coger la carta de arriba, bien del mazo ubicado en el tapete inicial, bien del tapete de sobrantes, e intentar depositarla en alguno de los mazos de los cuatro tapetes receptores, siempre y cuando se cumplan ciertas condiciones.
- El uso de las cartas del mazo de cada tapete es el siguiente:
    - Tapete inicial: conteniendo al principio el mazo completo y barajado aleatoriamente. Las cartas se cogerán por arriba, de una en una. La intención es intentar colocar la carta cogida en alguno de los tapetes receptores siempre y cuando se cumplan ciertas condiciones.
    - Tapete de sobrantes: empleado para depositar temporalmente en su mazo las cartas provenientes del mazo del tapete inicial. Cualquier carta del tapete inicial (siempre y cuando sea la de arriba) se puede depositar en este tapete. Además, la última carta depositada en el mazo de este tapete sirve de fuente alternativa de cartas al mazo del tapete inicial, para intentar depositarla sobre algún mazo de los tapetes receptores en cualquier momento, siempre y cuando se cumplan ciertas reglas tal y como ocurriría si proviniera del mazo del tapete inicial.
    - Cuatro tapetes receptores: donde se irán depositando, sobre los mazos respectivos, las cartas en orden decreciente (comenzará obligatoriamente con el 12 como primera carta y finalizará con la última) y alternando los colores (naranja y gris) en la secuencia de cartas que se van depositando en cada mazo. Una vez depositada una carta en uno de los cuatro tapetes receptores, no se podrá volver a colocar en otro tapete.
- Cuando se agoten las cartas del mazo del tapete inicial, las que queden en el tapete de sobrantes serán automáticamente barajadas y dispuestas de nuevo en el tapete inicial, volviendo otra vez a comenzar con estas cartas restantes.
- El juego finaliza cuando no queda ninguna carta ni en el tapete inicial ni en el tapete de sobrantes. Cuando ocurra tal situación, se mostrará una ventana de aviso indicando el fin del juego y proporcionando información sobre la partida: el tiempo de juego y la cantidad de movimientos realizados.

En la Figura 1 puede observarse el juego en un supuesto estado inicial.

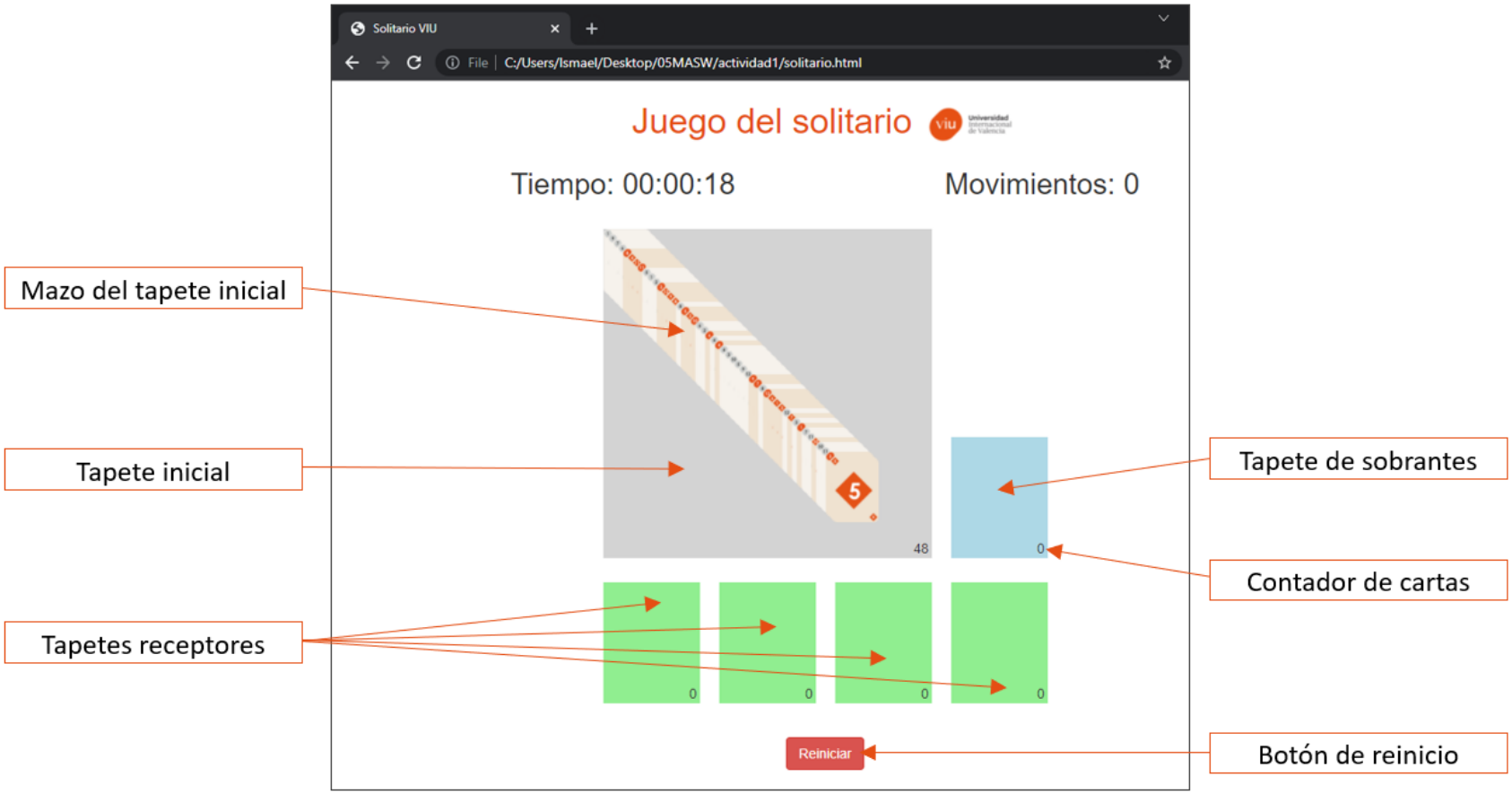

Figura 1. Situación inicial tras seleccionar el juego

3. Desarrollo y sugerencias

3.1. Material proporcionado

Como punto de partida, se le proporciona al alumno las siguientes carpetas y ficheros:

- Carpeta **`imágenes`**, la cual contiene:
  - Carpeta **`baraja`**, con 48 ficheros .png correspondientes a la baraja VIU. Obsérvese detenidamente de qué manera están formados los nombres de los ficheros y cómo identifican unívocamente cada carta.
  - **`logoVIU.png`**, imagen con el logo de la VIU.
- **`solitario.html`**: fichero HTML con la estructura básica de la página web. Aunque están prácticamente todos los elementos, será necesario incluir algún elemento para conseguir el aspecto mostrado en la Figura 1.
- **`solitario.css`**: fichero de estilos del juego. Al igual que con el fichero `solitario.html`, será necesario completar este fichero para hacer una página con un aspecto similar al mostrado en la Figura 1.

  Importante: el aspecto no tiene que ser exactamente el mostrado en la Figura 1, y el estudiante puede elegir los colores de los tapetes, así como el tipo de fuente utilizado y el formato del botón, entre otros.
- **`solitario.js`**: contiene una estructura elemental del fichero JavaScript que recoge las acciones a realizar. Así, este fichero se corresponde con una implementación parcial de la solución. Debe completarse correctamente, y para ello podemos acudir a las zonas del código resaltadas con el comentario:

```
/*** !!!!!!!!!!!!!!!!!!! CODIGO !!!!!!!!!!!!!!!!!!! **/
```

Se sugiere que se sigan las indicaciones que aparecen como comentarios tanto en los ficheros `solitario.css` y `solitario.html`, y especialmente en `solitario.js` para primeramente comprender el contexto del juego y a continuación rellenar el código JavaScript necesario para poner en marcha los primeros pasos del mismo.

3.2. Creación de los tapetes iniciales y contadores

A continuación se indican una serie de tareas que hay que realizar en los diferentes ficheros `solitario.*`:

- Insertar el logo de la VIU a la derecha del encabezado principal.
- Crear el mazo de la baraja con las cartas oportunas.

- Barajar el mazo de cartas y ponerlo sobre el tapete denotado en el código como tapete inicial. Según la plantilla, un tapete es un `div`, por lo que dejar una carta sobre un tapete será simplemente añadir el elemento `img` oportuno como hijo del citado `div`.
- Se sugiere que se tengan en cuenta detalles como los valores respectivos de la propiedad `position` que debe tener en los elementos contenedores y en los elementos contenidos, así como los valores para las propiedades `top` y `left` para indicar cierto desplazamiento vertical y horizontal a cada carta respecto a la anterior y así ofrecer una disposición en escalera del mazo sobre el tapete.
- Indicar el número de cartas que hay en el mazo de este tapete inicial. Inicialmente, convendría hacer esta indicación del número de cartas de todos los mazos de todos los tapetes.
- Respecto a la ubicación de esta indicación, la sugerencia es hacerlo en la parte inferior derecha, tal y como puede observarse en la Figura 1.
- Mostrar cómo va corriendo el tiempo que va transcurriendo con el formato `hh:mm:ss` en el sitio indicado a tal efecto.
- Los resultados esperados pueden apreciarse gráficamente en la Figura 1. En los ficheros proporcionados podrán encontrarse ayudas y sugerencias en los comentarios para alcanzar fácilmente estos objetivos.

3.3. Drag & Drop

3.3.1. Información adicional en la creación de los elementos `<img>`

Se sugiere que para la creación de los elementos `<img>` que representan cada una de las cartas, se emplee un doble bucle anidado (con, por ejemplo, índices `i` y `j`) y se añada a cada elemento `<img>` un par de atributos tipo `data-` que sirvan para identificar de forma clara y sencilla la carta, concretamente el número de la carta y el palo. Por ejemplo:

```
img_carta.setAttribute("data-palo", palos[i]);
img_carta.setAttribute("data-numero", numeros[j]);
```

Téngase en cuenta un pequeño detalle gráfico (que tendrá sus consecuencias probablemente en el control):

- Las cartas del tapete inicial tendrán un desplazamiento geométrico superior e izquierdo de varios píxeles (cinco, por ejemplo) entre una carta y su antecesora.
- Las cartas en el resto de tapetes no tendrán dicho desplazamiento, por lo que cada carta estará ubicada exactamente sobre su antecesora.

3.3.2. Arrastre y suelta de cartas

Recuérdese que es necesario programar los siguientes eventos asociados a los dos tipos de objetos involucrados:

- Objeto que se mueve (carta, es decir, elemento `<img>`):
  - Eventos: `dragstart`, `drag` y `dragend`
- Objeto que recibe al que se mueve (tapete, es decir, elemento `<div>`):
  - Eventos: `dragenter`, `dragover`, `dragleave` y `drop`

Objeto que se mueve

Para atender oportunamente a los eventos del objeto que se mueve, es decir, la carta (elemento `<img>`) hay que asignar funciones a las siguientes propiedades. Estas funciones serán invocadas cuando ocurra el evento; además serán invocadas con un argumento que contiene información sobre el evento ocurrido (parámetro `e` en los ejemplos). Se sugieren las siguientes asignaciones a las respectivas propiedades relacionadas con el movimiento:

- `objeto_que_se_mueve.ondragstart = al_mover;`
- `objeto_que_se_mueve.ondrag = function(e) { };`
- `objeto_que_se_mueve.ondragend = function() { };`

Puede observarse cómo para los eventos `drag` y `dragend` se han asignado funciones vacías, es decir, no se va a llevar a cabo ningún tipo de acción cuando ocurran cualquiera de estos dos eventos. Es más, en una de ellas, incluso se ha omitido el parámetro `e`, ya que no va a ser utilizado, y en JavaScript se puede hacer caso omiso de los parámetros en la descripción de una función.

Sin embargo, para el evento `dragstart`, es decir, cuando arranque el movimiento, la única acción a realizar será guardar en ciertas propiedades del propio evento (y que serán transferidas junto con el mismo) cierta información que nos permita identificar de alguna manera el objeto (carta) que se está moviendo, como por ejemplo el palo y el número guardado en atributos de tipo `data-`, tal y como se sugirió en el subapartado anterior, así como, si así lo deseamos, el atributo `id` que le hubiéramos proporcionado a la carta, si así lo hubiéramos hecho:

```
function al_mover(e) {
  e.dataTransfer.setData( "text/plain/numero", e.target.dataset["numero"] );
  e.dataTransfer.setData( "text/plain/palo", e.target.dataset["palo"] );
  e.dataTransfer.setData( "text/plain/id", e.target.id );
}
```

Esta función `al_mover` es la que se ha asignado a la propiedad `.ondragstart` de la carta que potencialmente se puede mover. El parámetro `e` representa el evento de arranque de movimiento; la propiedad `e.target` es el elemento HTML que se está moviendo, es decir, la carta (elemento `<img>`), y por lo tanto `e.target.dataset[atributo sin prefijo data-]` es la forma con la que podemos acceder a un atributo de usuario de prefijo `data-`.

Esto no deja de ser una sugerencia ya que se pueden plantear esquemas alternativos para manejar oportunamente esta información.

Los dos argumentos de la función `setData` del código ejemplo anterior son realmente un par *nombre-valor*: en primer lugar, el *nombre* del valor que se va a pasar (es usual que tenga aspecto de tipo MIME con bastantes grados de libertad, tal y como se puede observar en el ejemplo); el segundo argumento es propiamente el valor que deseamos pasar. Posteriormente, este valor podrá ser recuperado (por ejemplo, en la función asociada al evento `drop`, si así se estima necesario) aludiendo a su nombre empleando la función `getData` tal y como se describe en el siguiente ejemplo:

```
let numero = e.dataTransfer.getData("text/plain/numero");
let palo = e.dataTransfer.getData("text/plain/palo");
let carta_id = e.dataTransfer.getData("text/plain/id");
```

Téngase en cuenta el siguiente detalle: las reglas indican que tanto del mazo del tapete inicial como del de sobrantes, únicamente la última carta tiene capacidad de moverse. Contrólese el valor del atributo `draggable` oportunamente: por ejemplo, considérese alguna de las siguientes posibilidades —téngase especial cuidado con el tipo oportuno, `string` o `boolean` según el método de ajuste del atributo—:

```
carta.draggable = true;
carta.draggable = false;
carta.setAttribute( "draggable", true );
carta.setAttribute( "draggable", "true" );
carta.setAttribute( "draggable", false );
carta.setAttribute( "draggable", "false" );
```

y que cuando una carta se mueva a un tapete receptor o al tapete de sobrantes (véase el siguiente apartado), entonces una nueva carta del tapete original asumirá el rol de carta que se puede mover del mismo; actúese en consecuencia sobre la carta que acaba de moverse y que ha podido ser depositada en algún otro tapete. Podría ser interesante, si el alumno así lo estima oportuno, añadir un nuevo atributo del tipo `data-` a la carta que se mueve, indicando el tapete de procedencia para actuar en consecuencia ya en el tapete receptor.

Objeto que recibe al que se mueve

Del mismo modo que han sido programadas las acciones a realizar en el objeto que se mueve, también habrá que programar ciertas acciones asociadas a eventos que ocurrirán sobre el objeto que recibirá al objeto que se mueve.

Tal y como hemos comentado anteriormente, la sugerencia que indicamos es que el objeto que se mueva sea un elemento HTML de tipo `<img>` y el que reciba a este, un elemento `div`.

Hay que programar la reacción a los eventos: `dragenter`, `dragover`, `dragleave` y `drop`. Para los tres primeros no se va a realizar ninguna acción específica salvo indicarle al navegador que no haga lo que tiene asignado por defecto, ya que ello tendría como consecuencia la imposibilidad de conseguir el resultado deseado. Para el evento `drop`, sí que habrá que especificar una serie de acciones que estarán íntimamente ligadas con las reglas y operatividad del juego. Así pues, se sugiere que se programen los eventos indicados tal y como se muestra a continuación:

```
destino.ondragenter = function(e) { e.preventDefault(); };
destino.ondragover = function(e) { e.preventDefault(); };
destino.ondragleave = function(e) { e.preventDefault(); };
destino.ondrop = soltar;
```

Puede observarse que, para los tres primeros eventos, la acción a realizar se reduce a prevenir o evitar hacer cierta acción por defecto por parte del navegador mediante el método `preventDefault` asociado al evento ocurrido e.

Para el evento drop se sugiere que se diseñe un método que consista en recoger la carta soltada, si es que así debe ocurrir. Recuérdese que esta función soltar también debería tener como primera instrucción `e.preventDefault()` para eludir ejecutar la acción por defecto. Se sugiere que a continuación se extraigan del propio evento e y mediante el método `e.dataTransfer.getData(...)` la información de la carta que se introdujo en el evento mediante el método `e.dataTransfer.setData(...)` en la función asignada al evento `dragstart` del objeto que se movía. Una vez identificada la carta que se acaba de soltar, entonces hay que comprobar si cumple las condiciones necesarias para que sea recibida en el tapete en cuestión.

Importante: hay que tener en cuenta la secuencia decreciente de número de carta y la alternancia de colores naranja y gris. Es decir, únicamente se puede depositar una carta sobre otra si corresponde al número inmediatamente inferior y si es de un color diferente al de la carta. Por ejemplo, una carta “6 vius” (color naranja) únicamente se podrá depositar debajo de las siguientes cartas: “7 hexágonos” o “7 círculos” (ambas de color gris).

Importante: los tapetes receptores vacíos solo aceptan cartas con el número 12.

En caso de que se cumplan las citadas condiciones, la carta será añadida al tapete (eliminada, por tanto, automáticamente del tapete inicial (o del de sobrantes, si ese es su origen); esto lo hará en principio el navegador). Recuérdese que se sugirió que las cartas de cada tapete estuvieran dentro de un array (con prefijo `mazo-` en las variables empleadas para ello); en tal caso se puede eliminar de un mazo e introducirla en otro mediante los métodos pop y push propios de los arrays en JavaScript. Deberían actualizarse convenientemente los contadores de cartas de los tapetes origen y destino del movimiento. Si el tapete destino es un tapete receptor, se sugiere que se pongan las cartas una encima de otra sin desplazamiento entre ellas. Entre ciertas posibilidades, hay una forma sencilla de *centrar* un elemento dentro de otro:

```
// Con esto, el vértice superior izquierdo de la carta queda en el centro del tapete
carta.style.top = "50 %";
carta.style.left = "50 %";
// Con la siguiente traslación, se centra definitivamente en el tapete: se desplaza,
// a la izquierda y hacia arriba (valores negativos) el 50 % de las dimensiones de
// la carta.
carta.style.transform="translate(-50 %, -50 %)";
tapete_destino.appendChild(carta);
```

Si no se cumplen las condiciones de alternancia y orden, simplemente no habrá que hacer nada y el navegador se encargará automáticamente de *deshacer* el movimiento mostrando la carta en el sitio en el que estaba ubicada originalmente.

Téngase en cuenta que la programación de estos eventos debe ser para todos los tapetes potencialmente receptores. Ello implica que se puede realizar de forma replicada o individualizada para todos y cada uno de ellos, o bien, diseñar una función lo suficientemente genérica como para que se asigne por igual a

todos los tapetes receptores. Téngase en cuenta en este último caso, el papel que puede tomar la palabra reservada `this` dentro de la función asignada a la propiedad ondrop de cada tapete.

Una vez que el juego esté suficientemente validado, extiéndase el número de cartas del juego a toda la baraja.

La Figura 2 muestra un ejemplo de una partida en curso.

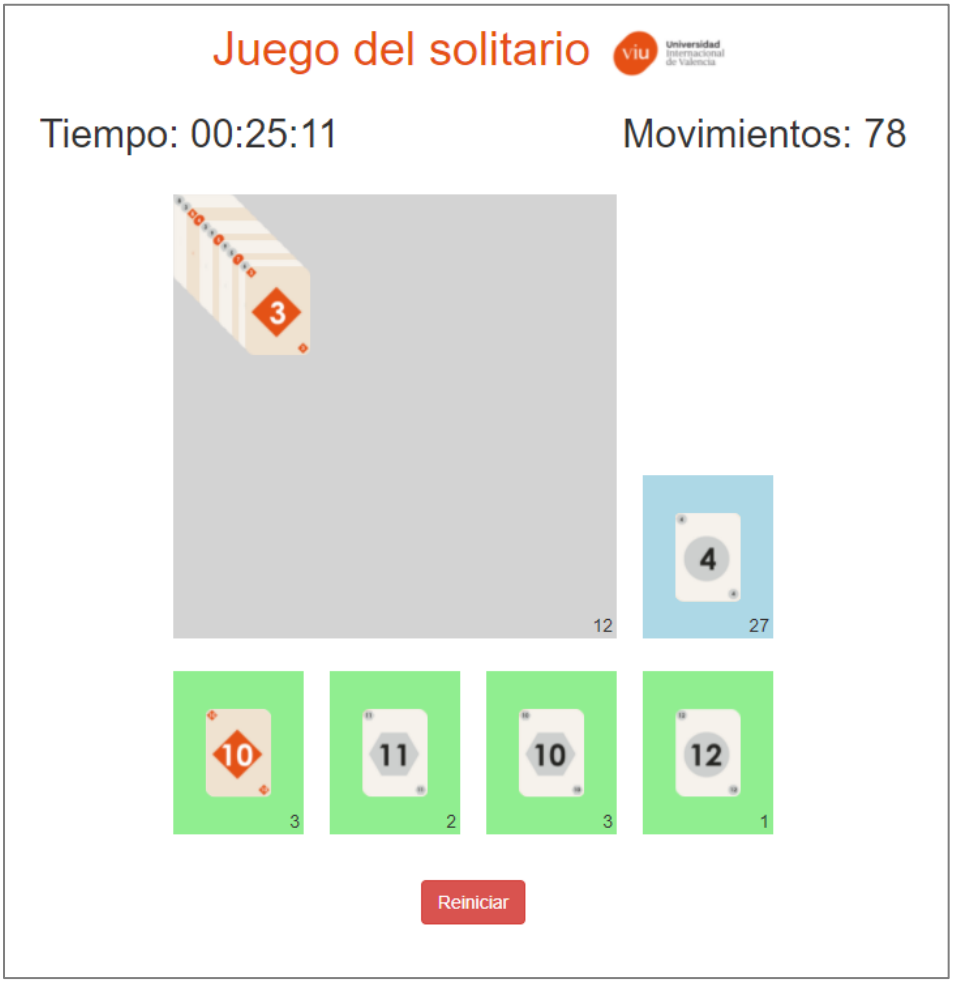

Figura 2. Juego en curso

Fin de juego

Cuando se den las condiciones oportunas de fin de juego, debe denotarse mediante una ventana de alerta, parándose el contador de tiempo, indicándolo oportunamente en la propia ventana de alerta, junto con la cantidad de movimientos habidos durante la partida.

Al pulsar el botón de `Reiniciar`, todo el juego debería aparecer en el estado inicial del mismo.

Quedan numerosos detalles por describir, pero se dejan a la libre interpretación del alumno para concretarlos, extenderlos y mejorarlos, siempre y cuando, la solución aportada sea razonable en el contexto en el que se está trabajando.

3.3.3. Botones: utilización de Bootstrap

La librería BootStrap tiene numerosos usos: permite concebir un comportamiento responsivo de la página web, adaptándose a las dimensiones de la ventana del navegador. También permite utilizar rediseños de elementos HTML como botones, cajetines, etc., con una estética muy particular y mejorada normalmente, respecto al aspecto estándar. Su uso requiere la importación de tanto de un fichero css de estilo como un fichero JavaScript.

BootStrap requiere adicionalmente la librería jQuery para su funcionamiento. Por cuestiones de eficiencia se sugiere que se importen ambas librerías desde ciertas CDN. Una posibilidad es la que se muestra a continuación:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
```

Se sugiere también que esta importación se realice antes de la de nuestros estilos propios para que, en el caso de tener la misma especificidad, nuestros estilos sean los que se impongan y no los de BootStrap, en el supuesto caso de que esas sean nuestras intenciones. En cualquier caso, observaremos muy probablemente que el aspecto de nuestra página ha cambiado en algunos detalles cuyo motivo será muy probablemente que BootStrap reescribe algunas declaraciones de estilo que no son controladas por nuestros ficheros de estilo. Búsquese información sobre tipos de botones en BootStrap. Una sugerencia de uso puede ser la que se muestra a continuación:

```
<button type="button" class="btn btn-primary" id="aSolitario">
  Solitario
</button>
```

4. Resultados a entregar

Como resultado de la práctica los alumnos deberán subir a la plataforma Campus VIU:

- **`solitario.html`**: fichero HTML completado con sus desarrollos.
- **`solitario.css`**: fichero de estilos completado con sus estilos.
- **`solitario.js`**: fichero JavaScript con los desarrollos realizados.
- Informe/memoria de la práctica, en el que se expliquen los desarrollos realizados, justificando los desarrollos realizados (se puede utilizar para ello capturas de pantalla y código fuente).

Dichos ficheros se pueden subir en un fichero .rar o .zip.

Consideraciones de la entrega:

- No se admitirán actividades entregadas fuera de plazo.
- La actividad ha de subirse a Campus VIU, por lo que no se aceptan actividades entregadas por otros medios (como correo electrónico).

5. Evaluación

Para la evaluación de esta actividad se tendrá en cuenta:

- **Funcionalidad (35 %)**: el código entregado ha de funcionar.
- **Cumplimiento de las especificaciones** indicadas a lo largo de la memoria **(35 %)**.
- **Limpieza de código** entregado **(10 %)**.
- **Memoria (10 %).**
- Se valorarán **otros** aspectos que demuestren el dominio y el trabajo del alumno, como la realización de funcionalidades adicionales o un diseño de la página más elaborado **(10 %)**.

Cada uno de los anteriores criterios de evaluación se evaluará según la siguiente escala:

- Perfecto (100 %).
- Excelente (85 %).
- Notable (70 %).
- Suficiente (50 %).
- Insuficiente (25 %).
- Nulo (0 %).