|  |  |
| --- | --- |
|  | **2019** |
|  | 2º DAW  Adrián González Gómez |

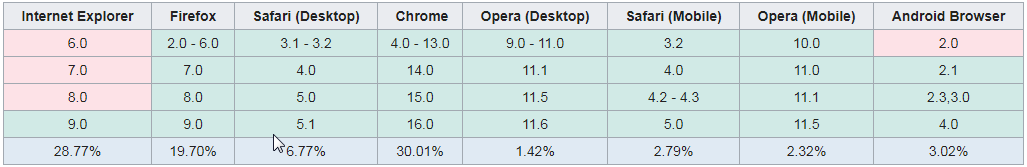
|  |
| --- |
| **[DES.WEB ENTORNO CLIENTE]** |
|  |

**INTRODUCCIÓN CANVAS**

Canvas (o lienzo traducido al español) es un elemento HTML incorporado en HTML5 que permite la generación de gráficos dinámicamente por medio del scripting. Entre otras cosas, permite la renderización interpretada dinámica de gráficos 2D y mapas de bits, asi como animaciones con estos gráficos. Se trata de un modelo de procedimiento de bajo nivel, que actualiza un mapa de bits.

El Canvas consiste en una región dibujable definida en el código HTML con atributos de altura y ancho. El código JavaScript puede acceder a la zona a través de un conjunto completo de funciones similares a las de otras APIs comunes de dibujo 2D, permitiendo así que los gráficos sean generados dinámicamente. Algunos de los usos previstos incluyen construcción de gráficos, animaciones, juegos, y la composición de imágenes.

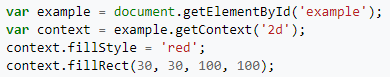
**COMPATIBILIDAD CON NAVEGADORES**



**Métodos y propiedades**

Width y height definen el tamaño del canvas en html, también se puede establecer el tamaño mediante javascript, buscando primero el elemento mediante “getElementById”





getContext(): canvas.getContext("2d");Devuelve un objeto que proporciona todos los métodos y propiedades para dibujar en el canvas. Para dibujar en 3d utilizamos “webgl”, pero no está implementado en todos los navegadores.

toDataURL(): canvas.toDataURL(tipo);Convierte el contenido del canvas en una imágen - data:uri. El parámetro entre paréntesis indica el tipo de imágen.image/png.

Para dibujar necesitaremos saber las coordenadas x e y de los puntos utilizados. Por ejemplo el código para dibujar un rectángulo es rect(x,y,w,h), donde x e y son las coordenadas de la esquina izquierda arriba, w es la anchura y h es la altura del rectángulo.  
En el canvas:  
- El punto cuyas coordenadas son: x=0; y=0; se corresponde con la esquina izquierda arriba del canvas.  
- Abajo en la esquina derecha x=canvas.width; y=canvas.height;

**LÍNEAS**

* Trazar una línea

Determinamos el grosor de línea con lineWidth, con strokeStyle definimos el color, comenzamos a dibujar con beginPath(), movemos el lápiz a un punto del canvas con moveTo(x,y), definimos la línea desde el punto anterior con lineTo(x,y). Con stroke() dibujamos la línea ya definida. Con translate(x,y) podemos dibujar líneas finas

* Puntas de línea (lineCap)

Los valores que puede tomar lineCap son "butt" (tope), "round"(redondeado) y "square" (cuadrado)

* Uniones de línea

Con lineJoin Determina (sets) o devuelve (returns) el aspecto de las juntas entre líneas. Posibles valores: bevel (biselado), round (redondeado), miter (en ángulo). Hay también una propiedad miterLimit que representa la distancia entre la parte interna y la parte externa del ángulo donde las dos líneas se encuentran. Puede tomar valores entre 1 (punta roma, aspecto biselado) y 5 (punta en ángulo).

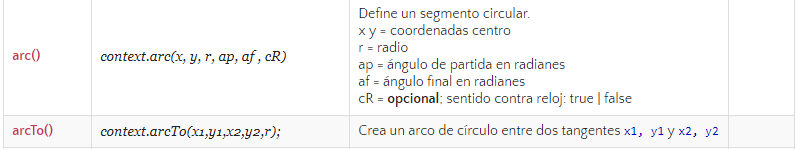
* Líneas discontinuas

Podemos dibujar líneas discontinuas en canvas utilizando el método  setLineDash(), que toma como argumento una lista de números separados por comas, que definen el patrón a seguir.  
Por ejemplo ctx.setLineDash([10, 5]) define una línea compuesta por fragmentos de línea de 10px separados por espacios de 5px.

lineDashOffset(context.lineDashOffset) Método que devuelve la lista de valores utilizada para crear una línea discontinua. Ejemplo: console.log(context.getLineDash());

**CURVAS**

* Arcos y círculos



* El método arcTo

Crea un arco de círculo entre dos tangentes x1, y1 y x2, y2

* Curvas cuadráticas de Bézier

quadraticCurveTo().Define una curva cuadrática de Bézier.  
cx,cy = coordenadas punto de anclaje (punto de control)  
x,y = coordenadas punto final (el punto de partida siendo determinado previamente).

* Curvas cúbicas de Bézier

bezierCurveTo(): Define una curva de Bézier.  
cx1,cy1,cx2,cy2= coordenadas puntos de anclaje P1 y P2 *(control points)*  
x,y = coordenadas punto final P3 (el punto de partida siendo determinado previamente).

* Dibujar un óvalo

arc(x,y,r,sA,eA,aC). Define un segmento circular.  
x y = coordenadas centro  
r = radio  
sA = ángulo de partida en radianes  
eA = ángulo final en radianes  
aC = sentido contra reloj (anti-Clockwise) true | false

scale(h,v) Reduce o amplía a escala el dibujo actual.  
h = horizontal; v = vertical

* Dibujar una elipse

ellipse(X, Y, rX, rY, ar, ap, af, cR); X y Y son las coordenadas del centro,   
rX y rY representan el radio en x y el radio en y,  
ar representa el ángulo de rotación del eje horizontal (en radianes),  
ap es el ángulo de partida ( en radianes ),  
af es el ángulo final ( en radianes ), y  
cR en el sentido del reloj ( false ) o en sentido contrario ( true )

* Otras curvas

Podemos dibujar un círculo como un polígono regular con 360 lados – por ejemplo. De hecho si el círculo es relativamente pequeño podemos reducir considerablemente el número de lados.

* Dibujar espirales

Para dibujarla necesitamos saber calcular las coordenadas x e y de un punto en la curva, y para esto utilizamos la misma fórmula que aplicamos para calcular un punto en la circunferencia de un círculo, solo que en este caso la distancia desde el centro **r** varía en función del ángulo **a** (en radianes). Los puntos cx y cy representan las coordenadas del centro alrededor del cual gira la espiral.

**TRAZADOS**

* Cerrar un trazado

closePath():Cierra una línea poligonal.

* Rectángulos

Rect(): Define un rectángulo desde un punto (x,y).

fillRect():Dibuja y rellena un rectángulo desde un punto (x,y).

strokeRect():Dibuja un rectángulo desde un punto (x,y).

clearRect():Borra los píxeles especificados dentro de un rectángulo dado.

* Dibujar un hexágono

Para construir un hexágono trazaremos un circulo imaginario con el radio R y el centro en un punto (X,Y) que en nuestro caso este punto coincide con el centro del canvas.  
A continuación tenemos que encontrar 6 puntos equidistantes situados en la circunferencia. Estos puntos están situados a 60° uno de otro. Si unimos estos puntos conseguimos un hexágono.

* Regiones de recorte

Clip():Recorta una región con la forma y tamaño del trazado dibujado previamente en el canvas. Cualquier cosa dibujada después, será visible solo dentro de la región de recorte.  el contexto de canvas no acepta más que una sola región de recorte.

* Combinar trazados

Para combinar trazados en el canvas podemos utilizar la propiedad globalCompositeOperation. Esta propiedad define la apariencia de nuevos trazados, y como estos afectan o están afectados por los trazados ya existentes en el canvas.  
Para combinar trazados la propiedad globalCompositeOperation puede tomar 12 valores diferentes. Si utilizamos 2 imágenes tendremos una imagen de origen y otra de destino (source/destination)

**RELLENO**

* Color de relleno

fillStyle: Determina o devuelve el color, gradiente o patrón del relleno.

fillRect(x,y). Dibuja y rellena un rectángulo desde un punto (x,y).

fillText(text,x,y,maxWidth): Dibuja texto relleno con un color, gradiente o patrón previamente definido.  
maxWidth es opcional. No se admite en Safari.

Fill():Rellena una forma geométrica.

* Gradiente lineal

createLinearGradient(x0,y0,x1,y1): Crea un gradiente lineal para utilizar en el canvas  
x0,y0 son las coordenadas del punto donde empieza el gradiente.  
x1,y1 son las coordenadas del punto donde acaba el gradiente.

addColorStop(color,stop): Especifica los colores y la posición donde para el gradiente.

* Gradiente radial

createRadialGradient(x0,y0,r0,x1,y1,r1): Crea un gradiente radial para utilizar en el canvas  
x e y son las coordenadas del centro de los circulos  
r es el radio de los circulos.

* Como crear patrones

createPattern():Repite una imagen en la dirección especificada.

* Aplicar sombras

ShadowColor():Determina (sets) o devuelve (returns) el color utilizado para las sombras.

ShadowBlur():Determina (*sets*) o devuelve (*returns*) el nivel de desenfoque de las sombras.

ShadowOffsetX. Determina (*sets*) o devuelve (*returns*) la distancia horizontal entre la sombra y la forma que la genera. El valor tiene que ser > 0 para que la sombra tenga efecto.

ShadowOffsetY. Determina (*sets*) o devuelve (*returns*) la distancia horizontal entre la sombra y la forma que la genera. El valor tiene que ser > 0 para que la sombra tenga efecto.

* Transparencia (globalAlpha)

globalAlpha: Determina (*sets*) o devuelve (*returns*) el valor alfa o la transparencia actual del dibujo.

**TEXTO**

* Escribiendo en el canvas

Font: Determina (*sets*) o devuelve (*returns*) las propiedades del texto ( *font-style, font-variant, font-weight, font-size, font-family* )

FillText(text,x,y,maxWidth): Dibuja texto relleno con un color, gradiente o patrón previamente definido.maxWidth es opcional.

StrokeText(text,x,y,maxWidth): Dibuja texto bordeado con un color, gradiente o patrón previamente definido.maxWidth es opcional.

* Alineación horizontal

textAlign: Especifica el tipo de alineación horizontal del texto.

* Alineación vertical

textBaseline: Determina (*sets*) o devuelve (*returns*) la alineación vertical del texto.

* Medir texto

measureText(text): Devuelve un objeto que contiene la anchura del texto especificado entre paréntesis. Podemos hacer saltos de línea.

* Texto con reflejo

Para crear texto con reflejo utilizaremos otros dos métodos de canvas *translate()* y *scale()* También aplicaremos transparencia con *globalAlpha.*

**TRANSFORMACIONES**

* Transformaciones

Translate(x,y): Mueve el origen (0,0) del canvas en un punto dado (x,y).

rotate(angulo): Gira los trazados posteriores un ángulo dado (en radianes). El punto alrededor del cual gira coincide con el origen del canvas (0,0)

Scale(h,v): Reduce o amplía a escala el dibujo actual.h = horizontal; v = vertical.

Transform():Cambia los trazados posteriores, cambiando la matriz de estos.

setTransform():Reinicia el canvas a los valores iniciales, antes de proceder a cambiar los trazados posteriores.

**IMÁGENES**

* Trabajar con imágenes

drawImage(img,x,y): Dibuja una imagen en el canvas desde el punto (x,y)

drawImage(img,x,y,w,h): Dibuja una imagen en el canvas desde el punto (x,y), donde w y h son el ancho y el alto de la imagen, respectivamente.

drawImage(img,sx,sy,sw,sh,x,y,w,h): Recorta la imagen empezando desde un punto (sx, sx), sw y sh siendo el ancho y el alto de la zona recortada.  
Dibuja esta imagen en el canvas desde el punto (x, y), w y h siendo el ancho y el alto de la imagen resultante.

* Imágenes reflejadas

 Consiste en redimensionar el contexto del  canvas para darle la vuelta: ctx.scale(**-1**,1); pero primero tenemos que trasladar el contexto exactamente dos alturas de imagen (2\*this.height). De otra manera el contexto quedará totalmente fuera del lienzo. Las imágenes tardan en cargarse, y aunque esta demora fuera de solo una fracción de segundo, probar a utilizar una imagen inexistente, resultaría en error. Para que esto no pase tenemos que escribir una función anónima que dibuja la imagen con drawImage(), y enlazar esta función como retrollamada ( *callback* ) al evento onload.

* Crear una marca de agua

Guardamos el texto en una variable, establecemos alineación y estilo del texto, calulamos el tamaño del texto para que después de escribirlo la anchura total del texto sea menor que la del canvas

* Manipular imágenes

El canvas nos permite modificar, uno por uno, los píxeles de una imagen. Para esto tenemos a nuestra disposición tres propiedades y tres métodos() de canvas

* El negativo de una imagen

Para sacar el negativo de una imagen en colores, hay que acceder uno por uno cada pixel de esta imagen y cambiar el color del pixel por el color complementario.  
Esta vez no manipulamos toda la imagen sino una zona. El método putImageData() toma en este caso siete argumentos:

- la imagen manipulada imgData,   
- las coordenadas x e y de donde poner la imagen en el canvas   
- las coordenadas dirtyX y dirtyY de dónde empezar a manipular la imagen   
- la anchura y la altura del rectángulo manipulado.

* Imágenes en blanco y negro

Para convertir a blanco y negro una imagen en colores, hay que manipular uno por uno cada pixel de esta imagen y cambiar el color del pixel por el valor calculado de la luminosidad ( *brightness* ) del mismo. Podemos considerar la luminosidad como un promedio de los valores de cada color.

* Imágenes en sepia

Muy parecido a convertir en blanco y negro. Una vez calculada la luminosidad modificaremos ligeramente el valor de cada componente de color para obtener los tonos de sepia.  
No hay que olvidar que el valor máximo de cada componente cromático es de 255. Para no pasarnos de 255, utilizaremos Math.min para escoger el mínimo entre el valor calculado del componente cromático y 255.

* Modos de fusión

En imágenes digitales los modos de fusión ( *blend modes* ) son utilizados para determinar cómo se fusionan dos capas. Para cada pixel, el color del elemento ( el color de origen o *source color* ) se combina o se fusiona ( *is blended*) con el color del fondo ( *the backdrop* ) para crear un nuevo color; y como que el color de cada pixel tiene un valor numérico, hay un gran número de posibles combinaciones.  
El modo de fusión por defecto en la mayoría de aplicaciones, simplemente esconde la capa inferior ( *de destino* ).

* Qué son los data:uri

Podemos incluir pequeños elementos de datos en línea, como si fueran referenciados hacia una fuente externa. Podemos utilizar data uri en html, css, jquery. Sintaxis: data:[<mediatype>][;base64],<data>

El <mediatype> puede ser algo como image/png para una imagen de tipo .png. El <mediatype> es opcional, y si lo omitimos, el <mediatype> por defecto es text/plain

Lo de base64 indica el tipo de codificación de los datos binarios. También es opcional. Podemos no especificarlo, como en este ejemplo: donde los datos son textuales.

* Crear data:uri desde canvas

El método toDataURL(tipo) convierte el contenido del canvas en una imagen data:uri. El parámetro entre paréntesis indica el tipo de imagen y puede tomar uno de los siguientes valores:

* Faviconos con canvas

Empezamos creando un canvas flotante, que se encuentra fuera de pantalla. Este canvas no pertenece al DOM ( **D**ocument **O**bject **M**odel ) y por lo tanto no será visible.  Dibujamos en el canvas. Creamos el favicono. Para esto creamos un nuevo elemento <link>. Añadimos los atributos; el valor del atributo href del favicono es la imagen dibujada en el canvas pasada a data:uri. Finalmente añadimos el favicono ( appendChild(favicon) ) al <head> de nuestra página.

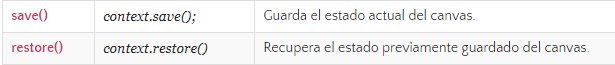
* Videos en el canvas

Para pintar un video en el canvas utilizamos el evento play que se dispara mientras que el video se está reproduciendo. Si el video está en pausa o se ha acabado ( if(vid.paused || vid.ended) ) no hay que hacer nada ( return ).  De lo contrario pintamos el video como si se tratara de una imagen.: drawImage();

**OTRAS**

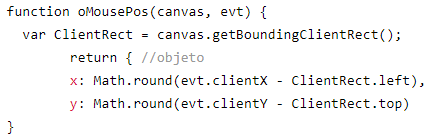
* Métodos sabe y restore

El estado del contexto del canvas puede ser manipulado mediante:

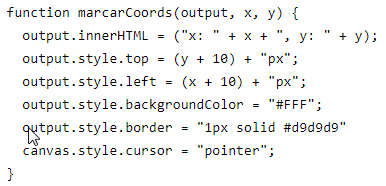
**

* La posición del ratón

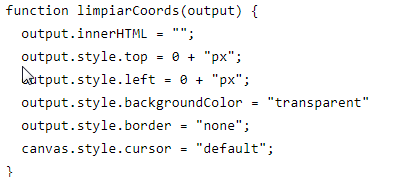
Para obtener la posición del ratón encima del canvas creamos la función oMousePos() que devuelve un objeto con las coordenadas x e y del ratón.  
Esta función utiliza el método *getBoundingClientRect()*



Usamos 2 funciones más, marcarCoords() que da formato al <div id="output"> donde aparece el valor de las coordenadas.



y limpiarCoords() que borra el contenido del <div id="output">



* Método isPointInPath

IsPointInPath(x,y): Detecta si un punto cuyas coordenadas son x e y se encuentra en un trazado dado.

* Crear un canvas adaptativo

 El canvas ( HTML5 ) no es un elemento que se adapte fácilmente a los diferentes tamaños de pantalla. No es "responsive". Las  dimensiones del canvas vienen especificadas en pixeles en el HTML. Podemos redimensionar el canvas con CSS, pero sería como una imagen redimensionada o con JavaScript, para ello en el HTML no especificamos las dimensiones del lienzo.

El método getComputedStyle() devuelve un objeto ( CSSStyleDeclaration ) con todos los estilos computados del elemento. Utilizamos este método para calcular la anchura y la altura del canvas> redimensionado.

* requestAnimationFrame

Para animar objetos en JavaScript, hasta hace muy poco se utilizaba ( y todavía se utiliza mucho ) el método setInterval().requestAnimationFrame(), es un método que genera animaciones fluidas, que paran en pestañas ( *tabs* ) inactivas, y es mucho más eficiente en los navegadores. Para utilizar este método:

Creamos una función que genera una nueva fotograma.  
Llamamos esta función utilizando requestAnimationFrame una vez para iniciar la animación.  
Después de un momento, la función volverá a llamarse a sí misma de manera recurrente.

El método requestAnimationFrame() devuelve un id, que podemos utilizar para parar la animación. Para esto utilizamos el método cancelAnimationFrame(id),para reiniciar la animación podemos utilizar un evento, como por ejemplo click:

* Trazados SVG en canvas

En SVG(Gráficos vectoriales) el elemento path se utiliza para dibujar formas complejas, una combinación de líneas, arcos y curvas. A diferencia de canvas donde los gráficos son dibujados de manera programática, en el SVG los elementos pertenecen al DOM ( Document Object Model ).

El método Path2D() es un constructor que devuelve un nuevo trazado. Si trabajamos con trazados generados con el método Path2D(), podemos especificar como argumento el trazado al cual nos referimos