JavaScript de Andy Harris

TEMA 8. HTML dinámico: El submarino silencioso

Hasta ahora hemos usado técnicas muy probadas. Todos los programas que hemos visto funcionarán incluso con los navegadores más antiguos. Para hacer programas más sofisticados, necesitamos llevar JavaScript hasta sus últimas capacidades. En este tema vamos a hacer algo de esto. Concretamente vamos a:

- **Escribir código independiente del navegador.**
- **Escribir un rutina que permita detectar el navegador.**
- **Posicionar elementos con hojas de estilo (CSS).**
- **Mover elementos flotantes.**
- **Cambiar el texto en elementos flotantes.**
- **Trabajar con ficheros de sonido.**

Todas estas técnicas (y algunas más) constituyen lo que se denomina HTML dinámico (DHTML). Las técnicas que se utilizan en este tema llevan los navegadores hasta sus límites. Conforme leamos este tema vamos a ver un montón de información, pero no se sienta apabullado. Necesita saber cómo se hacen estas tareas en DHTML, de forma que podamos apreciar el trabajo que realizan las librerías que usaremos de aquí en adelante.

El proyecto: “El submarino silencioso”

El programa que vamos a crear es un puzzle de estrategia llamado “El submarino silencioso”. Esta es la premisa: el jugador es un espía que ha acabado una peligrosa misión. Se embarca en un mini submarino silencioso y tiene que maniobrar para salir del puerto. Pero el enemigo está alerta, y hay barcos patrullando por todos sitios. Si el jugador se mueve a una casilla en la que hay una patrulla enemiga, esta detectará el submarino con un sonar. Si el enemigo consigue hacer tres blancos con el sonar el juego termina. El espía tiene un dispositivo que le dice cuántos barcos patrulla hay en los alrededores. El jugador debe usarlo para encontrar un camino seguro hacia el océano. Cuando el jugador alcanza el océnao, un submarino nuclear rescata el espía y el jugador ha ganado.
Las Figuras 8.1 a 8.3 muestran la interfaz del juego. Este juego introduce nuevos desafíos. La figura del sumarino se mueve alrededor de la pantalla controlada por el programa. Además, el marcador y el mando de control son elementos especiales que se pueden mover y modificar. El juego también introduce sonidos.

Figura 8.1
Figura 8.2
Figura 8.3

Tratando la dependencia del navegador

Estas nuevas capacidades del programa del submarino tiene un coste: **los navegadores** actuales pueden hacerlo perfectamente, el problema es que **lo hacen de forma diferente**. Por ejemplo, se puede almacenar una propiedad de un fichero de sonido en Internet Explorer, pero es

necesario incrustar los ficheros de sonido cómo plug-ins en Firefox. Veremos este desastre más adelante en el tema, pero es una buena muestra de un de los mayores quebraderos de cabeza de los desarrolladores de páginas web. Los fabricantes de navegadores se han saltado los estándares en su intento de conseguir navegadores más potentes. Aunque esto significa que ambos tienen capacidades muy interesantes, es muy difícil escribir código que funcione correctamente en ambos a la vez. A veces, tendrá que escribir dos versiones diferentes del código, una para cada navegador. Una solución consiste en desarrollar sólo para uno. Esta solución podría valer para aplicaciones en una intranet o cuando está dispuesto a perder una parte significativa de clientes. Por otro lado, se supone que desarrollar para Internet supone intentar llegar al mayor número de plataformas. En ese espíritu, intentaremos que nuestros programas sean capaces de funcionar en cualquier navegador. En este tema, vamos a ver cómo lo podemos conseguir.

Creando el detective de navegadores

La clave para escribir código multinavegador es tener un programa de detección de navegador. Las Figuras 8.4 y 8.5 muestran la interfaz de un programa de este tipo. El programa es sencillo pero muy importante. Determina examinando el modelo de objetos del documento (DOM) qué navegador está funcionando y devuelve un mensaje apropiado.

Figura 8.4
Figura 8.5

Este es el código del programa “Detective de Navegadores”:

```
<html>
<head> <title>Detective de Navegadores</title>
<script type="text/javascript">
var bVersion = 0;
var esMoz = false;
var esIE = false;
function compruebaNavegador(){
// Detective de Navegadores
// Comprueba qué navegador se está usando
if (navigator.appName == "Netscape"){
esMoz = true;
alert("Navegador: " + navigator.appName);
} else {
if (navigator.appName == "Microsoft Internet Explorer"){
esIE = true;
alert("Navegador: " + navigator.appName);
} // end IE if
} // end Netscape if
bVersion = parseInt(navigator.appVersion);
if (bVersion < 4){
alert("Es aconsejable actualizar el navegador! Este programa puede no
funcionar correctamente!");
} // end if
if ((!esMoz) && (!esIE)){
alert("No reconozco el navegador. El programa puede no funcionar. " +
navigator.appName + " Version: " +navigator.appVersion);
} // end if
} // end compruebaNavegador
```

```
</script>
</head>
<body onload="compruebaNavegador()">
<center>
<h1> Detective de Navegadores<hr /></h1>
<hr />
</body>
</html>
```

Detectando qué navegador se está usando

Hay muchas maneras de detectar el navegador, pero hemos elegido trabajar con tres variables. Hemos creado la variable de tipo booleano `esMoz`. Esta variable será verdadera si el programa confirma que el navegador pertenece a la familia de navegadores de Mozilla. Otra variable `esIE`, será verdadera si se confirma el uso de Internet Explorer. Otra variable comprueba la versión del navegador.
Dado que la mayoría del HTML dinámico (incluido el que vamos a ver a partir de ahora) require versiones mayores de la 4, parece razonable comprobarlo. Los navegadores tienen un objeto `navigator`, que contiene información sobre el navegador. El objeto `navigator` tiene una propiedad `appName`, que contiene el nombre del navegador actual, y otra llamada `appVersion`, que contiene la versión. La función `compruebaNavegador()` mira esas propiedades y asigna los valores apropiados a las variables. La versión se devuelve cómo cadena de caracteres, y la función la convierte en formato numérico.

Usando Hojas de estilo (*Cascading Style Sheets*)

Muchas de las más sofisticadas capacidades de JavaScript provienen de las hojas de estilo (CSS). Esta tecnología se desarrolló para aportar flexibilidad a HTML. Lo hace pero además añade nuevas posibilidades a los programas JavaScript.

Figura 8.6

Fundamentos de CSS

El objetivo de CSS es ampliar las capacidades de HTML. La Figura 8.6 muestra un ejemplo de una página que usa una forma elemental de hoja de estilo. La idea básica de los elementos CSS es ampliar las características de las etiquetas HTML. Los elementos CSS añaden un nuevo conjunto de propiedades a los documentos.

Cómo añadir hojas de estilo a una página HTML

Hay varias formas de incorporar elementos de hojas de estilo a un documento HTML. La forma más sencilla es colocar una cadena con los comandos CSS dentro del atributo `style` de una etiqueta HTML. Hay que usar punto y coma (;) para separar los comandos. Por ejemplo, para mostrar un párrafo con letras verdes sobre fondo amarillo, podemos usar esta variante de la etiqueta `<p>`:

```
<p style="color:green; background-color:yellow">
Este texto está en verde sobre fondo amarillo.
</p>
```

Si queremos añadir estilos CSS a trozos de texto, podemos usar las etiquetas o y añadir estilo a esas etiquetas. Los elementos y se usan para agrupar y estructurar un documento. Son elementos neutros en el sentido de que no añaden nada al documento en sí. Mientras que se usa dentro de un elemento a nivel de bloque, se usa para agrupar uno o más elementos a nivel de bloque. Con CSS y se pueden usar para añadir características visuales distintivas.

Este es el HTML del programa de demostración de las hojas de estilo (CSS).

```
<html>
<head> <title>Hojas de estilo - CSS</title>
</head>
<body> <center> <h1>Hojas de estilo – CSS<hr /></h1>
<span style="height:50px; width:100px; border-style:groove; border-
color:blue; border-width:9px; background-color:yellow; color:red;
font-size:14pt">
Este es el texto afectado por CSS
</span> </center> <hr />
</body>
</html>
```

El atributo style de la etiqueta controla todas las características especiales de CSS.

Trabajando con elementos posicionables con CSS

La tecnología CSS proporciona un nivel de control sobre la presentación de HTML que agradecen los autores de páginas HTML. Hay una categoría especial de comandos CSS que son todavía más importantes para un programador de juegos, son los que permiten que los elementos se muevan de forma dinámica por la página web. La Figura 8.7 muestra un ejemplo de este fenómeno, el programa “Mueve submarino”. El HTML estándar (ain CSS) no nos permite mover imágenes (ni ninguna otra cosa, de hecho) alrededor de la página. Los navegadores principales permiten este tipo de comportamiento a los elementos que están formateados con características especiales CSS. Echemos un vistazo al HTML.

Figura 8.7

```
<body>
<center> <h1>Mover Submarino<hr></h1>
<form name="miForm" id="miForm" action="">
<input type="button" value="Mover el submarino" onclick="mueveSub()">
</form>
</center>
<span name="sub" id="sub"
style="position:absolute;left:50px;top:200px;height:30;width:30;">
<img src="./img/sub.gif">
</span> <hr />
</body>
</html>
```

Creando un elemento posicionable

La página presenta un formulario con un botón y un objeto `span` que contiene la imagen del submarino. Un objeto `span` es un contenedor especial de HTML que no tiene características propias. Esto lo convierte en candidato ideal para almacenar el estilo. Fíjese en la parte `position: absolute` de la definición de estilo. El atributo `position` especifica que el elemento puede posicionarse en la pantalla. En lugar de utilizar las normas de posicionamiento de HTML, podemos utilizar los atributos `left` y `top` de CSS para definir la posición del submarino. Podemos definir la `position` cómo `relative` o `absolute`. Posicionamiento relativo significa que las posiciones `left` y `top` se calculan a partir de la posición HTML en la que iría el elemento. Posicionamiento absoluto hace referencia a la distancia desde la esquina superior izquierda del documento, ventana o marco actual. Para la programación de juegos, utilizaremos normalmente el posicionamiento absoluto. Los atributos left, top, height, y width se utilizan para describir la posición y tamaño del objeto. Cómo la mayoría de elementos CSS, estos atributos pueden utilizar una gran variedad de unidades de medida, incluyendo pulgadas (in), centímetros (cm), milímetros (mm) y pixels (px).

CONSEJO – Para los elementos de juegos, es preferible usar pixels, porque las relaciones entre los objetos se mantienen constantes aunque la resolución de la pantalla es impredecible (No sabemos la resolución, pero un pixel es siempre un pixel.)

EN EL MUNDO REAL – Las hojas de estilo se desarrollaron cómo una forma de brindar más control a los programadores avanzados de HTML sin perder la sencillez del lenguaje. Los elementos posicionables le dan al programador un control preciso sobre el aspecto de la página web. Aún así, debe ser muy cuidadoso y probar las páginas con CSS en diferentes navegadores, ya que aquí también los estándares se siguen de forma muy relajada.

Moviendo un elemento posicionable con código CSS

Una vez hemos definido un elemento HTML con un estilo posicionable, podemos escribir código para moverlo por la pantalla. Aunque los navegadores principales soportan este comportamiento, difieren en la manera de implementarlo. El código “Detective de Navegadores” va a ser importante aquí.

```
<script type="text/javascript">
var bVersion = 0;
var esMoz = false;
var esIE = false;
function compruebaNavegador(){
// Detective de Navegadores
// Comprueba qué navegador se está usando
if (navigator.appName == "Netscape"){
   esMoz = true;
   alert("Navegador: " + navigator.appName);
} else {
   if (navigator.appName == "Microsoft Internet Explorer"){
   esIE = true;
   alert("Navegador: " + navigator.appName);
} // end IE if
} // end Netscape if
bVersion = parseInt(navigator.appVersion);
if (bVersion < 4){
```

```
 alert("Es aconsejable actualizar el navegador! Este programa puede no
funcionar correctamente!");
} // end if
if ((!esMoz) && (!esIE)){
 alert("No reconozco el navegador. El programa puede no funcionar. " +
navigator.appName + " Version: " +navigator.appVersion);
} // end if
} // end compruebaNavegador

compruebaNavegador();

function mueveSubmarino(){
// Mueve submarino
if (esMoz) {
var posx = document.getElementById("sub").style.left;
posx = posx.substring(0, posx.length - 2);
posx = 15 + eval(posx);
document.getElementById("sub").style.left = posx + "px";
if (posx > 300) {
document.getElementById("sub").style.left = "50px";
} // end if
} else {
document.all.sub.style.pixelLeft += 20;
if (document.all.sub.style.pixelLeft > 300) {
document.all.sub.style.pixelLeft = 50;
} // end if
} // end if
} // end mueveSubmarino
</script>
```

La primera parte del código es la función `compruebaNavegador()` copiada directamente del principio del tema.

CONSEJO – Copiar y pegar código no es a menudo una buena idea. Cuando se encuentre duplicando código, es señal de que se podía haber escrito el código de forma más eficiente. Aunque es muy fácil copiar y pegar código, al final se pierde bastante tiempo haciendo pequeños cambios al código y ajustando los nombres de las variables. Perder algo de tiempo en diseñar código reutilizable dará grandes beneficios más adelante. Veremos en el próximo tema cómo importar código de una librería externa. Por ahora, el copiar y pegar pequeños fragmentos de código cómo este puede ser aceptable.

La línea `compruebaNavegador()` ejecuta la función para determinar cuál es el navegador que se está utilizando.

Moviendo un elemento en Mozilla

Este es el código específico Netscape para mover el submarino:

```
if (esMoz) {
var posx = document.getElementById("sub").style.left;
posx = posx.substring(0, posx.length - 2);
posx = 15 + eval(posx);
document.getElementById("sub").style.left = posx + "px";
```

```
} // end if
```

La línea:

```
var posx = document.getElementById("sub").style.left;
```

guarda en la variable `posx` el valor del atributo `left`. Pero el valor devuelto no es un número si no una cadena de caracteres con el valor + “px”. La línea siguiente le quita esos caracteres a la variable:

```
posx = posx.substring(0, posx.length – 2);
posx = 15 + eval(posx);
```

a continuación se incrementa el valor de posx, ya convertido en número, en 15 unidades.
Por último al asignar el nuevo valor al atributo left debemos añadir la cadena “px”:

```
document.getElementById("sub").style.left = posx + "px";
```

TRUCO – Recuerde, cuando tenga código que incremente o decremente el valor de una variable, hay que comprobar siempre que nos se han sobrepasado los límites del intervalo.

Figura 8.8

El enfoque de Internet Explorer

Internet Explorer tiene una forma diferente de abordar los elementos posicionables. IE no reconoce elementos `layer`, pero puede mover directamente un elemento especificado cómo posicionable. La Figura 8.9 muestra cómo el modelo de documentos de Internet Explorer ve los elementos posicionables. IE tiene un elemento especial llamado `all`, que es un contenedor de todos los objetos del formulario. El objeto `sub` es uno de los elementos del formulario, y se podrá acceder a él a través de `all`. IE no usa objetos `layer`, pero considera que cualquier objeto puede ser potencialmente desplazable. No obstante, el elemento con el estilo CSS `position` no tiene las propiedades `left` y `top` directamente. Estas propiedades pertenecen a la propiedad `style` del objeto. Por último, las propiedades no se llaman `left` y `top`, sino `pixelLeft` y `pixelTop`. Este es el código específico para IE del programa “Mover el submarino”:

```
document.all.sub.style.pixelLeft+= 20;
if (document.all.sub.style.pixelLeft > 300){
document.all.sub.style.pixelLeft = 50;
} // end if
```

La primera línea mueve el elemento `sub` 20 pixels a la derecha. La sentencia if comprueba si se ha sobrepasado el pixel 300. Si es así, la sentencia reinicia la posición.

Figura 8.9

Si cree que es de locos que los fabricantes de navegadores no se pongan de acuerdo en la forma de mover un objeto alrededor de la pantalla, no es el único. Toda la comunidad de programadores clama por unificar los estándares.

INCISO

Creación de W3C y estandarización de la Web

El World Wide Web Consortium (W3C), se fundó en 1994 para promover estándares abiertos para la World Wide Web, Netscape, Microsoft junto con otras compañías desarrollaron un estándar para lenguajes que funcionaran en un navegador que se llamó "ECMAScript". La primera versión del estándar se publicó en 1997. Las siguientes versiones de JavaScript y JScript implementan el estándar ECMAScript para una mejor compatibilidad entre navegadores. Después de la estandarización de ECMAScript, W3C empezó con la estandarización del Modelo de objetos del documento (DOM), que es la forma de representar e interactuar con los objetos en los documentos HTML, XHTML y XML. Las versiones del DOM Nivel 0 y Nivel 1 se introdujeron en 1996 y 1997. Sólo se consiguió un soporte limitado por parte de los navegadores, de hecho navegadores no compatibles cómo Internet Explorer 4.x y Netscape 4.x se seguían usando en el año 2000. El estándar del DOM se hizo popular con el Nivel 2, que se publicó en el año 2000. Introducía la función getElementById() así cómo un modelo de eventos soportaba espacio de nombres para XML y CSS. DOM Nivel 3, la versión actual, se publicó en Abril de 2004, añadía soporte para XPath y gestión de eventos de teclado, así cómo una interfaz para serializar documentos cómo XML. Hacía 2005, la mayor parte del DOM era soportada correctamente por loa mayoría de los navegadores cómo Microsoft Internet Explorer, Opera, Safari y los navegadores basados en Gecko (cómo Firefox, SeaMonkey y Camino).

EN EL MUNDO REAL – Puede decidir escribir sus programas para un sólo navegador, pero perderá muchos clientes. Mi técnica preferida es comprobar el tipo de navegador, y poner una sentencia alternativa en cada función que use técnicas específicas de cada navegador. Esto es un calvario, pero es la única solución si queremos que nuestro código se ejecute en distintos navegadores. En el próximo tema veremos otra solución que elimina estos problemas, pero es importante ver lo que supone codificar para distintos navegadores para apreciar en lo que vale **el uso de librerías**.

Cambiando el texto de un elemento posicionable

Una vez hemos definido un elemento HTML con un atributo position en style, podemos escribir nuevo código HTML a ese elemento. Esto se hace de forma diferente (por supuesto) en Netscape y en Internet Explorer, pero el resultado es el mismo. Las Figuras 8.10 y 8.11 muestran la interfaz del programa que ilustra cómo se puede escribir nuevo HLML a un elemento ya definido.

Figura 8.10
Figura 8.11

El programa copia el contenido de un cuadro de texto en un elemento posicionable. Desgraciadamente, los dos navegadores más utilizados tienen formas muy diferentes de hacerlo. No obstante, el código HTML es el mismo, independientemente del navegador. Este es el código HTML:

```
<body>
<center> <h1>Escribiendo en un layer<hr /></h1> </center>
<span name="salida" id="salida" class="output"
```

```
style="position:absolute;left:200px;top:200px;"> Texto por
defecto</span>
<form name="miForm" id="miForm" action="">
<input type="text" name="txtEntrada"> <br />
<input type="button" value="Cambiar salida" onclick="cambiar()" />
<hr />
</form>
</body>
```

Fíjese en el objeto `span` con un atributo `position` en `style`. Este es el trozo de código que será modificado. Tiene un atributo `name` y un atributo `id`. Netscape hace generalmente referencia al objeto `span` con el atributo `name`, mientras IE prefiere el atributo `id`. Hemos usado ambos con el mismo valor. El código para cambiar el texto del objeto `span` es muy dependiente del navegador. La función empieza comprobando qué navegador se está usando, y a continuación escribe en el objeto `span` de manera apropiada al navegador. Esta es la función `cambiar()`:

```
function cambiar(){
// Escribiendo en un layer
compruebaNavegador();
var miTexto=document.miForm.txtEntrada.value;
if (esMoz){
document.salida.document.open();
document.salida.document.write(miTexto);
document.salida.document.close();
} else {
document.all.salida.innerHTML = miTexto;
} // end if
} // end cambiar
```

El código entre el if y el else sólo se ejecutará si el navegador es de Netscape. El código entre el else y el final del if se ejecutará sólo en Internet Explorer. Cómo vemos, ambos navegadores resuelven este problema de forma muy diferente.

EN EL MUNDO REAL – Podemos utilizar la facilidad de cambiar un elemento de forma selectiva sin necesidad de cambiar la salida directamente en la página y sin usar áreas de texto o marcos para la salida. Esto se puede usar en aplicaciones que queremos que algunas partes de la página cambien sobre la marcha. Podemos obtener efectos muy interesantes con elementos dinámicos, pero estas capacidades vienen a costa de la sencillez.

Figura 8.12

Escribiendo en un objeto span en Internet Explorer

El modelo de documento de Internet Explorer reconoce un objeto `span` cómo un objeto accesible mediante el grupo `all`, cómo recordaremos de lo visto anteriormente. El objeto `span` tiene una propiedad `innerHTML`. Podemos cambiar el texto de un objeto `span` asignándole nuevo código HTML. La Figura 8.12 muestra un diagrama del modelo de objetos de documento de Microsoft. La versión IE del código JavaScript es muy sencilla:

```
document.all.salida.innerHTML = miTexto;
```

Cualquier valor que queramos escribir en el elemento puede copiarse a la propiedad `innerHTML` del elemento.

Escribiendo en un objeto span en Netscape

El modelo de objetos de documento de Netscape no reconoce la propiedad innerHTML. En su lugar, un elemento posicionable es más bien cómo un marco o una ventana externa. Tiene un objeto documento propio, con las características de un objeto de documento. La Figura 8.13 muestra un diagrama del modelo de Netscape.

Figura 8.13

El modelo de Netscape trata al nuevo elemento cómo un marco. Este tiene un objeto document sobre el que puede escribir. Por supuesto, antes de escribir en un objeto document tiene que abrirlo, y una vez realizada la escritura, cerrarlo. Esta la versión de código para Netscape:

```
document.salida.document.open();
document.salida.document.write(miTexto);
document.salida.document.close();
```

CONSEJO – La sintáxis document.salida.document parece bastante extraña, pero tiene sentido si miramos el modelo de documento de Netscape. El primer document hace referencia al objeto document principal. Casi todas las referencias a objetos en JavaScript empiezan así. salida es el nombre de una propiedad del documenteo, concretamente un elemento posicionable. El nombre completo de este elemento es document.object. Cómo a su vez este elemento tiene un objeto document, podemos hacer referencia a él con la sintáxis document.object.document.

Añadir sonido multiplataforma

Los efectos de sonido son muy importantes en el desarrollo de juegos. Ayudan a ambientar el juego y pueden dar importantes indicaciones al jugador sin ocupar espacio en la pantalla. Los navegadores principales permiten incorporar sonido desde código JavaScript. No le sorprenderá que las técnicas para hacerlo sean totalmente diferentes para Netscape y para Internet Explorer. La Figura 8.14 muestra un programa que demuestra el uso de ficheros de sonido. No es difícil incorporar sonido cómo un enlace a una página, pero los navegadores utilizan técnicas muy diferentes para incrustar sonido en la página, sonido que comenzará bajo control del programa. Este es el código HTML para el programa de demostración del sonido:

```
<body>
<!-incrustar sonido en Netscpae-->
<embed name="sndPing"
      src="ping.wav" hidden="true" autostarts="false" />
<!--sonido bgsound para IE -->
<bgsound id="bgPing" />
<form> <input type="button" value="Ping!" onclick="playPing()">
</form>
</body>
```

Cuando se trata de manipular el sonido, los navegadores no se ponen de acuerdo ni en la técnica HTML. Incrustamos sonido en una página mediante la etiqueta `<embed>` en Netscape y mediante la etiqueta `<bgSound>` en IE.

La aproximación al sonido de Netscape Navigator

El punto de vista Netscape implica aplicaciones de ayuda. Si queremos generar sonido en un documento Netscape, usamos la etiqueta `<embed>` para indicar que un determinado tipo de fichero debe incrustarse en la aplicación (ver Figura 8.15). Cuando el navegador intenta presenar la página, mira la propiedad **src** de la etiqueta `<embed>` para determinar el plug-in que necesita usar. En el caso de ficheros **.wav** o **.midi**, Netscape usa el intérprete de medios por defecto. Para sonidos que se van a controlar via programa, necesitaremos poner los atributos **hidden** y **autostarts** a `false`. El atributo **mayScript** indica que la aplicación va a acceder a algún método o atributo del objeto mediante un script. El objeto interprete de medios que usa Netscape para cargar el sonido tiene un método **play()**. Cuando se invoca, este método hace que se reproduzca el sonido. Además, se puede cambiar de forma dinámica el valor de la propiedad **src** para cambiar el sonido que se reproduce.

Figura 8.14
Figura 8.15

EN EL MUNDO REAL – Muchas páginas se benefician del soporte de sonido. Los programas educativos especialmente. Recuerde que los ficheros de sonido tardan bastante en cargarse, por tanto cuidado con no cargar ficheros muy grandes.

La técnica específica de Netscape para reproducir el sonido es:

```
document.sndPing.src="ping.wav";
document.sndPing.play();
```

Aproximación de Internet Explorer al sonido

Afortunadamente, el modelo de sonido de Internet Explorer es algo más robusto. IE soporta la etiqueta `<bgSound>`. Esta etiqueta se usa a menudo, para generar sonido que suena de fondo. A muchos usuarios les molesta no poder desconectar el sonido, por tante el modelo de sonido de IE nos permite controlar la etiqueta mediante programa. La Figura 8.16 muestra cómo el modelo de documento de IE trata el sonido. El código JavaScript para reproducir sonido es relativamente sencillo:

```
document.all.bgPing.src = "ping.wav";
```

Figura 8.16

Juntando todas las partes en el juego “Submarino silencioso”

El juego “Submarino silencioso” utiliza todos los elementos que hemos visto en este Tema, más alguna idea de temas anteriores, para crear un interesante juego de acción/estrategia. El diseño del juego utiliza varios elementos posicionables. También usa sonido y una matriz. La pantalla principal se carga desde otra página cómo en el juego de “Baloncesto” del Tema 7. Esta técnica

nos garantiza que la nueva página tenga el tamaño adecuado y que no sea modificado por el usuario.

Escribiendo el código HTML

El diseño de la página principal del submarino es razonablemente sencilla: El fondo se pinta con una rejilla en él. La página contiene tres elementos posicionables. El submarino en sí es un objeto span. El marcador es también un objeto span. Este elemento no se mueve pero contiene todos los resultados del marcador. El tanteo se actualiza dentro del marcador dinámicamente y simula ser parte de la pantalla. Por último, los botones de control forman parte también de un elemento posicionable. Este enfoque es la forma más fácil de asegurarnos que los botones se posicionan bien. Este es el código HTML para el juego del “Submarino silencioso”:

```
<body onLoad = "inicializa()"
      background = "./img/oceano.gif">
<span name = "sub" id = "sub"
      style = "position:absolute; left:22px; top:125px">
<img src ="./img/sub.gif"  height="30" width="30">
</span>
<span id = "salida"    name = "salida"
      style = "position:absolute;
               left:330px; top:30px;
               color:white;
               background-color:red;
               border-style:ridge;
               height:60;
               width:60;
               font-family:'sans-serif';
               font-size:8pt">
fila: 5<br>
col: 0<br>
barcos: 0<br>
tocados: 0
</span>
<center>
<span style = "position:absolute; left:30px; top:325px">
<center>
<form name = "miForm">
<input type = "button"
       value = "/\"
       onClick = "mueveSub(NORTE)"><br>
<input type = "button"
       value = "<--"
       onClick = "mueveSub(OESTE)">
<input type = "button"
       value = "-->"
       onClick = "mueveSub(ESTE)"><br>
<input type = "button"
       value = "\/"
       onClick = "mueveSub(SUR)"><br>
</form>
</center>
</span>
<!-- Sonidos incrustados para Netscape -->
```

```
    <embed mayScript
           name = "sndSound"
           src = "ping.wav"
           hidden = "true"
           autostarts = "false" />
-->
<!-- bgsound para IE -->
<bgsound id = "soundPlayer" />
</center>
</body>
</html>
```

Cómo vemos, el código HTML define los elementos posicionables e incrusta un elemento sonido de tipo oculto para Netscape. También prepara un objeto `bgSound` para IE. Cuando se termina de cargar el `body` de la página, se llama a la función `inicializa()` para realizar las tareas de inicialización. Los cuatro botones llaman a la misma función `mueveSub()`, pero con un valor diferente que indica en qué dirección se mueve el submarino.

Creando las variables globales

Cuando se estudia un programa nuevo, una de las mejores formas de hacerse una idea general es mirar a las variables que están definidas fuera de las funciones, las llamadas variables globales. Esto nos dará una idea de los datos que usa el programa. Este es el código que genera esas variables:

```
var esMoz = false;
var esIE = false;
var NORTE = 0;
var ESTE = 1;
var SUR = 2;
var OESTE = 3;
var filas = 10;
var cols = 10;
var filaActual = 5;
var colActual = 1;
var numTocados = 0;
var rejilla = new Array();
```

El programa tiene variables para determinar qué navegador se está utilizando. También utiliza una serie de variables para simplificar el trabajo con las direcciones (NORTE, SUR, ESTE y OESTE). Puesto que los valores de estas variables no van a cambiar en todo el programa las podemos considerar cómo constantes y para recordarlo las hemos puesto en mayúsculas. Las variables `filas` y `cols` especifican las filas y columnas que va a tener la estructura de la rejilla.

CONSEJO – La rejilla que ve el usuario no es más que un gráfico de fondo, no tiene nada que ver con el programa en sí. Hemos hecho el gráfico de forma que tenga el número de filas y columnas correcto, pero el programa trabaja con los valores de las variables y el gráfico podría ser cualquier cosa. Pero sin la ayuda del gráfico de la rejilla el juego pierde bastante.

Las variables `filaActual` y `colActual` se usan para determinar la posición del submarino dentro de la rejilla. La variable `numTocados` almacena las veces que el submarino ha sido alcanzado por los barcos. La variable más importante del juego es `rejilla`. Se trata de una matriz de dos dimensiones, que almacena las posiciones de los barcos. Puesto que el jugador no ve los barcos es importante guardar la posición de los mismos. La matriz `rejilla` hace eso precisamente. Es la estructura clave del programa. Todo el juego gira alrededor de esta variable. La Figura 8.17 muestra la imagen actual de la rejilla. El tablero de juego tiene 10 x 10 celdas. El programa tiene una función que analiza una celda y cuenta los barcos que hay en las celdas vecinas. Hubiese complicado bastante la función de análisis de los bordes, para simplificarla hemos añadido un marco de ceros alrededor. Las únicas celdas en las que puede posicionarse el submarino están en el rango 1– 10; además, otro conjunto de celdas que contienen el valor 0 las rodean, así el submarino tiene siempre cuatro celdas vecinas.

Figura 8.17

Creando las funciones

Además de estudiar las variables globales, en un programa necesitamos estudiar las funciones y, valga la redundancia, qué función desempeñan. Esto nos dará una visión global del funcionamiento del programa. Una vez nos hemos hecho una idea del funcionamiento global, podemos analizar cada una de las funciones de forma más específica.

compruebaNavegador()

La función `compruebaNavegador()` establece el valor de las variables `esMoz` y `esIE` a verdadero o falso. No lo explicamos pues ya lo hemos visto con detenimiento.

inicializa()

La función `inicializa()` establece los valores de las variables principales del programa, llama a la función `compruebaNavegador()`, llama a la función `creaRejilla()` y posiciona el gráfico del submarino en la posición inicial. Se llama en el evento `onLoad` del `body` y cada vez que se reinicia el juego. Este es el código:

```
function inicializa(){
  compruebaNavegador();
  creaRejilla();
  playSonido("inmersion");
  filaActual = 5;
  colActual = 1;
  numTocados = 0;
  //actualizaMarcador();
  //mueve submarino a posición inicio
  if (esMoz){
    //document.sub.moveTo(22, 125);
  } else {
    document.all.sub.style.pixelLeft = 22;
    document.all.sub.style.pixelTop = 125;
  } // end if
} // end inicializa
```

creaRejilla()

Esta función convierte la variable `rejilla` en una matriz bidimensional que contiene los datos de los barcos. La rejilla resultante es mayor que el número de filas y columnas que utiliza el usuario, al estar estas rodeadas de un marco de celdas con el valor 0. Cada elemento de la rejilla contendrá un 0 (no hay barco) o un 1 (hay un barco en esa posición de la rejilla). Cada celda tiene un 30 % de probabilidades de contener un barco. Los barcos se sitúan de forma aleatoria. La forma más fácil de cambiar la dificultad del juego es cambiar la probabilidad de que una celda contenga un barco. Un valor menor de .30 hace el juego más fácil, un valor mayor de .30 incrementa la dificultad del juego. Este es el código de la función:

```
function creaRejilla(){
  var fila = 0;
  var col = 0;
  //limpia los bordes
  for (fila = 0; fila <= filas+1; fila++){
    rejilla[fila] = new Array();
    rejilla[fila][0] = 0;
    rejilla[fila][cols+1] = 0;
  } // end fila
  for (col = 0; col <= cols+1; col++){
    rejilla[0][col] = 0;
    rejilla[filas + 1][col] = 0;
  } // end col

  //rellenar la rejilla con barcos de forma aleatoria
  for (fila = 1; fila <= filas; fila++){
    for (col = 1; col <= cols; col++){
      if (Math.random() < .30){
        rejilla[fila][col] = 1;
      } else {
        rejilla[fila][col] = 0;
      } // end if
    } // end col for
  } // end fila for
} // end creaRejilla
```

obtieneRejilla()

La función `obtieneRejilla()` se ha usado para depurar el programa y su misión era imprimir la posición de los barcos en un cuadro de diálogo. En el programa final no se utiliza.

contarBarcos()

La función contarBarcos() localiza la celda determinado por filaActual y colActual y calcula cuántos barcos hay en las celdas vecinas. La función suma el valor almacenado en las celdas vecinas. Este es el código de la función:

```
function contarBarcos(){
  //dada la fila y col actuales, contar cuántos barcos hay en los
  //alrededores
  var numBarcos = 0;
  numBarcos += rejilla[filaActual - 1][colActual];
```

```
  numBarcos += rejilla[filaActual][colActual - 1];
  numBarcos += rejilla[filaActual][colActual + 1];
  numBarcos += rejilla[filaActual + 1][colActual];
  return numBarcos;
} //end contarBarcos
```

actualizaMarcador()

La función actualizaMarcador() hace exactamente lo que su nombre indica. Examina el valor de determinadas variables y los concatena en una cadena HTML, que se escribe en el elemento posicionable.

```
function actualizaMarcador(){
  var marcador = "";
  marcador += "<font color = white>";
  marcador += " fila: " + filaActual + "<br>";
  marcador += " col: " + colActual + "<br>";
  marcador += " barcos: " + contarBarcos() + "<br>";
  marcador += " tocados: " + numTocados ;
  marcador += "</font>";

  if (esMoz){
    document.getElementById("salida").innerHTML=marcador;
  } else {
    document.all.salida.innerHTML = marcador;
  } // end if
} // end actualizaMarcador
```

Destaquemos que el programa llama a la función `contarBarcos()` que devuelve un número cuyo valor se concatena para mostrarlo al usuario.

playSonido()

La función `playSonido()` recibe el nombre de un sonido, localiza el fichero que contiene ese sonido y lo reproduce. Este es el código:

```
function playSonido(nomSonido){
   if (esMoz){
       document.sndSound.src = "./sonido/"+ nomSonido + ".wav";
       document.sndSound.play();
   } else {
     var fichSonido = "./sonido/"+ nomSonido + ".wav";
     document.all.soundPlayer.src = fichSonido;
  } // end if
} // end playSonido
```

mueveSub()

La función mueveSub() espera recibir una variable que indique la dirección del movimiento (NORTE, SUR, ESTE, OESTE). La función se llama cada vez que se pulsa uno de los botones de dirección. Cuando el usuario hace click sobre un botón, este llama a la función pasándole un parámetro que indica la dirección a la que se tiene que mover. La función analiza el parámetro, y mueve la imagen del submarino de forma adecuada, también comprueba si el submarino ha

sido alcanzado. Por último comprueba si el jugador ha ganado, es decir ha conseguido alcanzar la columna 10. O si por el contrario ha sido alcanzado por tercera vez en cuyo caso se entiende que ha sido hundido y se pierde el juego. Asimiso se llama a la función actualizaMarcador(), para actualizar la información del juego.

```
function mueveSub(direccion){
  var dx = 0;
  var dy = 0;
  switch (direccion){
    case 0:
      dx = 0;
      dy = -30;
      filaActual--;
      break;
    case 1:
      dx = 30;
      dy = 0;
      colActual++;
      break;
    case 2:
      dx = 0;
      dy = 30;
      filaActual++;
      break;
    case 3 :
      dx = -30;
      dy = 0;
      colActual--;
      break;
  } // end switch

  if (esMoz){
      var posx = document.getElementById("sub").style.left;
      posx = posx.substring(0, posx.length - 2);
      posx = dx + eval(posx);
      document.getElementById("sub").style.left=posx + "px";
      var posy = document.getElementById("sub").style.top;
      posy = posy.substring(0, posy.length - 2);
      posy = dy + eval(posy);
      document.getElementById("sub").style.top =posy + "px";
  } else {
    document.all.sub.style.pixelLeft += dx;
    document.all.sub.style.pixelTop += dy;
  } // end if

  //comprueba si ha ganado jugador
  if (colActual > 10){
    playSonido("inmersion");
    alert("Enhorabuena!! has ganado!!!");
    inicializa();
  } // end if

  //comprueba tocados
  if (rejilla[filaActual][colActual] == 1){
    playSonido("ping");
```

```
    numTocados++;
    if (numTocados >=3){
      playSonido("torpedo");
      alert("Lo siento! Has sido hundido!!");
      inicializa();
    } // end hundido if
  } // end  tocado if
  actualizaMarcador();

} // end mueveSub
```

Resumen

En este tema hemos profundizado en las dificultades del desarrollo multinavegador en HTML. Hemos visto cómo determinar qué navegador se está utilizando. Hemos experimentado con las hojas de estilo (CSS) cómo una forma de añadir diseño al código HTML. Además, hemos visto cómo usar CSS para posicionar los elementos HTML de forma más precisa. Hemos visto programas que mueven elementos, cambian el texto y reproducen sonido.

EJERCICIOS

8.1. Modificar el programa del submarino, de forma que el sensor devuelva 1 punto si el barco está en el NORTE, 2 si está en el OESTE, 4 si está en el SUR y 8 si está en ESTE. Si el usuario es capaz de averiguar esta regla, sabrá siempre dónde se encuentran los barcos y podrá escapar fácilmente.

8.2. Crear un puzzle de nueve piezas, en una rejilla de 3 x 3, a partir de una imagen. El juego empieza mostrando la imagen completa, cuando pulsamos un botón mezclar, quitamos aleatoriamente una de las piezas y colocamos las demás de forma aleatoria en la rejilla. El jugador tiene que recomponer la imagen original, cuando lo consiga se mostrará esta completa. El usuario mueve un trozo del puzzle haciendo click en botones similares a los del juego del submarino. La pieza vecina del hueco realiza el movimiento indicado por el botón y ocupa el hueco. El hueco queda dónde estaba el trozo de imagen. Si el movimiento no es correcto, no se hace nada.