Práctica 6

Juego del solitario II

Integración de Servicios Telemáticos
Máster Universitario en Ingeniería de Telecomunicación
ETSIT-UPV

Ismael de Fez Lava
Marisol García Valls
F. J. Martínez Zaldívar

Índice

1. Introducción y objetivos

En esta práctica finalizaremos la programación del juego del solitario basándonos en los resultados de las prácticas anteriores. Se utilizará JavaScript para interactuar con el usuario y para controlar que en todo momento se cumplan las reglas del juego.

En esta serie de prácticas vamos a seguir utilizando 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 aparezcan.

Durante el transcurso de la práctica 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 en forma de comentarios dentro del código HTML o CSS.

Asímismo, es probable que deba consultar algunos detalles de implementación todavía no impartidos en la asignatura; la intención de ello es ejercitar y valorar oportunamente la competencia transversal de *aprendizaje permanente*.

1.1. Organización de la práctica

Los grupos de prácticas pueden ser de uno o de dos alumnos. Se sugiere que estos grupos sean estables a lo largo de todas las prácticas y en esta, especialmente, ya que aprovecha los resultados de la práctica anterior. Se dedicará una sesión de dos horas para la realización de la práctica.

Se seguirá utilizando el repositorio de las dos prácticas anteriores (`Peri-digi-`), añadiendo con nuevos *commits* las aportaciones consecuentes de la realización de la práctica.

Recuérdese que al finalizar la sesión de prácticas del laboratorio se deberá realizar un *commit—push* etiquetado como `pr6SesionFin`. Para quienes no puedan acabar esta práctica en el transcurso de la sesión, se añadirá una semana adicional para ello. En tal caso, el *commit* que deba considerarse como el definitivo deberá estar etiquetado como `pr6DefFin`.

1.2. Material necesario

El material necesario para llevar a cabo la práctica será un ordenador de sobremesa o un portátil, con cualquier sistema operativo convencional, un editor de texto plano y un navegador (preferiblemente alguno que posea y del que se conozcan y manejen capacidades de depuración).

Muy probablemente será necesario consultar informción de HTML5, CSS y JavaScript. Recuérdese que un sitio bastante útil es `http://www.w3schools.com`, entre otros.

2. Control del juego

En principio faltan dos detalles importantes para que el juego sea operativo:

- Permitir el *arrastre y suelta* de las cartas *(drag & drop)*
- Controlar los movimientos permitidos y prohibidos

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

En la práctica anterior se crearon sintéticamente mediante JavaScript elementos `<img>` indicando la URL oportuna a partir del palo y del número de carta, especificándolo en el atributo `src`. Se sugiere que se modifique sutilmente el código, de modo que en su creación en cierto doble bucle anidado (con, por ejemplo, índices `i` y `j` respectivamente), se añada a este elemento `<img>` un par de atributos de tipo `data-` que nos sirvan para identificar claramente y de forma cómoda 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]);
```

Modifíquese oportunamente el código con los nombres de las variables realmente empleados, así como los índices `i` y `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 (tres, por ejemplo) entre una carta y su antecesora
- Las cartas en el resto de tapetes **no** tendrán dicho desplazmiento por lo que una estará ubicada exactamente sobre su antecesora.

2.2. Arrastre y suelta de cartas

Recuérdese que es necesario programar lo 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`

2.2.1. 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 siguiente 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 enventos `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 permitan 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 hubieramos proporcionado a la carta, si así lo hubieramos 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, sólo 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.

2.2.2. 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.ondraleave  = 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 (se debe tener en cuenta la secuencia decreciente de número de carta y la alternancia de colores rojo y negro).

En caso de que se cumplan las citadas condiciones, la carta será añadida al tapete (eliminada, por tanto, automáticamente del tapete inicial; 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 —véase el fichero `plantilla.html` de la práctica anterior—); 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 (consúltese esta información). 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 defintivamente 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 (para ello, obsérvese la plantilla que se proporcionó en su momento).

2.2.3. Fin de juego

Cuando se den las condiciones oportunas de fin de juego, este debe denotarse mediante una ventana de alerta o *alert box*, 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 boton de `Iniciar`, 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.

2.3. Estrategia de programación

Cada grupo es libre de establecer la estrategia con la que abordar la solución del problema. Una sugerencia es no programar todo hasta el mínimo detalle en primera instancia sin hacer ningún tipo de comprobación intermedia, sino ir avanzando poco a poco, resolviendo subproblemas y verificando su correcto funcionamiento, de forma que al final se consiga el resultado deseado: *"divide y vencerás"*.

Ello requiere reflexionar sobre cuáles deberían ser los objetivos parciales a seguir. Por ejemplo: primeramente intentar conseguir la reubicación de cartas de unos tapetes a otros; una vez que las cartas se puedan mover, entonces añadir las reglas que nos permitan o prohiban estos movimientos; a continuación, realizar los ajustes oportunos para mostrar los resultados (tiempos y movimientos); para finalizar, se podría abordar la forma con la que realizar correctamente la *reinicialización* del juego. Insistimos en que esto es una mera sugerencia y cada grupo puede abordarlo de la manera que le parezca más oportuna.

3. Resultados a entregar

Actualícese el repositorio de GitHub con los ficheros que cumplan las especificaciones requeridas a lo largo del enunciado de la presente práctica. Recuérdese las condiciones de la entrega para evitar una merma en la calificación de la práctica.