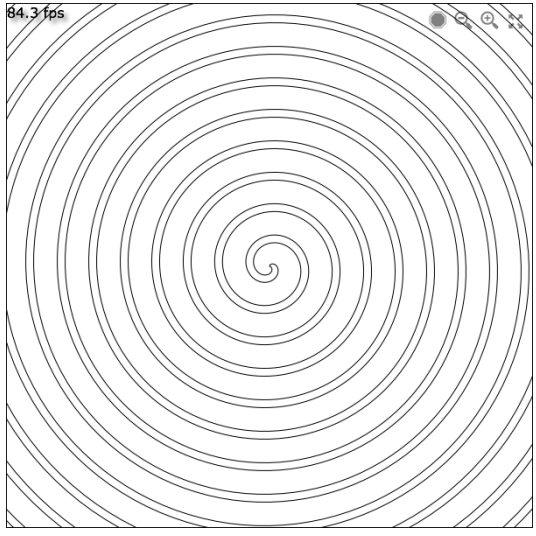
Capítulo 7
Los métodos de diseño

1. La etiqueta Canvas

El archivo **7_1_espiral.html** muestra una espiral animada. El diseño de la espiral se ha detallado en el capítulo La Web, y ahora agregaremos cierta interacción mediante JavaScript para modificar la velocidad de rotación de la espiral, por ejemplo.



La captura de pantalla anterior muestra en su parte izquierda información relativa al número de imágenes por segundo (o *frames per second*, fps) y a la derecha cuatro botones. El primero permite detener la animación, el siguiente ralentiza la animación, el tercero la acelera y el último permite pasar a pantalla completa.

El código HTML no contiene nada nuevo.

```
<body onload="dibuja()" >
    <div id="fullscreen">
        <canvas id="espiral_id" width="600" height="600"></canvas>
        <span class="fs-button"></span>
        <span class="mas_btn"></span>
        <span class="menos_btn"></span>
        <span class="stop_btn"></span>
    </div>
</body>
```

Tenemos, por tanto, un `div` para la pantalla completa, la etiqueta `canvas` para el dibujo de la espiral y, a continuación, cuatro `span` que permiten mostrar los botones.

Los estilos de las clases de los `span` utilizan la tipografía ModernPictogramsNormal para mostrar los pictogramas.

Existe un evento `onload` en la etiqueta `body` que permite poner en marcha la animación invocando a la función `dibuja()`.

Se utiliza una nueva instrucción en la función `dibuja`, se trata de `setInterval`.

```
intervId = setInterval(reDibuja, 1);
```

Esta línea de código permite invocar a la función `reDibuja` cada milisegundo, es decir 1000 veces por segundo. La variable `intervId`, que es una variable global, permite poner fin a este `setInterval`. Una vez lanzado, el `setInterval` ejecutará cada milisegundo la función `reDibuja`, si no se ha previsto nada para detenerlo habrá que salir del navegador para pararlo todo.

Antes de ver lo que hace la función `reDibuja`, veamos el código que permite gestionar el teclado. Se ha diseñado utilizando jQuery y seguramente tendrá otras ocasiones para escribir este tipo de código que enriquece jQuery.

```
$(function() {
    $(document).keydown(function(evt) {
        if (evt.keyCode === 32) {
            if (girar_bool) {
                girar_bool = false;
                clearInterval(intervId);
            } else {
                girar_bool = true;
                intervId = setInterval(reDibuja, 1);
            }
        }
        if (evt.keyCode === 27) {
            girar_bool = false;
            clearInterval(intervId);
        }
    });
});
```

La primera línea accede a jQuery y permite agregar una nueva funcionalidad. La segunda línea, con el `keydown`, intercepta las teclas del teclado que se presionan. Cuando se produce el evento, se almacena el código correspondiente a la tecla presionada en la propiedad del evento, `keyCode`.

Para aquellos que conozcan ASCII (*American Standard Code for Information Interchange*, que es el código americano normalizado para el intercambio de información, es decir, una norma de codificación de los caracteres; para obtener más detalles, consulte: http://www.asciitable.com), la primera comprobación se realiza con la barra de espacio, que tiene como `keyCode` 32.

En distintos lugares del código, existe un valor booleano `girar_bool` que se lee y/o actualiza, indicando si la espiral gira en este momento o no.

Si se presiona la barra de espacio y `girar_bool` vale `true`, entonces el valor booleano pasa a valer `false`, y se invoca a la función `clearInterval(intervId);`.

Es `clearInterval(intervId);` el que pone fin a `setInterval` definido anteriormente.

Si la tecla pulsada tiene como `keyCode` 27, se trata de la tecla [Escape] (o esc) del teclado, que detendrá la animación.

La siguiente sección del código, con:

```
$(document).ready(function() {
      ...
}
```

permite inicializar el `click` sobre los distintos botones: el paso o el fin de la pantalla completa para el primero, y luego `menosRapido();`, `masRapido();` y `stopEspiral();` para los otros tres botones.

Observación

Observe que el código que permite pasar a pantalla completa se ha escrito únicamente para Chrome y Safari, es decir, los navegadores que utilizan webkit.

La llamada a la función `setInterval` es lo más rápida posible puesto que se ha configurado cada milisegundo, la solución para modificar la velocidad será aumentar o disminuir el paso entre cada sección del diseño de la espiral.

Tenemos, a continuación, la función `reDibuja()`, que se invoca 1000 veces por segundo. Define, simplemente, el sentido del dibujo de la espiral y principalmente invoca a la función `dibujaEspiral()` que se ocupa realmente de dibujar la espiral.

`dibujaEspiral()` empieza con la línea de código:

```
canvas.width = canvas.width;
```

Esta línea permite, simplemente, borrar el `canvas`. Es el caso con muchos sistemas de diseño, donde se crea un diseño en primer lugar. A continuación, para animarlo, se borra el conjunto y se dibuja un poco más lejos, para crear de este modo la animación. El hecho de escribir que el ancho es igual a la altura permite reinicializar rápidamente el `canvas`, y de este modo borrarlo.

El código que permite dibujar la espiral se ha visto en el primer capítulo. Aquí, destacaremos que la línea siguiente:

```
var theta = anguloInicial;
```

se escribe en primer lugar para la primera espiral, y a continuación, una vez dibujada, el lápiz pasa al centro y se define un nuevo ángulo inicial opuesto al primero, agregando la mitad de un círculo (PI/2).

```
ctx.moveTo(0, 0);

// partimos de 90°
theta = anguloInicial + Math.PI / 2;
```

`canvas` permite mostrar una sombra de manera bastante sencilla. Es lo que hacen las líneas que terminan el dibujo. Las propiedades `shadowOffsetX` y `shadowOffsetY` definen las características de la sombra, `shadowBlur` el tamaño del flujo y el último parámetro define el color.

```
// sombra
ctx.shadowOffsetX = 4;
ctx.shadowOffsetY = 4;
ctx.shadowBlur = 5;
ctx.shadowColor = 'rgba(0,0,0,0.6)';
```

Una vez diseñado el conjunto, pasamos a la parte de texto para la velocidad de visualización.

Aquí se utilizan dos variables: `thisloop` y `lastloop`. `lastloop` se inicializa con la fecha completa al inicio del programa. Su precisión es del orden del milisegundo. Cuando el diseño de la espiral ha terminado, se almacena en `thisloop` la fecha en curso, que será mayor que `lastloop`, y la diferencia entre ambos valores equivale al tiempo de dibujo. Al final, la variable `thisFrameTime` contiene la diferencia entre `thisloop` y `lastloop`, si bien conocemos el tiempo que ha hecho falta para dibujar una espiral.

```
var text = (1000 / frameTime).toFixed(1) + " fps";
ctx.font = "12pt Verdana";
ctx.textAlign = "left";
ctx.textBaseline = 'top';
ctx.fillStyle = 'rgb(0,0,0)';
```

La visualización de la velocidad, en imágenes por segundo, parte del valor 1000, pues el tiempo recuperado está en milisegundos, y se divide entre `frameTime` para obtener un valor en segundos.

La instrucción `.toFixed(1)` permite redondear a una cifra decimal el resultado de la división.

A continuación, se utilizan las propiedades del `canvas` ligadas al texto para definir la tipografía utilizada y su tamaño, la alineación y el color.

2. La etiqueta SVG

El formato de diseño SVG permite dibujar mediante etiquetas. Puede crearse en un editor de código o exportarse mediante una aplicación de diseño vectorial.

Es posible, también, utilizar CSS para configurar el aspecto y scripts para las interacciones. En el siguiente ejemplo, el script es, realmente, ECMAScript.

Observación

A modo de cultura general, ECMAScript está en el origen de JavaScript. La tecnología Flash utiliza ActionScript, que parte también de ECMAScript, produciendo sintaxis muy similares.

Ejemplo de tooltip

Podríamos escribir un libro completo acerca del formato SVG. El siguiente ejemplo nos va a permitir hacernos una idea sobre lo que es posible realizar. El archivo **7_2_tooltip.svg** muestra una animación y algo de interacción.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg [
     <!ENTITY duracionAnimacion "2s">
]>
```

El encabezado del archivo SVG es XML. A continuación, la entidad (`ENTITY`) es el equivalente a una variable. Aquí, `duracionAnimacion` memoriza el texto "2s" que será la duración de la que haremos con SVG.

```
<svg xmlns="http://www.w3.org/2000/svg" onload="init(evt)"
width="480" height="300">

    <style type="text/css">
        .texto {
        font-family: Arial, sans-serif;
        font-size: 14px;
        }
        .tooltip {
        font-size: 12px;
        }
        .tooltip_bg {
        fill: white;
```

```
        stroke: #800;
        stroke-width: 2;
        opacity: 0.90;
        }
    </style>
```

El encabezado del SVG contiene un elemento `onload`, como con HTML. El CSS se parece en gran medida a lo que se utiliza en HTML.

Las instrucciones específicas de SVG son `fill` para rellenar una zona, `stroke` que define el color del trazo (generalmente el contorno de un objeto), y `stroke-width` para indicar la anchura del trazo.

El código prosigue con el script, pero es más sencillo ver, en primer lugar, las etiquetas que permiten comprender cómo funciona.

```
    <defs>
        <linearGradient id="degradadoRojo" x2="0%" y2="100%">
            <stop offset="0%" stop-color="white"/>
            <stop offset="50%" stop-color="red"/>
            <stop offset="100%" stop-color="#880808" stop-
opacity="0"/>
        </linearGradient>
    </defs>
```

La etiqueta `<defs>` permite definir efectos visuales, sombras, flujos... Aquí, se trata de un degradado que tiene como ID `degradadoRojo` y que podrá utilizarse en cualquier momento en las etiquetas SVG. `x2` e `y2` permiten definir la extensión del degradado. Aquí, se configura sobre el eje Y. Empieza con un color blanco, hacia la mitad del tamaño del objeto sobre el que se aplica será rojo, para terminar sobre el rojo y la transparencia.

```
    <text class="texto" x="10" y="25">El ratón pasa por
encima y</text>
    <text class="texto" x="10" y="45">hace aparecer un
tooltip.</text>

    <rect id="rect1" x="200" y="10" width="60" height="60"
fill="url(#degradadoRojo)"
          onmousemove="mostrarTooltip(evt, 'Un cuadrado rojo
degradado')"
          onmouseout="ocultarTooltip(evt)"/>

    <rect id="rect2" x="280" y="10" width="60" height="60"
```