JavaScript de Andy Harris

Tema 7. - Intercambio de imágenes. Matrices

En este tema, vamos a ver dos de las técnicas más potentes de la programación moderna. El intercambio de imágenes es la posibilidad de cambiar imágenes de forma dinámica. Vamos a ver cómo se usan estos efectos en el diseño Web y cómo se pueden aplicar en el desarrollo de juegos. También veremos cómo los programas pueden almacenar información compleja: las matrices. Las habilidades específicas que vamos a ver son:

- **Utilizar el objeto imagen del DOM.**
- **Cambiar una imagen con la propiedad src.**
- **Usando los controladores de eventos onload, onMouseOver, y onMouseOut.**
- **Creando vectores (*arrays*) de variables y creando matrices.**

Proyecto: Juego de baloncesto

Las Figuras 7.1 a 7.3 muestran el programa “Juego de baloncesto”. El juego permite al usuario jugar una simulación de un partido de baloncesto muy elemental. El usuario controla cinco jugadores que compiten contra un oponente controlado por el ordenador. La interfaz tiene muy pocos controles. El usuario controla la acción haciendo click en los jugadores y en la canasta. La pelota simula moverse de un jugador a otro. Una caja de texto en la parte inferior de la pantalla muestra información. El juego acaba cuando alguién llega a 21 puntos. Tan sencillo cómo parece, este juego necesita ya de un cierto nivel de programación.

Figura 7.1
Figura 7.2
Figura 7.3

Intercambio de imágenes

Ya hemos visto que no es fácil cambiar una página web (aparte de elementos cómo las cajas de texto) una vez que se ha terminado de cargar el documento. Hay una excepción importante a esta regla. Se puede utilizar JavaScript para cambiar las imágenes sobre la marcha. Las Figuras 7.4 y 7.5 muestran un programa, “Intercambio sencillo de imágnes”, que muestra cómo se puede hacer.

Figura 7.4
Figura 7.5

El código HTML

Este código es especialmente interesante, porque muestra tres formas diferentes de manipular imágenes desde JavaScript. Cómo siempre, empezamos con el código HTML:

```
<body>
<h1>Intercambio de imágenes</h1>
<form name="miForm" id="miForm" action=""> <hr />
<table border="1">
<tr> <td colspan="3">
<center>
<img name="imgDisplay" id="imgDisplay" src="./img/triangle.gif"
height="100" width="100" /> </center> </tr>
<tr> <td>
<img src="./img/triangle.gif" height="50" width="50"
name="imgTriangulo" id="imgTriangulo" />  </td> <td>
<img src="./img/circle.gif" height="50" width="50" name="imgCirculo"
id="imgCirculo"/>
</td>
<td>
<img src="./img/square.gif" height="50" width="50" name="imgCuadro"
id="imgCuadro"/>
</td> </tr>
<tr>
<td><input type="button" value="Triangulo"
onclick="mostrarTriangulo()" /> </td> <td>
<input type="button" value="Circulo" onclick="mostrarCirculo()" />
</td> <td>
<input type="button" value="Cuadrado" onclick="mostrarCuadrado()" />
</td> </tr>
</table>
</form>
</body>
```

La página es un formulario básico con un botón en él. La interfaz muestra una imagen grande y varias imágenes pequeñas. Notad que cada imagen tiene un nombre asociado. Cada imagen pequeña tiene un botón debajo. El evento onclick de cada botón llama a una función diferente. Cómo veremos, cada función utiliza una técnica diferente para mostrar la imagen. Vamos a ver las funciones de una en una para ver cómo trabajan.

La técnica del triángulo

El código de la función `mostrarTriangulo()` es:

```
function mostrarTriangulo(){
window.document.imgDisplay.src = "./img/triangle.gif";
} //end mostrarTriangulo
```

El código no puede ser más sencillo. Las imágenes son objetos en el modelo de objetos del documento (*DOM*). Se sitúan justo por debajo del documento. Una imagen se puede referenciar con `document.nombreImagen` y puede tener una propiedad `src`, que puede ser leída y escrita. Si se asigna una nueva URL a la propiedad `src`, el efecto es que cambia la imagen en la página.

CUIDADO – El tamaño de la imagen en pantalla puede ser un problema. Si se pretende cambiar una imagen mediante JavaScript, hay que asegurarse que la etiqueta HTML tiene establecidas la propiedades height y width. De hecho, añadir estas propiedades a una imagen HTML es una buena práctica que nos evitará sorpresas desagradables.

La técnica del círculo

The code for the circle function is slightly different. Here's how it is written:

```
function mostrarCirculo(){
window.document.miForm.imgDisplay.src =
window.document.miForm.imgCirculo.src;
} // end mostrarCirculo
```

El código aquí es similar a la función anterior. Pero hay una diferencia significativa: en lugar de asignar una URL a la propiedad src de `imgDisplay`, la función copia el valor de la propiedad src de `imgCirculo`. Esta técnica puede ser útil si hay una imagen en pantalla que puede ser copiada.

El objeto imagen y el cuadrado

La función `mostrarCuadrado()` usa una técnica diferente. Se crea un objeto para almacenar la imagen y se establece la propiedad `src` cómo la imagen Cuadrado. Este es el código:

```
var imgObjetoCuadrado = new Image(100,100);
imgObjetoCuadrado.src = "./img/square.gif";
function mostrarCuadrado(){
window.document.miForm.imgDisplay.src = imgObjetoCuadrado.src;
} // end mostrarCuadrado
```

Primero, creamos un objeto imagen fuera de las funciones, de forma que esté disponible para otras funciones. Cuando se crea un objeto imagen, se usa la palabra clave `new` y se especifican las propiedades `height` (altura) y `width` (anchura) de la imagen. Después de crear un objeto imagen, podemos hacer referencia a su propiedad `src`. Esta técnica tiene una ventaja muy importante: creamos la imagen sin mostrarla en la página. Con esta técnica podemos cargar una imagen en un objeto y copiar la propiead `src` cada vez que queramos.

Usando los eventos MouseOver

Las Figuras 7.6 y 7.7 muestran otra versión del programa "Intercambio sencillo de imágenes". El programa "Intercambio de imágenes con MouseOver" tiene algunas diferencias con el anterior.

Figura 7.6
Figura 7.7

En primer lugar, la página no tiene botones! La imagen principal cambia cuando el usuario mueve el ratón sobre las imágenes pequeñas. Cuando el ratón no está sobre ninguna imagen la salida se pone en blanco.

Creando imágenes con gestores o manipuladores de eventos (*handler*)

La falta de botones es el cambio más evidente, pero el código también introduce otras mejoras. Echemos un vistazo al código HTML. La única parte diferente es la de las imágenes pequeñas. Todas las imágenes llaman a la misma función, luego las diferencias están en el propio código HTML. Este es el código para el triángulo. Para las demás es muy parecido.

```
<a name="dummy">
…
<tr><td><a href="#dummy" onmouseover="show("imgObjTriangulo")"
      onmouseout = "document.imgDisplay.src='./img/blank.gif'">
<img src="./img/triangle.gif" height="50" width="50"
name="imgTriangulo" border="0" />
</a>
</td>
```

Podemos ver un par de cosas interesantes en este código. El problema principal es este: queremos que la imagen principal cambie cuando pasamos con el ratón por encima de la imagen pequeña del triángulo. Hubiese sido muy fácil, si el objeto imagen tuviese un evento del tipo “el ratón está sobre mí”. Pero las imágenes no tienen ningún tipo de evento. Los desarrolladores web se han inventado una forma de darle la vuelta a este problema. Los enlaces internos (*anchors*) (creados con la etiqueta `<a>`) tienen algunos eventos definidos. Estos son los eventos del objeto anchor:

Tabla 7.1 Eventos del objeto anchor

Evento	**Descripción**
onclick	El usuario hace click en el enlace
ondblclick	El usuario hace doble click en el enlace
onmouseover	El ratón está situado sobre el enlace
onmouseout	El ratón sale del enlace

Estos son los eventos más utilizados. Una característica interesante de los enlaces es que pueden abarcar imágenes o texto. Si queremos conseguir que una imagen responda a eventos podemos conseguirlo incluyéndola en el enlace y aprovechar los eventos de este.

CONSEJO – En este caso concreto, la URL utilizada en el enlace es irrelevante— no queremos ir al enlace, sólo aprovechar los eventos. La solución más elegante (que es la que hemos utilizado) consiste en crear un enlace interno. Si lo situamos lo suficientemente cerca del enlace, el usuario no notará ningún salto interno.

Para mejorar la ilusión, necesitamos que el enlace no sea obvio. Si ponemos la propiedad border a 0 eliminamos el marco azul que aparece en las imágenes con enlace. Hemos añadido un gestor del evento onMouseOver. Este consiste en una llamada a una función que cambiará la imagen. También hemos añadido un gestor de evento onMouseOut. Este gestor hace que la imagen grande cambie a blanco cuando el ratón salga del área del triángulo.

Variaciones de esta técnica se han hecho comunes en el mundo web recientemente. Seguro que ha visto muchas páginas que usan este efecto: cuando el ratón se sitúa sobre una determinada zona de la pantalla, una imagen cambia de aspecto, o cambia el color, etc.

EN EL MUNDO REAL - Variaciones de esta técnica se han hecho comunes en el mundo web recientemente. Seguro que ha visto muchas páginas que usan este efecto: cuando el ratón se sitúa sobre una determinada zona de la pantalla, una imagen cambia de aspecto, o cambia el color, etc. El código será parecido al siguiente:

```
<a href="http://www.whatever.com"
     onmouseover="document.laImagen.src='./img/destacado.gif'"
     onmouseout="document.laImagen.src='./img/normal.gif'">
<img src="normal.gif" name="laImagen" />
</a>
```

Este código muestra una imagen resaltada cuando el ratón se sitúa sobre el enlace. En cuanto el ratón sale, el evento onmouseout devuelve la imagen a su aspecto original. Notad el uso de comillas simples y dobles.

Escribieno JavaScript para cambiar las imágenes

Ahora que sabemos cómo dotar a una imagen de gestores de eventos, veamos el codigo que lo aprovecha. Este es el código JavaScript:

```
<script type="text/javascript">
//crear los objetos imagen
var imgObjCuadrado = new Image(100,100);
var imgObjTriangulo = new Image(100,100);
var imgObjCirculo = new Image(100,100);
//inicializar los objetos imagen
imgObjCuadrado.src = "./img/cuadrado.gif"
imgObjCirculo.src = "./img/circulo.gif"
imgObjTriangulo.src = "./img/triangulo.gif"

function mostrarImagen(imgMostrar){
//recibe un objeto imagen cómo argumento
//copia el valor a la salida
document.imgDisplay.src = imgMostrar.src;
} // end mostrarImagen
</script>
```

Antes de empezar la función, hemos creado una serie de objetos de tipo imagen y los hemos inicializado. El código de la función hace referencia a esos objetos.

Usando parámetros en las funciones

Hubiera sido posible hacer una función para cada imagen. El código hubiese sido idéntico excepto en el valor de la imagen a mostrar. A los programadores no les gusta repetir código que hace lo mismo y tratan de evitarlo. Imagine que en lugar de 3 imágenes tuviesemos 20, y que nos damos cuenta que queremos modificar algo de la función. Tendríamos que corregirlo en 20 sitios. Esto es lo que tratan de evitar los programadores experimentados. Se puede utilizar una

única función `mostrarImagen`, gracias a que esta contiene un **parámetro**. Hasta ahora, hemos utilizado las funciones con paréntesis vacíos. Es decir, sin parámetros. Los parámetros nos van a permitir pasarle valores a la función. Cambiando estos valores vamos a conseguir que la función haga cosas diferentes. Aunque pocas, algunas funciones con argumentos hemos visto. Por ejemplo `alert()`. El valor que le pasamos entre paréntesis es el mensaje que después se mostrará al usuario. Para especificar que la función va a trabajar con un parámetro, basta con escribir el nombre de una variable en la definición de la función, y cuando la usemos, especificar un valor entre paréntesis.

Veamos cómo se usan los parámetros en la función `mostrarImagen()`: la función se usa siempre para copiar la propiedad `src` de un objeto imagen a la imagen `imgDisplay`. El problema es que la imagen específica que hay que copiar varía en función de la imagen pequeña sobre la que pasa el ratón. Los eventos `onMouseOver` especifican el nombre del objeto a mostrar. Cuando el usuario pasa el ratón sobre la imagen del triángulo, su evento `onMouseOver` dispara la función `mostrarImagen()` con el valor `imgObjTriangulo` cómo parámetro. Del mismo modo, la imagen cuadrado envía el objeto `imgObjCuadrado`, y así sucesivamente. Cuando la función `mostrarImagen()` se ejecuta, cualquier valor pasado cómo parámetro se guarda en la variable `imgMostrar`. La propiedad `src` de cualquier objeto de imagen que contenga la variable `imgMostrar` se copia al de la imagen a mostrar. Los parámetros se usan para hacer que una función sea más flexible y pueda responder a distintas entradas.

Creando vectores (*Arrays*)

Los programas se van haciendo más complejos, y empiezan a tener un número cada vez mayor de varialbes. Cuando se necesita almacenar una serie de valores del mismo tipo se usan estructuras llamadas vectores (*arrays*). Recordemos que los hemos visto ya en el Tema 5. En ese tema, construimos una serie de componentes HTML con el mismo nombre que se convertían automáticamente en un vector. También podemos crear vectores de cualquier tipo de variable JavaScript. Las Figuras 7.8 y 7.9 muestran un sencillo programa, “Demo Vectores”, que utiliza un vector de cadenas de caracteres. El programa cambia de una imagen en blanco a triángulo, círculo y cuadrado y de nuevo a blanco cuando el usuario hace click en el botón.

Figura 7.8
Figura 7.9

El código HTML consiste en un área de texto, llamada `txtDescripcion`, y de un botón, que llama a la función `actualiza()` cuando se le hace click. Este es el programa JavaScript:

```
var descripcion = new Array(3);
var contador = 0;
function inicializa(){
// inicializa el vector con valores iniciales
descripcion[0] = "blanco";
descripcion[1] = "triangulo";
descripcion[2] = "circulo";
descripcion[3] = "cuadrado";
} // end inicializa

function actualiza(){
//incrementa el contador y muestra la descripción
counter++;
```

```
if (counter > 3){
counter = 0;
} // end if
document.miForm.txtDescripcion.value = descripcion[contador];
} // end  actualiza
```

Creando un variable vector (*Array*)

La primera línea de código crea una variable llamada descripcion:

```
var descripcion = new Array(3);
```

Esta variable es especial, ya que almacena un vector. Para crear un vector, usamos la sintaxis `new Array()`. El valor entre paréntesis indica el número de elementos del vector. JavaScript no requiere en principio ningún número y la línea siguiente funciona también:

```
var descripcion = new Array();
```

Cómo la mayoría de lenguajes requieren que se declare el número de elementos que va a contener el vector, es conveniente hacerlo aunque JavaScript no lo exija.

CUIDADO – No olvide utilizar el término Array con mayúscula! Este es un error muy fácil de cometer y muy difícil de depurar.

Rellenando el Array

Una vez creado el vector, es esencialmente una lista de variables. De alguna manera vamos a necesitar rellenar el array con valores. En este programa, hemos elegido usar una función especial llamada `inicializa()`. La función contiene una serie de sentencias de asignación que rellena el vector con los valores adecuados. Los vectores en JavaScript son muy flexibles. Se pueden mezclar los tipos de las variables en el mismo vector (algo que en otros lenguajes está totalmente prohibido), además se pueden añadir elementos una vez creado el vector.

CUIDADO – Tened en cuenta que el comportamiento de JavaScript respecto a los arrays no es habitual. A veces puede ser interesante sacar partido de estas ventajas, pero estas facilidades son también muy peligrosas y pueden ser muy difíciles de depurar.

La función `inicializa()` debe ejecutarse pronto. Hemos elegido usar un evento especial que se dispara cuando se carga la página y llamar a la función ahí. La etiqueta <body> de HTML tiene un evento `onLoad`. Cualquier función que se asocie con ese evento se ejecutará en cuanto el navegador cargue el body del documento. Esto lo convierte en un lugar perfecto para colocar las inicializaciones. Aunque no hemos mostrado el código HTML, esta es la línea importante:

```
<body onload="inicializa()">
```

EN EL MUNDO REAL – En cualquier programa que requiera que se ejecute algo antes de que el usuario tome el control del entorno, puede ser interesante utilizar una función inicializa() de este tipo.

Mostrando el elemento siguiente

El botón llama a la función `actualiza()`. El trabajo de esta función es mostrar el siguiente elemento del vector. Para hacerlo, la función guarda la pista en una variable llamada `contador`. Tened en cuenta que `contador` se crea fuera de cualquier función, para que se guarde el valor entre llamada y llamada a la función. Cada vez que el usuario hace click en el botón Siguiente, el programa incrementa la variable `contador`. El programa comprueba que el valor de contador no es demasiado grande. Si se ha pasado, el programa pone el `contador` a 0. Por último, el programa actualiza la caja de texto con el elemento correspondiente del vector.

CONSEJO – Siempre que incremente o decremente una variable, debe considerar comprobar si se ha producido cualquier desbordamiento de los límites en el valor de la variable, y tomar las acciones adecuadas si ha pasado.

Creando vectores con imágenes

JavaScript soporta vectores de cualquier tipo de variables, incluidas las imágenes. Las Figuras 7.10 y 7.11 muestran una versión mejorada llamada “Demo vector de imágenes”. Esta versión soporta gráficos. Este programa utiliza dos vectores. Uno contiene las descripciones y el otro las imágenes. Hemos diseñado los vectores de forma que estén exactamente en el mismo orden, de forma que imagen[1] se refiera a lo mismo que descripcion[1].

Figura 7.10
Figura 7.11

Escribiendo el código HTML para el programa “Demo vector de imágenes”

El código HTML para esta versión del programa es relativamente sencilla:

```
<body onload="inicializa()">
<center> <h1>Demo Vector imágenes<hr /></h1>
<form name = "miForm">
<img src="./img/blanco.gif"
name="imgDisplay" height="100" width="100" /> <br>
<input type="text" name="txtDescripcion" value="transparente" /> <br>
<input type="button" value="Siguiente" onclick="actualiza()" />
</form> </center> <hr>
</body>
```

Lo más destacado son la función de inicialización asociada al evento onload del body y el objeto de imagen llamado imgDisplay.

Creando e inicializando los vectores

El programa “Demo vector de imágenes” contiene un vector de cadenas de caracteres y un vector de imágenes. El código inicializa los dos vectores de forma diferente. Echemos un vistazo al código:

```
var descripcion = new Array("transparente", "triángulo", "círculo",
"cuadrado");
var imagenes = new Array(3);
```

```
var contador = 0;
function inicializa(){
// inicializa el array con los valores iniciales
imagenes[0] = new Image(50, 50);
imagenes[0].src = "./img/transparente.gif";
imagenes[1] = new Image(50, 50);
imagenes[1].src = "./img/triangulo.gif";
imagenes[2] = new Image(50, 50);
imagenes[2].src = "./img/circulo.gif";
imagenes[3] = new Image(50, 50);
imagenes[3].src = "./img/cuadrado.gif";
} // end inicializa
```

La variable `contador` se crea cómo de costumbre, y el vector `imagenes` también se crea cómo hemos visto, pero la inicialización del vector `descripcion` es algo diferente. JavaScript permite definir vectores sobre la marcha. Cómo ya sabemos los valores que queremos almacenar, simplemente inicializamos el vector al crearlo. Los objetos de tipo imagen necesitan un poco más de trabajo, primero necesitamos crear el objeto y posteriormente asignar un valor a su propiedad `src`.

Actualizando la página

La función `actualiza()` es muy similar a la del programa “Demo Vectores”, pero hemos añadido una línea para actualizar la imagen. Este es el código de la función:

```
function actualiza(){
//incrementa el contador y muestra la siguiente descripción
counter++;
if (counter > 3){
counter = 0;
} // end if
document.imgDisplay.src = imagenes[contador].src;
document.miForm.txtDescripcion.value = descripcion[contador];
} // end actualiza
```

La nueva línea es la que hace referencia a `imgDisplay`. Coge el elemento del vector `imagenes` referenciado por la variable `contador` y copia el valor de la propiedad `src` a la propiedad `src` de `imgDisplay`. Consiguiendo que `imgDisplay` muestre la imagen actual.

Usando Matrices

Los vectores pueden ser muy útiles cuando tenemos que manejar gran cantidad de datos. Cómo ejemplo, pensemos en el programa “Juego de baloncesto” que hemos visto al principio del tema. Para mostrar la utilidad de los vectores vamos ha crear un programa más sencillo “Demostración de Matrices” que va a determinar la probabilidad de que un jugador tenga éxito al realizar un pase a otro jugador o un tiro.

Creando el programa “Demostración de matrices”

Las Figuras 7.12 y 7.13 muestran el programa “Demostración de matrices”. La lista desplegable permite al usuario seleccionar un tirador (el jugador que tiene el balón) y el objetivo (la canasta

o el jugador al que el usuario quiere pasar el balón). Cuando el usuario hace click sobre el botón Lanzar, en la salida se muestra la probabilidad (expresada en porcentaje) de que la jugada tenga éxito.

CONSEJO – Cuando se están escribiendo programas complejos, es conveniente aislar las nuevas ideas en programas de prueba más pequeños, para asegurarnos de que cada concepto funciona correctamente de forma aislada antes de juntarlos de forma más compleja.

Vamos a ver el código del programa enseguida, pero es importante entender los conceptos en que está basado antes.

Figura 7.12
Figura 7.13

Codificando los porcentajes de un jugador

Imaginémos que el base tiene la pelota. El jugador puede escoger entre pasar la pelota a cualquier otro de los cuatro jugadores de su equipo o intentar un tiro a canasta (al menos en el juego de baloncesto simplificado que estamos modelando). Es muy probable que el base acierte un pase al jugador más próximo. Es poco probable que enceste del otro lado de la cancha. Podemos resumir las probabilidades (en porcentaje) de cada una de las opciones del jugador:

Jugador Base

Canasta	.01
Pivot	.05
Ala-pivot	.10
Alero	.30
Escolta	.90

Esta tabla describe la probabilidad de acierto del base pasando la pelota a los demás jugadores. O sea, estamos diciendo que la probabilidad de encestar es del 1% (.01). La probabilidad de completar un pase al pivot es del 5%, un 10% de completar un pase al ala-pivot, un 30% de probabilidad de completar un pase al alero, y un 90% de probabilidad de completar un pase al escolta. Sabemos que la función random() nos devuelve un número aleatorio entre 0 y 1, por tanto si queremos simular si un pase con un 30% de probabilidad se ha completado, podemos generar un número aleatorio entre 0 y 1 y compararlo con 0.3. Podemos almacenar esta información en un vector de la siguiente manera:

```
var baseVals = new Array(.01, .05, .10, .30, .90);
```

A continuación, si queremos saber si ha habido acierto en el pase al pivot podemos usar el código:

```
//base pasando a pivot
if (Math.random() < baseVals[1]){
alert("Pase del base al pivot completado!");
} // end if
```

Añadiendo los demás jugadores

Tenemos cinco jugadores en la cancha, luego necesitamos cinco vectores para gestionar todos las posibles jugadas. Hemos hecho una tabla en la que mostramos todas las jugadas con su probabilidad:

Jugador	Canasta	Pivot	Ala-pivot	Alero	Escolta	Base
Pivot	.5	-1	.7	.3	.8	.8
Ala-pivot	.3	.6	-1	.7	.8	.8
Alero	.25	.5	.7	-1	.8	.8
Escolta	.05	.4	.5	.7	-1	.8
Base	.01	.05	.1	.3	.9	-1

Cada fila representa un jugador, y cada columna un objetivo. Cada celda contiene la probabilidad de que un jugador consiga su objetivo. Por ejemplo, para saber cuál es la probabilidad de que un alero complete un pase al pivot, miramos en la fila del alero y en el cruce con la columna del pivot encontramos la probabilidad que en este caso es del 50%. El pivot tiene una probabilidad del 50% de encestar cuando tira a canasta. Hemos decidido que un jugador no se pasa a sí mismo y la probabilidad de esa jugada la hemos marcado con –1.

CONSEJO – No hay ningún método científico detrás de la asignación de estos valores. Si queremos hacer un juego mucho más real, tendríamos que buscar estadísticas de la NBA o de la ACB para que fuesen más correctos.

Codificando la Tabla

Esta estructura bidimensional se llama tabla o matriz. Para crearla en JavaScript, se crean una serie de vectores (*array*) uno por fila y se juntan en un vector mayor. Un vistazo al código fuente nos ayudará a comprender mejor cómo funciona. Para empezar hemos creado un conjunto de variables para representar a los jugadores en la cancha:

```
//definimos unas constantes para las posiciones
var CT = 0; //canasta
var PV = 1; //pivot
var AP = 2; //ala-pivot
var AL = 3; //alero
var EC = 4; //escolta
var BS = 5; //base
```

Vamos a crear varios vectores, y en cada uno el elemento 0 representará la canasta y el quinto el base. Para hacer el código más fácil de entender, le hemos dado nombre a las variables con abreviaturas de las posiciones. A veces, necesitaremos el nombre completo del tipo de jugador, luego nuestro primer vector va a ser una lista de los nombres de los tipos de jugador:

```
var plNombre = new Array( "canasta", "pivot", "ala-pivot", "alero",
"escolta", "base");
```

Fijaros que el nombre del jugador están en el mismo orden que las constantes, de forma que plNombre(BK) es “canasta”. También se podría modificar el nombre de los jugadores en el vector de forma que contenga el de algún equipo de baloncesto. A continuación, vamos a crear una serie de vectores para codificar los porcentajes de cada jugador:

```
//definir los vectores para almacenar las filas
var PVValores = new Array(.50, -1, .70, .30, .80, .80);
var APValores = new Array(.30, .60, -1, .70, .80, .80);
var ALValores = new Array(.25, .50, .70, -1, .80, .80);
var ECValores = new Array(.05, .40, .50, .70, -1, .80);
var BSValores = new Array(.01, .05, .10, .30, .90, -1);
```

Cada vector especifica una fila de la tabla original. El orden de los valores en el vector es muy importante. Si queremos saber cuál es la probabilidad de que el escolta complete un pase al pivot, podemos escribir una sentencia cómo:

```
alert(ECValores[PV]);
```

Por último, vamos a combinar los vectores de cada jugador en un vector mayor.

```
var todosValores = new Array (0, PVValoress, APValores, ALValores,
ECValores, BSValores);
```

Notad que los elementos del vector son otros vectores. Asimismo, el valor 0 está sólo para ocupar un plaza, ya que la canasta no va tirar a sí misma. Ahora que todos los datos de las probabilidades de todos los jugadores están almacenadas en una variable llamada `todosValores`. Podemos obtener cualquier valor de la tabla haciendo referencia al vector de vectores `todosValores`. Por ejemplo:

```
alert(todosValores[BS][PV]);
```

nos devolverá la probabilidad de que se complete un pase del base al pivot. El valor `[BS]` va a especificar el quinto elemento de todosValores (recordar que `BS`, es un atajo para 5). Dado que el quinto elemento del vector `todosValores` es otro vector, podemos preguntar por el valor `[PV]` o primer valor de ese vector. Este tipo de estructura (vector de vectores) es un array bidimensional o matriz. Otros lenguajes tienen mecanismos para generar las matrices directamente, pero de un modo u otro la mayoría de lenguajes las soportan.

EN EL MUNDO REAL – Las matrices son de gran utilidad cuando queremos trabajar con información que se representa en forma de tablas. Pueden ser tablas de precios, de descuentos, de impuestos, etc. De hecho, las matrices no están limitadas a dos dimensiones, colocando una matriz dentro de un vector obtendríamos una matriz de tres dimensiones y así sucesivamente.

Obteniendo valores de una Matriz

Es muy fácil obtener información de una matriz una vez creada e incializada. Esta es la función lanzaBola() del programa “Demostración de matrices”:

```
function lanzaBola(){
// Demostración de matrices
```

```
// alert("estamos en lanzaBola");
var marcador = document.miForm.txtSalida;
var tirador;
var objetivo;
var rndValor = Math.random();
var resultado;
//seleccionado = document.miForm.selTirador;
tirador = document.miForm.selTirador.selectedIndex;
objetivo = document.miForm.selObjetivo.selectedIndex;
if (rndValor < todosValores[tirador][objetivo]){
resultado = "Lo conseguiste !!";
} else {
resultado = "Fallado";
} // end if
marcador.value = "Tirador: " + tirador + " " + plNombre[tirador] +
"\n";
marcador.value += "Objetivo: " + objetivo + " " + plNombre[objetivo] +
"\n";
marcador.value += "Porcentaje: " + todosValores[tirador][objetivo] +
"\n";
marcador.value += "Resultado: " + rndValor + "\n" + resultado;
} // end  lanzaBola
```

La función genera variables para el tirador, el objetivo y un valor aleatorio. Los valores de tirador y objetivo se extraen de dos objetos lista desplegable. La línea siguiente es la clave:

```
if (rndValor < todosValores[tirador][objetivo]){
```

Se compara el valor aleatorio generado con la probabilidad correspondiente almacenada en la matriz. Si la condición se evalúa a verdadera, el programa considera que el pase o el tiro se ha completado. El resto del código sólo envía información al cuadro de texto para informar de lo que está sucediendo.

EN EL MUNDO REAL – Los programadores novatos se sienten, a veces, superados por las nociones de tablas y matrices. De hecho se puede escribir este programa sin usarlas, sustituyendo el código por estructuras switch. Pero esto supondría unas 130 líneas de código con hasta cinco niveles de anidamiento en los if. Incluso para un programador experimentado se convierte en una pesadilla. Esta versión con matrices necesita de 15 líneas de código. Además todos los datos de probabilidades están en el mismo sitio con lo que sería muy fácil modificarlos. A veces, cómo en este caso el uso de una estructura más elaborada al final nos va a simplificar el programa.

Completando el “Juego de Baloncesto”

El “Juego de Baloncesto” es sencillamente una combinación de las ideas que hemos visto hasta ahora en el Tema. La base es la estructura de matriz que acabamos de ver. Hemos añadido una interfaz gráfica y la posibilidad de almacenar un marcador para convertirlo en un juego.

La ventana del juego

El aspecto gráfico del juego es muy importante, debe parecer una simulación de un partido de baloncesto. Hemos empezado con el diseño de una cancha de baloncesto en nuestro programa

de dibujo. La cancha se usará cómo imagen de fondo, las demás imágenes (jugadores) se superpondrán sobre ella. El aspecto desagradable de las imágenes de fondo es que si son menores que el tamaño de la pantalla, se repiten hasta rellenar todo el fondo. Si son demasiado grandes, no se verá la imagen completa de la cancha. Necesitamos asegurarnos el tamaño de la ventana del juego, por lo que hemos decidido usar una ventana secundaria. La primera ventana que ve el usuario, sólo tiene un botón. Este botón hace emerger una ventana secundaria con el tamaño adecuado a la imagen de fondo y al marcador. Cómo el usuario no podrá redimensionar la nueva ventana, hemos resuelto el problema del tamaño de la imagen de fondo. Este es el código que hace aparecer la ventana del juego:

```
function empiezaJuego(){
// Baloncesto
// carga la imagen de una cancha de baloncesto en una nueva ventana
var cancha = window.open("bball.html", "bbAnch", "height=450,
width=300");
cancha.focus();
} // end  empiezaJuego
```

Mostrar el juego en una nueva ventana es una técnica muy habitual. No tenemos control de la configuración del navegador cuando el usuario accede por primera vez a la página. Sólo podemos controlar una ventana que creamos. Cuando la creamos podemos determinar su tamaño exacto y asegurarnos que el usuario no puede redimensionarla.

El diseño gráfico

La cancha de baloncesto es la imagen de fondo de la página principal del juego. Además, tenemos cinco figuras para las cinco posiciones de los jugadores. Conseguir que las imágenes de los jugadores se coloquen en la posición que queremos no es fácil usando únicamente técnicas de HTML estándar. Para posicionar las imágenes correctamente hemos utilizado algunos trucos de diseño HTML. La Figura 7.14 muestra la página del juego con algunas modificaciones para mostrar cómo vamos a colocar las figuras. Cada jugador va a estar en una tabla. Las tablas tienen una fila con una imagen vacía y una imagen de un jugador. El la Figura 7.14, hemos cambiado la imagen vacía por una imagen toda en blanco, para que se vea mejor cómo se hace el espaciado. Cambiando el tamaño de la imagen “espaciadora”, obtenemos un cierto control de la posición en la que se colocará el jugador. Por ejemplo, la altura de la imagen en blanco sobre la canasta asegurará que el ala-pivot esté situado debajo de la canasta. La anchura de la imagen hacia la izquierda asegurará que se coloca a la derecha de la canasta. Por supuesto, una vez que consigamos colocar los jugadores correctamente, sustituiremos las imágenes en blanco por unas transparentes.

EN EL MUNDO REAL - HTML es un lenguaje muy expresivo. El programador puede confiar razonablemente que el contenido de su página se muestre en una amplia gama de plataformas. Pero la flexibilidad interplataformas tiene un coste. Es muy difícil controlar cómo se colocan exactamente los elementos en una página. En el Tema 8, “HTML dinámico: El submarino fantasma”, usaremos la tecnología de las hojas de estilo (CSS) para posicionar los elementos. En este tema, vamos a utilizar sólo las viejas técnicas de HTML. Los programadores de HTML descubrieron rápidamente que el uso de las tablas permitia soslayar el problema de posicionar elementos en la pantalla. Muchos de los formularios de este libro se crean haciendo uso de las tablas. Es teóricamente posible determinar la altura y anchura de la celda de una tabla en pixels. Parece que con esto los programadores HTML tendrían solucionado el problema de posicionar elementos en la pantalla, pero los

navegadores son muy inconsistentes en la forma de soportar las tablas. También es posible determinar el tamaño de una imagen en pixels, y la mayoría de los navegadores lo hace bien. Por tanto, los programadores HTML ha utilizado la técnica de la imagen transparente para colocar elementos en su páginas web. Construyendo una tabla con imágenes transparentes podemos obtener un control muy eficiente del aspecto de la pantalla sin usar CSS.

Figura 7.14

Este es el código HTML que muestra la forma de colocar las imágenes:

```
<body background =./img/cancha.gif" onload="resetJuego()">
<form name="miForm">
<!- canasta y pivot -->
<table border="0">
<tr><td> <img src="./img/transparente.gif" height="20" width="130" />
<a href="javascript:lanzarA(CT) ">
<img src="./img/transparente.gif" name="img0"
height="30" width="20" border="0" /> </a>
<a href="javascript:lanzarA(PV) ">
<img src="./img/jugador.gif" name="imgJugador1"
height="40" width="40" border="0" /> </a>
</td> </tr>
</table>
<!- Ala-pivot -->
<table border = 0>
<tr><td> <img src="./img/transparente.gif"
height="20" width="180" />
<a href="javascript:lanzarA(AP)">
<img src="./img/jugador.gif" name="imgJugador2"
height="40" width="40" border="0" /> </a> </td></tr>
</table>
```

Este trozo de código gestiona sólo las dos primeras tablas, pero muestra la técnica que se usará para todas. Notad que hemos eliminado el marco de las imágenes para que los gráficos aparezcan integrados en la página. Fijarse en el uso de `javascript:lanzarA()` cómo el `href` de las etiquetas de enlace. En cualquier lugar que se puede poner `url` o `href`, podemos poner también "`javascript:`" y una línea de código JavaScript. En este caso hemos llamado a la función `lanzarA()`. Todas las imágenes llaman a la misma función pero con distinto parámetro.

CONSEJO – En futuros temas, veremos el uso de las hojas de estilo para posicionar elementos. Pero no está mal comprobar que estas "viejas" técnicas siguen funcionanado para ciertas ocasiones.

La imagen del jugador es muy tosca pero nos vale para el trabajo (ver Figura 7.15). Hemos dibujado una imagen de 100×100 de resolución con un fondo transparente. Tenemos dos versiones jugador con y sin bola.

Variables globales

La matriz es la clave de este programa. Hemos empezado por su definición al igual que hicimos en el programa “Demostración de matrices”. También hemos añadido variables para guardar el marcador:

```
//definir las variables principales
var marcadorJugador;
var oponenteMarcador;
var jugadorActual;
//definir las constantes para las posiciones
var CT = 0; //canasta
var PV = 1; //pivot
var AP = 2; //ala-pivot
var AL = 3; //alero
var EC = 4; //escolta
var BS = 5; //base
//define el vector para los nombres de los jugadores
var plNombres = new Array( "canasta", "pivot", "ala-pivot", "alero",
"escolta", "base");
//definir los vectores para almacenar las filas
var PVValores = new Array(.50, -1, .70, .30, .80, .80);
var APValores = new Array(.30, .60, -1, .70, .80, .80);
var ALValores = new Array(.25, .50, .70, -1, .80, .80);
var ECValores = new Array(.05, .40, .50, .70, -1, .80);
var BSValores = new Array(.01, .05, .10, .30, .90, -1);
//crea la matriz
var todosValores = new Array (0, PVValores, APValores, ALValores,
ECValores, BSValores);
```

Figura 7.15

La función resetJuego()

El programa debe inicializar todas las variables a valores iniciales correctos. Necesitará ejecutar el mismo código gane o pierda el jugador. Por tanto, hemos puesto el código para reinicializar el juego en una función `resetJuego()`. Este es el código de la función:

```
function resetJuego(){
//reinicializa todas las variables del juego
marcadorJugador = 0;
oponenteMarcador = 0;
jugadorActual = BS;
document.miForm.tableroMarcador.value = "";
actualizaPantalla(jugadorActual);
} // end resetJuego
```

El código establece los valores necesarios en las variables clave par iniciar el juego. `TableroMarcador` es una referencia al cuadro de texto que muestra la información al usuario. La función acaba con una llamada a la función `actualizaPantalla()`. Veremos esta función en un momento.

La función actualizaMarcador()

Otra función que se llamará cada vez que un jugador o el ordenador enceste una canasta. Este es el código:

```
function actualizaMarcador(){
var tarjetaMarcador = "";
tarjetaMarcador += "Jugador \tOrdenador \n";
tarjetaMarcador += jugadorMarcador;
tarjetaMarcador += "\t" + oponenteMarcador;
document.miForm.tableroMarcador.value += tarjetaMarcador;
//comprobar si hay ganador
if (oponenteMarcador >= 21) {
alert ("Has PERDIDO !!!");
resetJuego();
} else if ( jugadorMarcador >= 21) {
alert ("Has GANADO !!!");
resetJuego();
} // end if
} // end actualizaMarcador
```

La función crea una cadena de texto con la información del marcador. Cómo se ejecuta después de cada enceste, es un buen sitio para comprobar si alguién ha ganado.

La función actualizaPantalla()

El programa llama a la función actualizaPantalla() cada vez que la pelota cambia de manos. El jugador que recibe la pelota se pasa como parámetro a la función. Este es el código:

```
function actualizaPantalla(jugadorActual){
//Borra todas las imágenes, muestra el jugador actual con la pelota
//Borra todas las imágenes
for (i=1; i<=5; i++){
var imgJugador = eval("document.imgJugador" + i);
imgJugador.src = "./img/jugador.gif"
} // end for
//muestra imagen actual
imgJugador = eval("document.imgJugador" + jugadorActual);
imgJugador.src = "./img/bola.gif";
window.status = plNombres[jugadorActual];
} // end actualizaPantalla
```

El programa coloca la imagen (sin balón) en cada posición, a continuación coloca la imagen con balón en la posición especificada por el parámetro `jugadorActual`. Fijarse en el uso de l sentencia `eval`. Las imágenes de los jugadores se llaman imgJugador1, imgJugador2 y así sucesivamente. Concatenamos la cadena `document.imgJugador` con la variable i (en el bucle esta variable tomará los valores del 1 al 5) y `jugadorActual`. A continuación, usamos la sentencia `eval` para forzar al ordenador a evaluar la cadena resultante cómo una sentencia JavaScript. La evaluación genera un objeto imagen, que se guarda en la variable `imgJugador`.

La función lanzarA()

Cada vez que el usuario hace click en una de las imágenes de un jugador, el programa llama a la función `lanzarA()`. El trabajo de esta función es determinar si el pase o el tiro se han completado con éxito. Este es el código de la función:

```
function lanzarA(receptor){
if (Math.random() < todosValores[jugadorActual][receptor]){
//pase correcto o tiro
//comprobar enceste
if (receptor == CT){
document.miForm.tableroMarcador.value = "CANASTA!!! \n";
jugadorMarcador += 2;
//comprobar los tiros de tres puntos!
if ( jugadorActual >= AL){
document.miForm.tableroMarcador.value += "Canasta de 3 puntos! \n";
playerScore +=1;
} // end if
actualizaMarcador();
jugadorActual = BS;
} else {
document.miForm.tableroMarcador.value = "pase completado a " +
plNombres[receptor] + "\n";
jugadorActual = receptor;
} // end if enceste
} else {
//pase o tiro fallado
jugadorActual++;
document.miForm.tableroMarcador.value = "capturada!!";
if (jugadorActual > BS){
jugadorActual = BS;
} // fin marcador oponente
//dar al oponente la oportunidad de encestar
if (Math.random() < 0.20){
document.miForm.tableroMarcador.value += "Ordenador encesta!! \n";
oponenteMarcador += 2;
actualizaMarcador();
jugadorActual = BS;
} // end if
} // end if
actualizaPantalla(jugadorActual);
} // end funcion lanzarA
```

La función espera recibir un `receptor` cómo parámetro. Todas las llamadas a esta función incluyen el valor del jugador que se supone recibe la bola. Lo primero es determinar si ha habido canasta o no. Esto se hace en la siguiente línea de código.

```
if (Math.random() < todosValores[jugadorActual][receptor]){
```

`jugadorActual` es una variable global, y contendrá siempre el valor del jugador que actualmente tiene la bola. El código hace una referencia a la matriz y recupera el valor de la probabilidad de la combinación pasador-receptor. El programa compara este valor con un número aleatorio. Si el pase ha tenido éxito, se debe comprobar si era un intento de tiro a canasta. Esto es fácil, sólo es un tiro si el receptor es la canasta. En los demás casos, se trata de

un pase a otro jugador. Si el intento de canasta tiene éxito, el programa debe incrementar el marcador en dos puntos. Una comprobación más determina si el jugador estaba detrás de la línea de tres puntos. Si es así, el programa suma otro punto. Cómo el tanteo ha cambiado, el programa llama a la función `actualizaMarcador()`.

Table 7.2 – Resumen de sintáxis

Sentencia	**Descripción**	**Ejemplo**
`varImg = new Image (height,width)`	Crea un nuevo objeto Imagen llamado varImg con la altura y anchura especificadas.	`var miImagen = new Image(100,100);`
`varImg.src = URL`	Asigna la URL a la propiedad src de la imagen. URL deben apuntar a un fichero de imagen válido.	`miImagen.src=”./img/cara.gif”;`
`varNombre=new Array()`	Crea una nueva variable de tipo vector llamada varNombre	`var miVector = new Array();`
`varNombre=new Array(length)`	Crea una nueva variable de tipo vector llamada varNombre con el número de elementos especificado	`var miVector = new Array(3);`
`varNombre=new Array (valorA,valorB,valorC)`	Crea una nueva variable de tipo vector llamada varNombre inicializando los elementos con los valores especificados	`var miVector = new Array(“pequeño”,”mediano”,”grande”);`

Si el receptor no es la canasta, el intento era un pase. En este caso, el programa simplemente envía un mensaje al `tableroMarcador` indicando el pase y cambia el valor de la variable `jugadorActual` para reflejar el jugador que tiene la pelota. Si se falla el pase o el tiro, el programa mueve la pelota una posición hacia atrás. Por ejemplo, si el alero falla el tiro, la pelota pasa al escolta. Si el pasador era el base, hemos decidido dejarle la pelota. Cada vez que un tiro o un pase son bloqueados, el oponente tiene la oportunidad de hacer un tiro a canasta. Hemos decidido darle un 20% de probabilidad de encestar. Este valor parece que iguala bastante el juego. El jugador gana las veces necesarias para seguir interesado en el juego, pero no gana siempre. Cambiar este valor es la forma más sencilla de hacer el juego más difícil. Si se incrementa el oponente ganará con más facilidad. Si se hace menor, el jugador ganará más fácilmente.

Resumen

En este tema, hemos visto dos conceptos muy interesantes. En primer lugar el de intercambio de imágenes. Hemos visto el objeto `Image` de JavaScript, y cómo manipular la propiedad src de forma que el objeto muestre nuevas imágenes en pantalla. También hemos visto cómo añadir gestores de eventos a una imagen mediante la etiqueta `<a>`, tambień hemos visto cómo hacer las

funciones más flexibles mediante los parámetros. El otro concepto importante ha sido la creación y el uso de matrices de elementos. Hemos visto cómo crear vectores de variables normales y de imágenes. Hemos visto código que nos permite recorrer un vector elemento por elemento, y por último hemos aprendido a crear matrices. También hemos visto algunos trucos gráficos para crear el programa de Baloncesto. Por ejemplo, cómo el crear una ventana nueva nos permite controlar el tamaño y que el usuario no pueda redimensionarla. También hemos visto cómo usar imágenes transparentes para posicionar imágenes en tablas. Por último, hemos creado una simulación de juego de baloncesto relativamente elaborada y hemos visto un buen número de funciones que nos han permitido crear el juego de forma más organizada. En el tema siguiente, veremos cómo usar las hojas de estilo y otros conceptos avanzados de HTML para incorporar movimiento y sonido en nuestros programas JavaScript.

EJERCICIOS

7.1. Modificar el juego de Baloncesto de forma que refleje los nombres de los jugadores de su equipo favorito.

7.2. Añadir niveles de dificultad al juego. Cada vez que gana el jugador se incrementa el nivel de dificultad incrementando algo el porcentaje de acierto del oponente. Mostrar al usuario el nivel en el que está jugando.

SUPEREJERCICIO 7 (0.5 puntos)

Realizar un nuevo juego de Baloncesto en el que en lugar de un equipo aparezcan 2 (usar cancha en horizontal). Las imágenes de los jugadores serán fotos de jugadores reales. El jugador jugará de la misma forma que en el programa del tema, pero el oponente (ordenador) jugará con el otro equipo.
Cuando la jugada no se completa, en lugar de que el oponente intente un tiro a canasta se pasa el turno al otro jugador y tiene la bola el base. El ordenador decide a qué jugador le pasa la pelota de forma aleatoria aunque las probabilidades deben ser diferentes en función de qué jugador tiene la bola. (Ajustar los valores de las probabilidades de forma que el ordenador sea un rival fácil, de nivel medio y por último muy fuerte).
Para que nos dé tiempo a ver las jugadas del ordenador es necesario incluir un temporizador de 2 segundos en cada jugada del ordenador. (Ver temporizadores)