JavaScript de Andy Harris

Tema 6. Salida dinámica

En este tema, vamos a utilizar el código HTML para generar salida. Hasta ahora hemos usado salida mediante cuadros de diálogo, campos de texto o áreas de texto. Aunque este uso es sencillo, también es muy limitado. Concretamente, vamos a ver:

• Incorporar JavaScript y marcos HTML.
• Escribir código que escribe en otro marco.
• Generar ventanas externas.
• Controlar la apariencia y comportamiento de las ventanas externas.
• Escribir en ventanas externas.
• Generar imágenes para juegos.
• Incorporar gráficos en un juego sencillo.

Proyecto: “Pétalos alrededor de la rosa”

El juego que vamos a programar en este tema está basado en un viejo juego de dados. Se lanzan cinco dados y se pregunta “Cuántos pétalos alrededor de la rosa?” Hay un patrón para obtener la respuesta correcta. El juego es relativamente sencillo, pero puede ser más complicado averiguar la solución. Las figuras 6.1 a 6.4 muestran la interfaz del juego.

Figura 6.1
Figura 6.2
Figura 6.3
Figura 6.4

Este programa introduce varios elementos nuevos. Es el primer programa que vamos a hacer que incorpora gráficos. Además genera la salida directamente sobre la página web, sin usar áreas de texto o cuadros de diálogo. También incorpora una ventana emergente de ayuda, que se implementa cómo una segunda página web. Para construir el juego, vamos a ver cada uno de los nuevos elementos. El juego en sí es relativamente sencillo de construir una vez que conocemos todos los elementos.

Generando la salida en marcos

El primer elemento que tenemos que dominar es cómo generar código HTML sobre la marcha. Esto es muy importante porque HTML es un lenguaje con gran expresividad. Si quiere incorporar gráficos en sus programas, o quiere generar páginas web sobre la marcha necesitamos el HTML. Cómo hemos visto en el tema 4, podemos utilizar el método `document.write()` para añadir HTML a una página. No obstante, este método tiene un fallo importante: no nos permite escribir en un documento una vez que el navegador ha terminado de cargar el documento.

EN EL MUNDO REAL – La mayoría de aplicaciones web incluyen algún tipo de ineracción dinámica con la pantalla. Los cuadros de diálogo son estéticamente poco adecuados y además interrumpen el flujo del programa. La capacidad de generar salida de forma dinámica es crítica en cualquier aplicación web profesional. Las técnicas que nos van a permitir generar HTML en una función JavaScript y mostrarlo en un marco o una ventana diferente son muy útiles en cualquier aplicación web.

Creando el progrma “Prueba de marcos HTML”

Parece imposible escribir funciones que generen páginas web, pero hay formas de hacerlo. Echemos un vistazo al programa “Prueba de marcos HTML” de las Figuras 6.5 y 6.6. El programa “Prueba de marcos HTML” es una herramienta para enseñar y practicar HTML. Cuando estamos aprendiendo HTML, podemos teclear cualquier código en el área de texto, hacer click en el botón y ver los resultados. No necesita ni un editor de textos. El truco es que la entrada y la salida ocurren en dos páginas diferentes. El formulario con el área de texto, el botón y la función están en el lado izquierdo de un marco,

Figura 6.5
Figura 6.6

y el lado derecho se construye con la función. Crear este programa no es tan complicado cómo puede parecer

Creando los marcos

Quizás sea conveniente repasar brevemente los marcos en HTML. Recordemos que un grupo de marcos implica varios documentos. Este es el código de la página principal del programa “Prueba de marcos HTML”.

```
<html>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<head><title>Prueba de HTML – Version con marcos</title>
<frameset border="0" cols="50%,50%">
<frame src="FrameTester.html">
<frame src="blank.html" name="frameSalida">
</frameset>
</html>
```

Esta página no tiene elemento `body`, sólo `head`. La página contiene un elemento conjunto de marcos (*frameset*) con dos marcos. El primer marco contiene la página `"FrameTester.html"`. Y el segundo `blank.html`. Fíjese que hemos ocultado el recuadro del marco (`border = 0`). Esto es muy común cuando trabajamos con JavaScript si queremos que el resultado parezca una sola página. Hemos llamado al segundo marco `"frameSalida"`. Es muy importante darle un nombre a los marcos que van a albergar la salida. La página `"blank.html"` es una página vacía que utilizamos para inicializar el *frameset*.

Escribiendo en un segundo marco

El código HTML de `"FrameTester.html"` es muy estándar. Cómo de costumbre, el código JavaScript lo incluimos después.

```
<body>
<center>
<h1>Prueba de marcos<hr /></h1>
    <form name="miForm" id="miForm" action="">
        <textArea name="txtEntrada" rows="10" cols="30">
            <html>
            <head><title>Practique</title>
            <body bgColor="lightblue">
            <h3>Página de prácticas</h3>
            <img src="./img/chico.gif" />
            </body>
            </html>
        </textarea>
        <br />
        <input type="button" value="Mostrar página"
onclick="crearPagina()" />
    </form>
<hr />
</center>
</body>
```

El código HTML solamente prepara un área de texto y un botón. Fijarse que el navegador no interpreta directamente el código entre las etiquetas `<textarea>` y `</textarea>` cómo HTML, si no cómo el texto que aparecerá en el área de texto. El botón ejecuta la función `crearPagina()`. Este es el código:

```
<script type="text/javascript">
function crearPagina(){
// Prueba de marcos
// Muestra cómo escribir HTML en otro marco
var elCodigo = document.miForm.txtEntrada.value;
var elDocumento = window.parent.frameSalida.document;
elDocumento.open();
elDocumento.write(elCodigo);
elDocumento.close();
} // end function crearPagina
</script>
```

La primera línea obtiene el texto almacenado en el área de texto y lo copia a la variable `elCodigo`. La segunda línea es un poco más especial:

```
var elDocumento = window.parent.frameSalida.document;
```

Esta línea también crea una variable, pero esta variable es una referencia al documento del marco `frameSalida`. Cuando una página usa varios marcos, es necesario hacer referencia al conjunto de marcos (*frameset*). En este caso, lo hemos hecho añadiendo una referencia a `window.parent`. `frameSalida` es el nombre del marco específico, y `window.parent.frameSalida.document` es una referencia al marco dentro del conjunto de marcos del navegador.

La sintaxis `window.parent.frameSalida.document` es incómoda y es difícil trabajar con ella. Por ello guardamos una referencia en la variable `elDocumento` y a partir de ahora

trabajamos con ella, podemos invocar sus métodos y manipularla. La llamada `elDocumento.open()` borra cualquier contenido anterior y prepara el documento para nuevo texto. `elDocumento.write(elCodigo)` escribe el contenido de `elCodigo` en la página. Por último, `elDocumento.close()` le dice al navegador que la página está terminada y que puedea mostrar el docuemento.

CUIDADO – No puede presuponer que una llamada a document.write() será visible al usuario hasta que no se haya cerrado el documento. No olvide incluir la llamada a documento.close(). Si no se incluye no hay garantía de que todos los elementos del documento se muestren.

Lo que necesitamos para generar salida en otro marco es algo de planificación. Incializar el frameset, asegurarse de dar nombre a cualquier marco que vaya a contener salida. Construir una función que ensamble la página web en una variable de texto. Crear una referencia al marco, abrir el marco, escribir en él y por último cerrarlo.

TRUCO – Si solamente quiere cargar una página en un marco, modificar la propiedad location.href. Por ejemplo, la siguiente línea lanza mi página principal en el marco frameSalida:
elDocumento.location.href = "http://www.cs.iupui.edu/~aharris"
Esta técnica puede ser útil cuando la página que necesita ya está completa y disponible en la Web.

Mostrar salida en ventanas diferentes

Se puede crear fácilmente una nueva ventana para mostrar una página existente. Eso es lo que hemos hecho para la página de Ayuda en el juego “Pétalos alrededor de la rosa”. La pantalla de Ayuda es simplemente una página HTML. Las figuras 6.7 y 6.8 muestran el programa “Muestra la Ayuda”, que muestra esta página en una nueva ventana. La posibilidad de hacer emerger una página de esta manera puede ser útil, cuando queremos mostrar información sin modificar la disposición de la página actual.

CONSEJO – El código HTML para el programa “Muestra la Ayuda” es muy predecible (un botón que llama a la función muestraAyuda()).

Este es el código para la función `muestraAyuda()`:

```
<script type="text/javascript">
function muestraAyuda(){
// Muestra la Ayuda
// Demuestra cómo cargar una página HTML en una nueva ventana
var winAyuda = window.open("petalosAyuda.html", "pAyuda",
"width=450,height=450,resizable=yes");
winAyuda.focus();
} // end muestraAyuda
</script>
```

El código es muy corto pero muy potente. La primera línea genera una variable llamada `winAyuda`. A continuación se invoca al método `window.open()`, que simplemente abre una nueva ventana. Este método tiene tres parámetros. El primero es el nombre del fichero a cargar. El segundo es el nombre HTML del documento. Usaremos el nombre HTML si tenemos enlaces HTML al documento en la nueva ventana. (Normalmente, no los vamos a terner, pero de todas

formas es necesario especificarlo). El último parámetro es una lista de valores de atributos de la nueva ventana. La Tabla 6.1 muestra las características más importantes que podemos utilizar.
Tabla 6.1 – Atributos comunes del método window.open()

Atributos	**Descripción**	**Ejemplo**
height	Altura de la ventana	height=200
width	Anchura de la ventana	width=200
location	Si se muestra la barra location.	location=yes|no
menubar	Si se muestra la barra menubar.	menubar=yes|no
resizable	Si el usuario puede redimensionar la ventana.	resizable=yes|no
scrollbars	Si las barras de scroll están presentes.	scrollbars =yes|no
status	Si se muestra la barra de status.	status=yes|no
toolbar	Si se muestra la barra toolbar.	toolbar=yes|no
directories	Si se muestra la barra directory.	directories=yes|no

CONSEJO – Esta tabla incluye sólo aquellos atributos que son comunes a la mayoría de navegadores. Algunos otros atributos sólo funcionan en un navegador pero no en otros.

Figura 6.7
Figura 6.8

Para determinar las características de su nueva ventana, hay que crear una cadena con la lista de características que queremos incluir, separadas por comas. Si queremos una ventana de 300x300 con barra de herramientas:

```
height=300,width=300,toolbar=yes
```

Fíjese que no hay espacios entre los valores. Y que sólo se incluyen los elementos listados. Si no se especifica nada, la nueva ventana tendrá exactamente las mismas características que la actual. Hay otra línea más en la función:

```
winAyuda.focus();
```

Esta línea le dice a la nueva ventana que tome el foco igual que si el usuario hubiese hecho click en ella. O sea, cuando se crea la ventana, automáticamente obtiene el foco; pero, el usuairo puede volver a colocar el foco sobre la pantalla principal. Si el usuario vuelve a pulsar el botón de Ayuda, la pantalla se actualizará pero cómo está en segundo plano no se verá. El comando `focus()` no asegura que la pantalla de Ayuda quedará en primer plano.

EN EL MUNDO REAL – Podría pensar que una ventana sin barras no es útil, pero sin embargo sí se utiliza. Cuando no se incluyen menus ni barras, el usuario no sabe que es una nueva ventana del navegador, tiene la impresión que el programa ha lanzado una ventana emergente, que es el efecto que se pretendía conseguir.

Construyendo el programa “Prueba de ventanas”

Podemos combinar las capacidades de escribir páginas del programa “Prueba de marcos HTML” con la capacidad de abrir nuevas ventanas. Las figuras 6.9 y 6.10 muestran otra versión del programa “Prueba de marcos HTML”. Esta versión crea una nueva ventana en la que el navegador muestra la salida.

Figura 6.9
Figura 6.10

Este programa es una combinación de “Prueba de marcos HTML” y de “Muestra la Ayuda”. Sólo necesita un fichero. Este el el código HTML:

```
<body>
<center>
<h1>Prueba de ventanas<hr /></h1>
<form name="miForm" id="miForm" action="">
<textArea name="txtEntrada" id="txtEntrada" rows="10" cols="30">
<html>
<head>
<title>practicar</title>
</head>
<body bgColor="lightblue">
<h3>Página de prácticas</h3>
<img src="guy.gif" />
</body>
</html>
</textarea>
<br>
<input type="button" value="Mostrar página" onclick="crearPagina()" />
<input type="button" value="Cerrar ventana"
onclick="cerrarPagina()" />
</form>
</center>
<hr />
</body>
```

Cómo podemos ver, este código HTML es muy similar al de “Prueba de marcos”. Hemos añadido un botón “Cerrar ventana”, por lo demás es prácticamente igual. El código JavaScript es también similar pero incluye nuevas instrucciones para crear una ventana.

```
var laVentana;

function crearPagina(){
// Prueba de ventanas
// Demuestra cómo escribir HTML en una nueva ventana
laVentana = window.open("", "practicar", "height=300,width=300");
var elCodigo = document.miForm.txtEntrada.value;
var elDocumento = laVentana.document;
elDocumento.open();
elDocumento.write(elCodigo);
elDocumento.close();
laVentana.focus();
} // end crearPagina
```

```
function cerrarPagina(){
laVentana.close();
} // end cerrarPagina
```

Este código tiene algunas novedades que no hemos visto hasta ahora. Por primera vez, estamos definiendo más de una función en una página. De hecho se pueden definir tantas funciones cómo necesitemos. En este caso, hemos hecho una para cada botón. Esto es una práctica habitual. Fíjese también que hemos definido una de las variables fuera de las funciones. Esto lo hemos hecho porqué ambas funciones necesitan acceder a la variable nueva ventana. Hablaremos más en profundidad de esto. Por ahora, compruebe que se entiende todo el código.

La función `crearPagina()` empieza abriendo un objeto ventana:

```
laVentana = window.open("", "practicar", "height=300,width=300");
```

La siguiente abre el documento para poder escribir:

```
elDocumento.open();
```

Los objetos window y document tienen ambos un método open(), pero el significado del término es diferente. Después de abrir el documento, escribimos código HTML en él, lo cerramos y ponemos el foco en la nueva ventana. Fíjese que hemos cerrado el documento para decirle al navegador que hemos terminado y que puede empezar a mostrar el contenido.

EN EL MUNDO REAL – Puede parecer desconcertante que window.open() sea diferente de document.open(). Es perfectamente válido para objetos diferentes tener métodos con el mismo nombre que no hacen lo mismo. Este es un concepto que se conoce en programación orientada a objeto cómo polimorfismo. El polimorfismo puede definirse de diferentes maneras, pero básicamente, significa que los objetos pueden personalizar la forma en que realizan determinadas funciones.

Cerrando la ventana

El usuario lanza la función `cerrarPagina()` haciendo click en el botón “Cerrar ventana”. La función llama al método `close()` de la ventana creada por `crearPagina()`. El método `close()` de una ventana la hace desaparecer.

Diseñando el juego “Pétalos alrededor de la rosa”

El juego “Pétalos alrededor de la rosa” que hemos visto al principio del tema, utiliza las técnicas que acabamos de ver para realizar una interfaz mucho más elaborada que la del juego de dados del tema 2. Además es el primer programa que vamos a ver que utiliza gráficos.

CUIDADO – Si no conoce el juego “Pétalos alrededor de la rosa” es conveniente echar un vistazo al siguiente enlace para conocer su funcionamiento. Recuerde siempre que NO SE PUEDE PROGRAMAR LO QUE NO SE CONOCE.

Entendiendo la estrategia general de diseño

Esta es la información clave para el juego “Pétalos alrededor de la rosa”: El punto central del dado es la rosa, y los puntos alrededor del central son los pétalos. Luego sólo nos valen los dados con valor 1, 3, y 5. El valor 1 no tiene pétalos, sólo rosa. El 3 tiene dos pétalos y el 5 cuatro pétalos. El programa determina el número de pétalos sumando los valores que acabamos de ver del conjunto de dados de la tirada. El juego está colocado en un *frameset*, en una página llamada `petalMas.html`. La página principal se divide en dos secciones. La superior contiene la interfaz de usuario, con una caja de texto y un par de botones. Esta página contiene todo el código JavaScript. La parte inferior muestra cinco dados y la pantalla ganadora. Habitualmente, el código del marco superior genera el código HTML del marco inferior. El marco inferior no contiene ningún código JavaScript. El programa tiene además una pantalla de Ayuda, que se implementa en una nueva ventana. Este es el código HTML para el *frameset* (`petalMas.html`):

```
<html>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<head>
<frameset rows="30%, 70%" border ="0">
<frame src="petalos.html">
<frame src="empieza.html" name="salida">
</frameset>
</head>
</html>
```

El marco superior contiene `petalos.html`, que contiene todo el código JavaScript. El marco inferior se llama `salida`. Contiene código HTML muy sencillo, con etiquetas `<img>` para los cinco dados. El código HTML del marco superior genera la interfaz de usuario. Este es el código:

```
<body bgcolor="tan">
<center> <h1>Pétalos alrededor de la rosa<hr /></h1>
<form name="miForm" id="miForm" action="">
Cuántos pétalos hay alrededor de la rosa?
<input type="text" name="txtIntento" />
<input type="button" value="OK" onclick="tirarDados()" />
<input type="button" value="Ayuda" onclick="pantallaAyuda()" />
</form>
</center>
</body>
```

Cómo vemos el formulario es muy sencillo, con un cuadro de texto y dos botones. Cada botón llama a una función diferente.

Crear gráficos para juegos JavaScript

La aportación más novedosa del juego “Pétalos alrededor de la rosa” son los gráficos. Las imágenes mejoran ostensiblemente el aspecto de un programa. Con unos pequeños trucos y consejos, podremos añadir buenos gráficos a nuestros programas sin ser un artista gráfico. Podemos añadir imágenes de un par de maneras diferentes. Primero, podemos buscar en Internet alguna imagen que nos sirva. La cantidad de imágenes disponibles en Internet es abrumadora.

Cuidado que muchas imágenes en Internet tienen derechos de autor. Si va a utilizarlas de forma profesional es conveniente asegurarse que son de difusión libre o pedir permiso para utilizarlas.

EN EL MUNDO REAL – Aunque la manipulación de imágenes es especialmente importante en la programación de juegos, la incorporación de gráficos es esencial en cualquier página web profesional también.

Se pueden contratar los servicios de un artista profesional. Para juego comerciales esto es obligatorio. El problema es que los buenos artistas son caros. Por último, puede generar los gráficos usted mismo. Aunque no tenga cualidades especiales para el diseño, los modernos programas de diseño gráfico nos permiten crear unos gráficos más que aceptables para nuestras necesidades.

Creando los dados

La Figura 6.11 muestra las herramientas que se han utilizado para crear los dados usando el programa GIMP cómo editor de imágenes. Empezamos haciendo un rectángulo. A continuación insertamos una textura para el fondo. Por último usando un filtro *bump map* le damos un efecto tridimensional.

CONSEJO – Un filtro bump map es una imagen con escala de grises. Cuando se aplica a otra imagen, las áreas oscuras se interpretan cómo cóncavas, y las claras se dejan sin tocar. El editor aplica a continuación un efecto de sombreado. El resultado es una textura con un aspecto muy conseguido. He utilizado dos bump maps para las imágenes de los dados, usando uno para redondear los bordes y otros para crear las depresiones de los puntos.

Por supuesto, también se pueden pintar los puntos si se prefiere. Por último he salvado cada imagen en un fichero dado1 a dado6. Hay que guardar las imágenes en formato gif o jpeg. En general, las fotografías trabajan mejor con jpeg, y para las imágenes creadas desde cero (sobre todo si requieren transparencias) es mejor usar gif.

Figura 6.11

Creando los dibujos de las caricaturas

Aunque los dados son muy elegantes, lo que de verdad le da a este juego su personalidad son las caricaturas que se muestran en la pantalla de Ayuda y cuando el jugador gana. Estos dibujos son muy interesantes para los juegos por varias razones. En primer lugar, son razonablemente fáciles de dibujar. Son agradecidos, al ser caricaturas nadie espera que estén perfectamente dibujados. La imagen base es la de la Figura 6.12. El personaje de la viñeta es una caricatura muy sencilla de realizar. Se puede modificar fácilmente para incluir: pelo corto, sonrisa a lo goofy, cejas pobladas. He dibujado la imagen en negro sobre fondo transparente. Esto lo hace más fácil al no haber sombras implicadas. Además, el fondo transparente permite reutilizar la imagen en otros programas. Por último, una imagen con fondo transparente se integra mejor en una página web. Si sabe utilizar herramientas de niveles en su editor gráfico, puede poner cada elemento (ojo, boca, nariz, etc.) en un nivel diferente de forma que sean más fácil modificarlos sin tocar el resto. Esto permite, por ejemplo, cambiar a una cara triste cuando el jugador pierde. Nos hemos concentrado en la cabeza pero hemos dejado espacio de forma deliberada en un lado de la imagen para incluir una mano. A continuación, he salvado la imagen en formato gif, porqué soporta fondos transparentes. Para el programa “Pétalos alrededor de la rosa”, hemos decidido no usar la imagen original, si no que hemos creado dos variantes. La primera un tipo con una rosa, ver Figura 6.13. Se usa esta imagen en la pantalla de Ayuda.

Figura 6.12
Figura 6.13
Figura 6.14

Para la pantalla ganadora, nos hemos decidido por una imagen con la rosa en los labios del personaje, ver Figura 6.14. Cómo vemos, con un poco de imaginación y las herramientas gráficas adecuadas podemos conseguir imágenes muy aceptables para nuestros programas.

Generando la rutina de lanzamiento de dados

Las funciones en "`petalos.html`" hacen todo el trabajao. Una de las funciones se llama tirarDados(). Se ejecuta cada vez que el usuario hace click sobre el botón OK . El pseudocódigo para la función tirarDados() es el siguiente:

obtener el intento del usuario del área de texto
Si la respuesta es correcta,
    Mandar la pantalla "Ganador" a la variable de salida
Si la respuesta no es correcta
    Mostrar la respuesta correcta en el cuadro de diálogo
Borrar el marco de salida
Hacer esto cinco veces:
    Obtener un valor aleatorio entre 1 y 6
    Añadir el gráfico adecuado a la variable de salida
    Fin del bucle
Fin del Si
Escribir la variable de salida en el marco inferior

TRUCO – Este guión se llama pseudocódigo. El mayor error que comete un programador (sobre todo al principio) es escribir código sin un guión. Ahora que los programas empiezan a hacerse más largos, será cada vez más difícil hacer un seguimiento de por dónde vamos. Una vez tenemos el guión es razonablemente sencillo convertirlo en código JavaScript. No podremos insistir bastante en lo adecuado de hacer esto antes de lanzarnos a programar.

Comprobando la entrada del usuario

La primera tarea de la función `tirarDados()` es obtener el intento del usuario. Lo hacemos con el cuadro de texto, este es el código para la primera parte de la función:

```
var numPetalos = 12;
//almacena la respuesta correcta
function tirarDados(){
// Petalos alrededor de la rosa
// Adaptación de un viejo juego de dados
// require un documento llamado "salida" esté disponible
var elOrigen = "";
var lanzamiento = 0;
var intento = 999; //almacena el intento del usuario
//comprueba el intento del usuario
intento = eval(document.miForm.txtIntento.value);
```

El programa declara una variable fuera de cualquier función. La variable `numPetalos` contiene el valor actual correcto del número de pétalos. El valor inicial se pone en 12, porque en la página que se carga por defecto en el marco inferior el valor correcto es 12. Cada vez que se genera un nuevo conjunto de dados, el programa cambia el valor de `numPetalos` adecuadamente. La variable `numPetalos` se declara fuera de la función porque el programa necesita preservar su valor incluso cuando la función no se está ejecutando. Recuerde que las variables que se declaran dentro de una función no conservan su valor una vez se ha terminado de ejecutarse la función. La variable `numPetalos` se define en la ejecución de la función, pero su valor se necesita cuando el usuario haga click en el botón OK, por eso es necesario declarla fuera del cuerpo de la función. La función empieza con los comentarios habituales y genera una serie de variables. `elOrigen` es una variable de texto que contendrá el código del marco de salida. Parece más fácil generar el HTML de la página en una cadena de caracteres larga, y después escribir esa cadena en la página con un solo comando. La variable `lanzamiento` contiene el valor del dado generado aleatoriamente. La variable `intento` se utiliza para almacenar el intento del usuario. Se ha inicializado en 999, que es un valor sin duda incorrecto. Lo hemos hecho para asegurarnos que el programa no empieza con la pantalla de acierto sin dar una oportunidad al usuario de intentar una respuesta. La línea siguiente obtiene el intento del usuario del cuadro de texto, lo convierte en variable numérica y lo asigna a la variable `intento`:

```
intento = eval(document.miForm.txtIntento.value);
```

Gestionando la posición ganadora

A continuación el programa tiene la siguiente estructura:

```
if (intento == numPetalos){
elOrigen += "<html>";
elOrigen += "<body bgColor = tan>";
elOrigen += "<center>";
elOrigen += "<h1>Has ganado!!!</h1>";
elOrigen += "<img src='./img/payaso.jpg'>";
document.miForm.txtIntento.value = "";
} else {
//el jugador no ha acertado
} // end if
```

La sentencia if comprueba si el jugador ha acertado. Si el `intento` del jugador es igual que el valor almacenado en `numPetalos`, la función crea el código HTML de la página ganadora y lo almacena en la variable `elOrigen`. A continuación esta línea limpia el cuadro de texto:

```
document.miForm.txtIntento.value = "";
```

Generando la página de los dados

Si el usuario no ha acertado el programa genera un nuevo puzzle. He aquí el código:

```
if (intento == numPetalos){
// el usuario ha acertado
} else {
```

```
alert("Hay " + numPetalos + " pétalos alrededor de la rosa.");
numPetalos = 0;
document.miForm.txtIntento.value = "";
elOrigen += "<html>";
elOrigen += "<body bgColor='black'>";
elOrigen += "<center>";
elOrigen += "<br><br><br><br>";
for (i = 1; i <= 5; i++){
tirada = Math.floor(Math.random() * 6) + 1;
switch (tirada){
case 1:
elOrigen += "<img src='./img/dado1.gif'> ";
break;
case 2:
elOrigen += "<img src='./img/dado2.gif'> ";
break;
case 3:
elOrigen += "<img src='./img/dado3.gif'> ";
numPetalos += 2;
break;
case 4:
elOrigen += "<img src='./img/dado4.gif'> ";
break;
case 5:
elOrigen += "<img src='./img/dado5.gif'> ";
numPetalos += 4;
break;
case 6:
elOrigen += "<img src='./img/dado6.gif'> ";
break;
default :
elOrigen += "ERROR!!";
} // end switch
} // end loop
} // end if

elOrigen += "</center>";
elOrigen += "</body>";
elOrigen += "</html>";
window.parent.salida.document.open();
window.parent.salida.document.write(elOrigen);
window.parent.salida.document.close();
} // end function
```

Si el usuario no ha acertado, lo primero es decirle al usuario la respuesta correcta. Hemos usado un cuadro de diálogo para lograr que se pare la ejecución hasta que el usuario haga click en el botón. Además, permite que el usuario vea el último puzzle y la solución del mismo. En cuanto el usuario haga click en el botón OK del cuadro de diálogo, el programa coloca un nuevo conjunto de dados en el cuadro de salida. Ponemos la variable `numPetalos` a 0, ya que el programa calcula el valor correcto al generar los dados. Hacemos un bucle for para generar los cinco dados. El procedimiento para generarlos es idéntico. Usamos el algoritmo que vimos en el tema 2 para generar un número aleatorio del 1 al 6, a continuación una sentencia switch para añadir la imagen del dado adecuada a la página. Echemos un vistazo especial al case 3 y al case 5. Estos son los únicos dados que tienen pétalos alrededor de la rosa. Si estamos en el case 3

sumamos dos pétalos y si estamos en el case 5 sumamos cuatro pétalos. Colocamos además un case default en la estructura switch por si se produce algún error.

Generando la pantalla Ayuda

La pantalla de Ayuda es una simple página web que ya está generada. Para hacerla aparecer, usamos el mismo código que en programas vistos anteriormente. Esta es la versión del programa “Pétalos alrededor de la rosa”:

```
function pantallaAyuda(){
//muestra una pantalla de ayuda
//requiere petalosAyuda.html
var ventanaAyuda = window.open("petalosAyuda.html", "pHelp",
"height=550,width=550");
ventanaAyuda.focus();
} // end pantallaAyuda
```

La función genera una nueva ventana y establece el foco en ella. No se necesita nada más.

Resumen

En este Tema, hemos visto cómo generar salida hacia una página diferente a la que contiene la función. Concretamente, hemos aprendido a escribir en otro marco y en una nueva ventana que hemos creado mediante código. Las ventajas de la salida externa es que podemos construir la página web completa como salida de nuestro programa, lo que nos permite incorporar las capacidades de HTML en nuestra salida. Hemos comentado el concepto de polimorfismo en programación orientada a objeto, y hemos visto cómo incorporar gráficos a nuestros programas.

EJERCICIOS

6.1 Incluir gráficos en el programa 3.2 “Adivinar un número”, de forma que en lugar de indicaciones con palabras, el programa muestre una flecha arriba para demasiado bajo, una flecha abajo para demasiado alto, y muestre una medalla o un certificado cuando el usuario acierte.

6.2 Hacer una versión gráfica del programa “Piedra, papel, tijeras”. Se representará con una imagen la elección del usuario y la del ordenador y se imprimirá quién es el ganador. El juego se seguirá jugando hasta que el usuario pulse un botón “Fin”. Se llevará un registro de las jugadas y del número de aciertos del jugador y del ordenador.

6.3 Escribir un programa que presente una mano de poker. Deberá elegir aleatoriamente 5 cartas de entre las 52 de la baraja y presentar gráficamente la carta elegida (no se pueden repetir las cartas). Se pueden usar imágenes de cartas descargadas de Internet o crear una baraja propia.