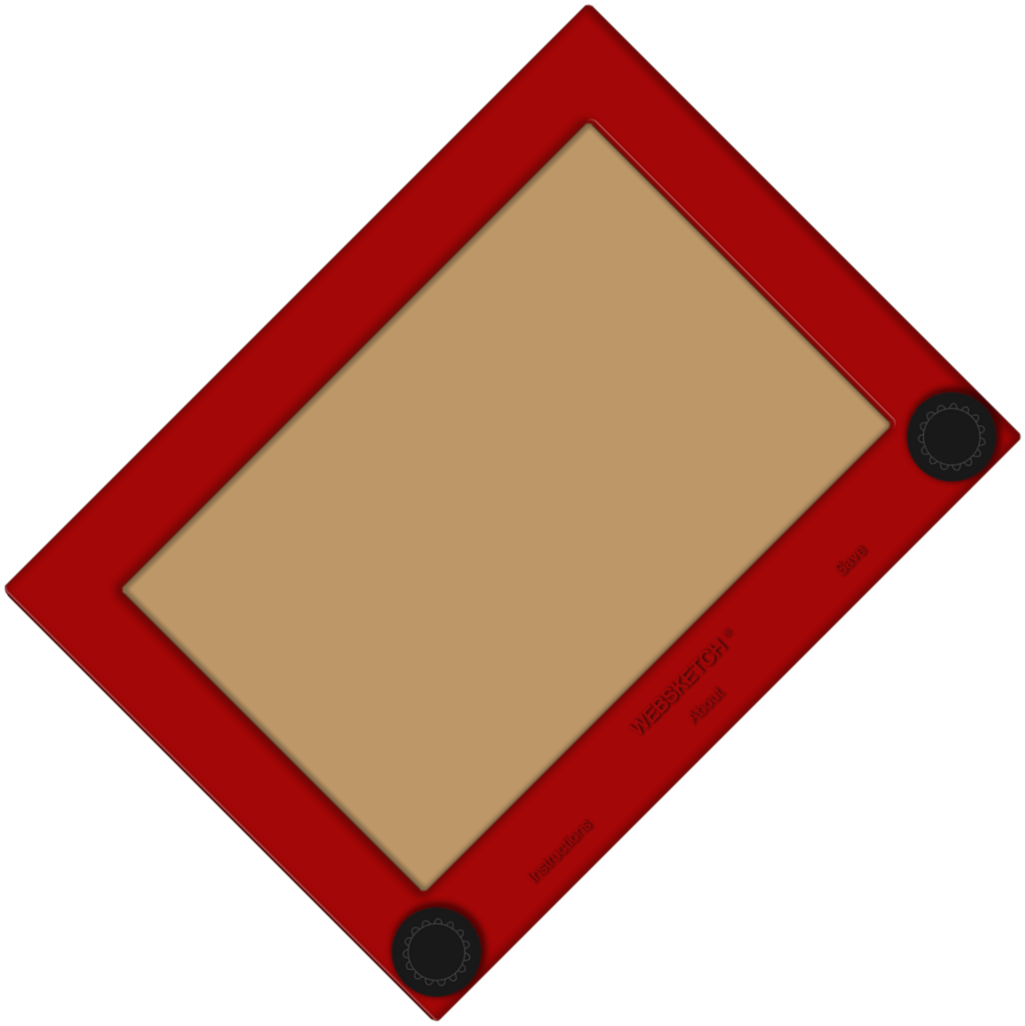


Websketch

La versión HTML5, CSS3 y JavaScript del famoso juguete para dibujar

Memoria del proyecto final del alumno **José Pujol** realizado para el curso de Diseño Web y Desarrollo Web Adaptativo con HTML5, CSS3, JavaScript y JQuery de la escuela CICE

ÍNDICE

INTRODUCCIÓN

Este documento forma parte del proyecto de final del alumno José Pujol del curso de Diseño Web y Desarrollo Web Adaptativo con HTML5, CSS3, JavaScript y JQuery de la escuela CICE (www.cice.es) impartido por el profesor Enrique Blasco Blanquer.

Se pueden encontrar todos los archivos que se mencionan en este documento en la memoria USB que se adjunta a la versión impresa de esta memoria y se puede disfrutar de Websketch en este enlace:

http://www.mesadeluz.es/websketch

Websketch se ha construido y colgado en internet sin ánimo de lucro. El símbolo de copyright que aparece junto al nombre de Websketch en la página web ha sido añadido por razones estéticas y no tiene valor legal. Animo a los dueños de los derechos del juguete físico a que se pongan en contacto conmigo si tienen cualquier duda o reclamación.

¿QUÉ ES WEBSKETCH?

Websketch es una reproducción en formato web del famoso juguete de dibujo Telesketch. Toda la estructura de la web esta realizada en HTML5, el estilo del diseño en CSS3 y los aspectos funcionales dinámicos en JavaScript con el apoyo de la biblioteca JQuery.

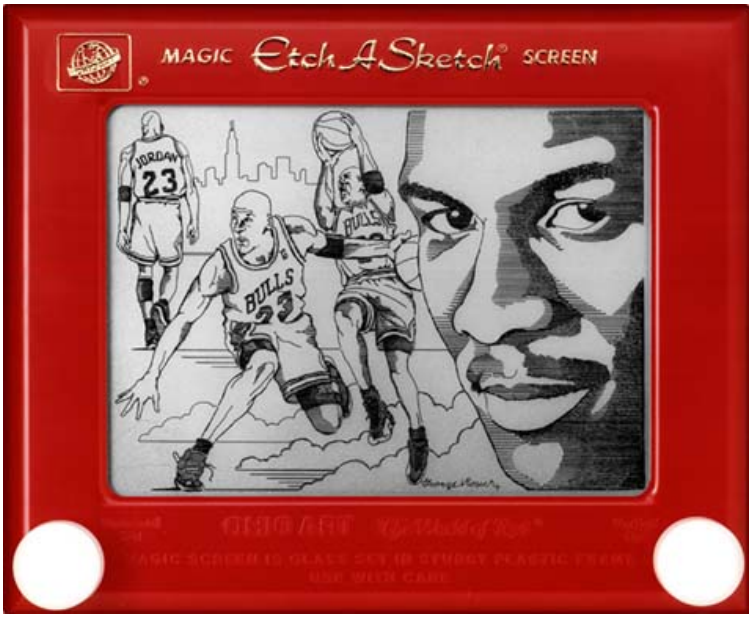
¿QUÉ ES UN TELESKETCH?

Ya que Websketch es una réplica web del juguete real Telesketch, dedicaremos un momento a aprender cómo es, qué funcionalidades tiene y cómo funciona este juguete. De este modo, podremos transportar estas cualidades y funciones a nuestra réplica web.

De ahora en adelante cuando hablemos del 'Telesketch' o del 'juguete' estaremos haciendo referencia al Telesketch físico y cuando hablemos de 'Websketch' o 'web' estaremos refiriéndonos a la página web protagonista de este proyecto.

HISTORIA DEL TELESKETCH

El Telesketch es un juguete inventado en 1959 por el francés André Cassagnes y que salió al mercado con el nombre de Etch-A-Sketch en las navidades de 1960 fabricado por la empresa estadounidense Ohio Art Company. Desde entonces, ha marcado la infancia de varias generaciones siendo el más famoso de todos los juguetes de dibujo del mundo.



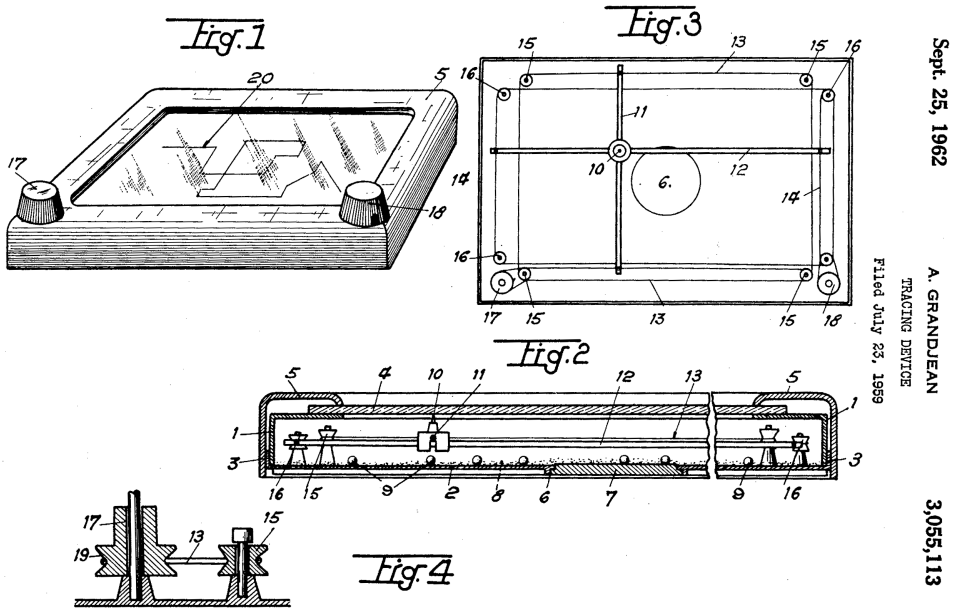
En España fue comercializado por la marca Borras con el nombre de Telesketch, haciendo referencia a su parecido con una pantalla de televisión. Hoy su empresa heredera Educa Borras sigue fabricándolo.

En 1987 aparece una versión electrónica llamada Etch-A-Sketch Animator con una pantalla pixelada y que permitía guardar varios dibujos y con ellos construir una animación. Esta versión y una posterior presentada en 1988 que incluía un sistema de cartuchos, dejaron de fabricarse debido a su fracaso de ventas.

En 2010 se estimó que se habían vendido 150 millones de unidades de Teleskecth en sus diferentes formatos y ediciones. En la actualidad sigue fabricándose y se puede encontrar en cualquier juguetería de tu ciudad.

¿CÓMO FUNCIONA UN TELESKETCH?

Un Telesketch se compone de un marco, una pantalla y dos botones giratorios que hacen moverse un punzón de manera horizontal y vertical por la parte interior de la pantalla. La superficie interior de esta pantalla esta recubierta de polvo de aluminio y partículas de estireno en la que la punta metálica va realizando un surco. De esta manera, en el exterior se puede ver como se dibuja una línea negra en la pantalla.



Para borrar el dibujo solo hay que poner el juguete boca abajo y agitarlo para que las partículas de aluminio y estireno vuelvan a recubrir la superficie interior rellenando los surcos.

TELESKETCH Y WEBSKETCH
Analizando el funcionamiento del juguete podemos encontrar ciertas similitudes con nuestras computadoras y sus lenguajes de programación. Estas similitudes fueron el punto de partida y la inspiración que me llevaron a comenzar este proyecto.

Una pantalla de Telesketch no deja de ser una pantalla de ordenador y, como podemos ver en la versión electrónica que fracasó a finales de los 80, las partículas de aluminio adheridas al interior de la pantalla no dejan de ser píxeles.

Hacer diagonales o curvas suaves resulta muy difícil en el juguete y también lo es en Websketch. La técnica para poder dibujar con cierta precisión es ir haciendo filas de líneas como si se tratase de las líneas de píxeles de una pantalla de ordenador.

Me gustaría hacer referencia a esta frase tomada de una entrevista realizada en 2010 a Bill Killgallon, presidente de Ohio Art Company:

> *"Mentalmente, André Cassagnes estaba muy centrado en diseños que incluían los ejes de coordenadas 'X' e 'Y'. Esta es una de las razones por las que fue capaz de inventar el Etch-A-Sketch."*

El interior, el espíritu, la funcionalidad del juguete está inspirada y basada en el clásico eje de coordenadas 'X' e 'Y'. Un sistema matemático presente en cualquier lenguaje de programación de computadoras.

En principio, debería bastar con unos píxeles y una manera de desplazarnos por el eje de las 'X' y de las 'Y' para crear nuestro Telesketch web… para crear nuestro Websketch.

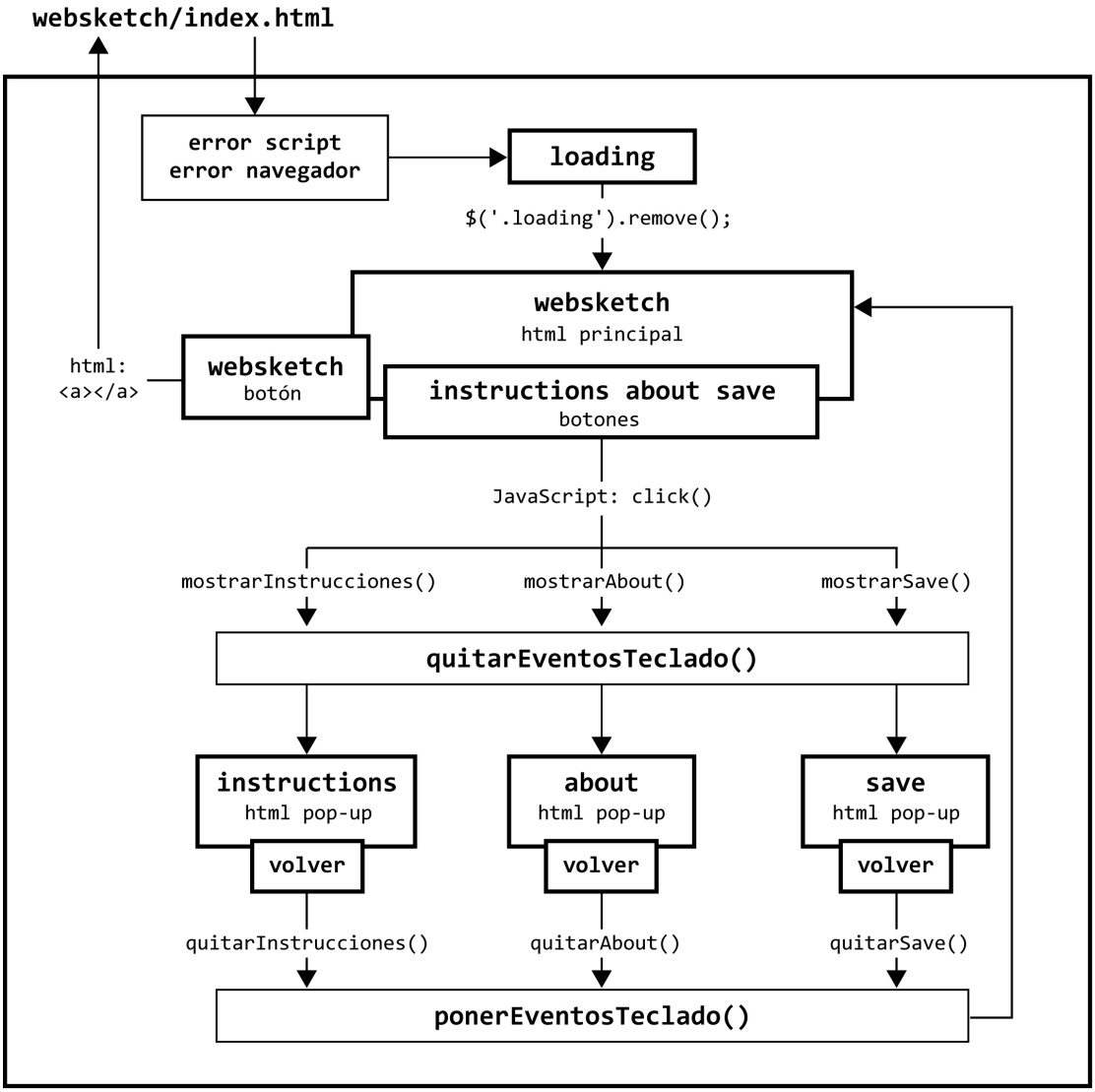
ÍNDICE NOSTÁLGICO

Este proyecto está motivado por un sentimiento nostálgico. Por eso he preparado este índice donde se enumeran las características que hacen especial al Telesketch, indicándose la página de la memoria donde se encuentra la respuesta que da Websketch a esa misma característica para intentar reproducir esa nostalgia en el visitante.

PUNTO DE PARTIDA
He trabajado durante todo el desarrollo en mi ordenador de sobremesa iMac de finales de 2009 con sistema operativo OSX y un monitor adicional que ha facilitado el desarrollo al permitirme mantener una pre-visualización abierta en todo momento.

HERRAMIENTAS
Para desarrollar y diseñar Websketch he utilizado una serie de herramientas. La principal ha sido NetBeans IDE en su versión 8.0 (www.netbeans.org) para crear el código HTML5, CSS3 y JavaScript. Apuntar que la versión utilizada de la librería JQuery ha sido la 1.11.1 que estaba disponible en su web al inicio del proyecto.

También he utilizado otros programas para ciertos elementos del diseño. Adobe Illustrator para los botones negros y el gráfico que explica el funcionamiento de Websketch. Y Adobe Photoshop para elaborar las estrellas de las pantallas de mensajes. A la hora de subir Websketch a mi servidor personal en www.mesadeluz.es he utilizado Transmit ftp client (www.panic.com/transmit/).



MAPA DE NAVEGACIÓN

La planificación y planteamiento de la navegación en este proyecto no ha sido un factor muy importante. Como se puede observar en el gráfico anterior, Websketch se centra en una única página en la que se muestra el juguete. En el tiempo en el que se carga Websketch, se muestra un mensaje de loading con información sobre su funcionamiento. El título del marco recarga la página y los tres textos inferiores de *'Instructions'*, *'About'* y *'Save'*, nos muestran una pantalla que esconde el Websketch. En realidad en ningún momento abandonamos la url www.mesadeluz.es/websketch/index.html

ESTRUCTURA – HTML

HypeText Markup Language o HTML, es un lenguaje para crear páginas web con textos, imágenes, audios y vídeos que pueden ser interpretadas y mostradas por un navegador. Se escribe utilizando etiquetas abrazadas por corchetes en ángulo <> que a su vez pueden abrazar un contenido creando un elemento dentro de la estructura. Las etiquetas pueden incluir atributos a los que se les pueden asignar valores.

CIMENTAR

La estructura HTML y sus diferentes elementos y etiquetas, son los cimientos de cualquier website y no es menos en Websketch. Es cierto que la parte más importante de este proyecto está centrada en las funcionalidades que simulan el juguete y que poco tienen que ver en principio con la estructura del HTML. En cualquier caso, me puse como reto el crear una estructura ordenada, sencilla, preparada para la creación de contenido de manera dinámica en su interior y que con cambios de estilo en CSS (que veremos más adelante) y el menor número de imágenes posibles, recrease el aspecto del juguete.

Todo el Websketch esta cimentado en un HTML a base de `<div>` si no tenemos en cuenta las pantallas de mensajes que si utilizan otras etiquetas de texto como `<h1>`, `<h2>` o `<p>` y para las imágenes `<img>`.

He enlazado en `<head>` el documento CSS con `<link>` y el código JavaScript y la librería JQuery con `<script>`.

ESTILO DEL CÓDIGO

Cualquier diseñador web/programador debe cuidar el estilo y formato de su código para facilitar su lectura y desarrollo. Por ello me he impuesto una serie de normas de estilo a la hora de escribir el código HTML:

- Utilización de comillas dobles
- Comentar principio y final de los principales bloques
- Estos comentarios siempre abrazan al bloque
- Vigilar las tabulaciones
- Incluir saltos de línea
- Los nombres de las `class` e *`id`* en minúsculas y sí se componen de dos palabras, separadas por guión bajo

META DATA

Al tratarse de una página web poco usual al no tener prácticamente etiquetas y contenido atractivo para buscadores como son los títulos, párrafos, imágenes o enlaces, he incluido una serie de información para facilitar la indexación de la página web y para que sea mejor compartida en Facebook.

Los metadatos, metadata o `<meta>` es una etiqueta especial que se incluye dentro de `<head>` y que permanece oculta de la pantalla del navegador pero que puede ser leída por ciertos programas.

Además de las dos tradicionales del idioma y formato de visualización, he incluido una `<meta name=”description”/>` que complementa al `<title>` siendo estas dos las etiquetas más importantes para el posicionamiento en buscadores. También he incluido metadatos especiales para Facebook para controlar las imágenes y textos que aparecen cuando un usuario comparte nuestra url.

FAVICON

He incluido un icono que representa un Websketch para que sea mostrado por el navegador junto a la pestaña o url. Este icono se denomina favicon y también se muestra en las listas del historial o favoritos de nuestro navegador ayudando a reconocer nuestro website.

Para que se cargue debemos incluirlo con la etiqueta `<link>` indicando su localización. Aunque el navegador lo busca en el mismo nivel de la página que estamos cargando por defecto, lo que no es muy útil si tienes una navegación compleja. Para nuestro website no es muy relevante.

MENSAJES

Como he comentado más arriba, no existe navegación por diferentes páginas web y el website Websketch se compone de una sola página y de un solo HTML. Esto implica que en este HTML se incluyan ciertos mensajes colocados al inicio del código para que, en caso de ser mostrados, oculten el Websketch.

Un factor muy importante para disfrutar de Websketch es que el usuario tenga activado en su navegador los scripts que permiten la ejecución del código JavaScript. A través de la etiqueta `<noscript>` se muestra el contenido que abraza en caso de que no estén activados los scripts en el navegador. He generado una pantalla que alerta para el usuario en este caso.

En el mapa de navegación que se incluye más arriba se pueden ver todos los mensajes que se incluyen en el HTML.

ESTILO - CSS
Cascading Style Sheet o CSS, es un lenguaje que da estilo a una estructura creada en HTML. Su sintaxis es muy sencilla y básicamente se compone de un selector que identifica a un elemento o elementos del HTML, seguido de una o varias propiedades colocadas entre las llaves `{}` a las que se les asigna un valor determinado.

En Websketch se ha incluido el CSS en un documento aparte que ha sido enlazado dentro del HTML como comentábamos más arriba.

IMITAR
El trabajo realizado con CSS en Websketch es un trabajo de imitación. He utilizado las diferentes propiedades que la versión 3 me brinda para recrear un Telesketch en una página web. Para ello he trabajado principalmente con los colores, sombras interiores, sombras exteriores, bordes negros y bordes blancos para crear volumen. He utilizado `border-radius` para redondear esquinas y he jugado con `position` y `z-index` para poder colocar ciertas cajas encima de otras.

Mediante Photoshop he elegido los siguientes colores principales, además del negro y blanco, para imitar el juguete original:

- Rojo: #A60808 RGB (166,8,8)
- Dorado: #BF9969 RGB (191,153,105)
- Letras Mensajes: #434145
- Fondo Mensajes: #EBF0EE

Cargamos ciertas fuentes que tenemos almacenadas en nuestro servidor mediante `@font-face` al inicio del documento. Destacar el recurso de utilizar texto del color rojo del marco con la propiedad `text-shadow` para simular las letras en relieve tan características del juguete.

Uno de los principales problemas de CSS es la dificultad que existe para centrar el contenido de manera vertical y automática como se consigue en posición horizontal con el sencillo sistema `margin: 0 auto`.

Ante la necesidad de centrar verticalmente mi Websketch, investigué por la red hasta dar con este sistema con la nueva propiedad `transform` y utilizando valores en % dentro de `translate` (incluyendo los webkits para los navegadores):

```
{
	position: absolute;
	top: 40%;
	-ms-transform: translate(0,-40%);
	-webkit-transform: translate(0,-40%);
	transform: translate(0, -40%);
}
```

La imagen de fondo que simula el suelo se colocó en una caja a parte para poder darle una sombra interior, ya que no funcionaba correctamente el efecto si se realizaba sobre la etiqueta `<body>`.

Como comentaba en el anterior punto sobre el HTML, solo he utilizado dos imágenes para los botones y una imagen repetida en el fondo para simular el suelo de madera. El resto del Websketch ha sido construido con `<div>` con determinados estilos en CSS.

Hay que tener en cuenta que gran parte de las dimensiones del Websketch se crean de manera dinámica desde JavaScript y se escriben en el CSS cuando el JS se ejecuta partiendo de la dimensión `width` del contenedor (como se indica en un comentario dentro del CSS). Más adelante en el apartado sobre JS se entra en el detalle de este punto.

Solo utilizo el sistema responsive `@media` para el mensaje que se muestra si la pantalla del navegador es menor que el tamaño del Websketch. Por el momento Websketch no es adaptative ni responsive y está ideado para funcionar solo en máquinas de sobremesa o portátiles con teclado. El camino de la adaptación a otros dispositivos como tablets y móviles, necesita de otro desarrollo que comentaremos más adelante en la parte dedicada a JS y en las conclusiones.

ESTILO DEL CÓDIGO

Al igual que con el HTML, me he marcado una serie de pautas para hacer el código CSS más ordenado y fácil de leer y modificar. Estas son algunas de las normas que he seguido:

- Comentar los principales bloques
- Crear índice con el número de línea de código donde se encuentra el bloque
- Tras el selector espacio antes de la llave
- Tras la llave de inicio salto de línea
- Solo una propiedad por línea de código (salvo en el reset)
- Salto de línea previo a la llave que cierra
- Utilización de comillas simples
- Comentar los webkits indicando para que navegador son
- Tres saltos de línea entre bloques
- Uno entre cada regla
- Espacios entre comas en listas de selectores
- Espacio tras los dos puntos antes de los diferentes valores

En todo el código se ha tenido en cuenta la posición de cada una de las reglas, ya que en CSS, la última prevalece sobre la anterior. Este factor fue importante a la hora de colocar las que afectan al color de los píxeles como veremos más abajo.

FUNCIONALIDAD – JAVASCRIPT

JavaScript o en su forma abreviada JS, es un lenguaje de programación interpretado que, en el lado del cliente, permite dinamizar la visualización de las paginas web. En la actualidad todos los navegadores son capaces de interpretar JS y su conocimiento es básico para el desarrollo de páginas web y su adaptación a dispositivos móviles.

JavaScript se compone principalmente de variables y funciones que son llamadas por eventos u otras funciones. Dentro de estas funciones podemos cambiar estilos CSS de determinados selectores, modificar variables o crear código en el HTML entre otras muchas cosas. Y también podemos ordenar que se realicen estas modificaciones solo si se cumplen determinadas condiciones que elijamos.

Pero la sintaxis y semántica de JS por si sola es bastante compleja, por ello he utilizado la biblioteca JQuery. Esta biblioteca gratuita y de código abierto facilita la selección de elementos del DOM, la inclusión de eventos, la obtención de información del navegador y la realización de efectos o animaciones, entre otras muchas cosas. Por así decirlo, JQuery contiene piezas complejas de JS que llamas con pocas líneas de código, ahorrándote de este modo mucho trabajo.

En el documento HTML está enlazado el archivo que contiene el código JavaScript y el archivo con la biblioteca JQuery, como hemos comentado mas arriba.

SIMULAR

El principal reto de este proyecto se basa en la creación de las diferentes funcionalidades en JavaScript que, trabajando sobre la estructura HTML y el estilo CSS existente, logre simular al juguete real Telesketch.

He construido una serie de funciones que son llamadas cuando suceden determinados eventos y que cambian las clases de determinados elementos del HTML dependiendo del estado de ciertas variables.

En resumen, he utilizado el lenguaje que se utiliza para desplegar un menú, cambiar la foto en un slider o cambiar el color de una caja cuando el ratón pasa por encima, para simular un juguete real de dibujo.

ESTILO DEL CODIGO

Como ocurre con el código en HTML y CSS, he seguido una serie de normas al redactar mi código JavaScript para hacerlo más fácil de leer y manipular. He cuidado el tabulado que ayuda a identificar las funciones anidadas y el encapsulamiento. Estas son algunas normas que me he impuesto:

- En los selectores utilizo las comillas simples
- Utilizo comillas dobles a la hora de seleccionar propiedades o cambiar clases en CSS o HTML desde JS
- Introduzco espacios entre la función y la llave
- No escatimar en saltos de línea
- Siempre salto de línea delante y detrás de llave

- Aunque los 'punto y coma' pueden ser omitidos tras las sentencias son incluidos para favorecer la lectura
- Los nombres de las funciones empiezan con minúscula y si se componen de dos o más palabras se escribe en mayúscula la primera letra de la nueva palabra sin espacios
- Las variables se nombran en minúscula y si contienen varias palabras se separan con guión bajo
- No se escatima en comentarios y los bloques principales incluyen asteriscos para resaltarlos
- Intentar declarar primero la función y luego llamarla (al parecer esto facilita la ejecución del código)

ENCAPSULADO

Como se puede apreciar en el Mapa General que encontraréis en un poco más adelante, he intentado encapsular el código en funciones para facilitar el desarrollo, la resolución de problemas y la futura implantación de nuevas funciones a Websketch.

He declarado las variables dentro de la función `websketch()`, pasando a ser variables locales en la teoría pero globales en la práctica dentro de nuestro Websketch. De esta manera, si incluimos en el futuro otras funciones que puedan tener el mismo nombre de variables que nuestro Websketch, no existirá ningún tipo de conflicto.

$(DOCUMENT).READY Vs $(WINDOW).LOAD

Todo el código JS es ejecutado tras `$(window).load` que fue la fórmula que finalmente escogí tras ciertas investigaciones en la red y algunas pruebas.

La diferencia principal entre los dos métodos, es que `$(document).ready` es llamado cuando el DOM ha sido cargado excepto las imágenes, siendo muy útil cuando tu JS no trabaja con esas imágenes o gráficos, mientras que `$(window).load` espera a que toda la página haya sido cargada, incluyendo imágenes.

Al necesitar que todas mis imágenes de los mensajes estuvieran cargadas, elegí la segunda opción.

DETECTORPANTALLAWEBSKETCH( )

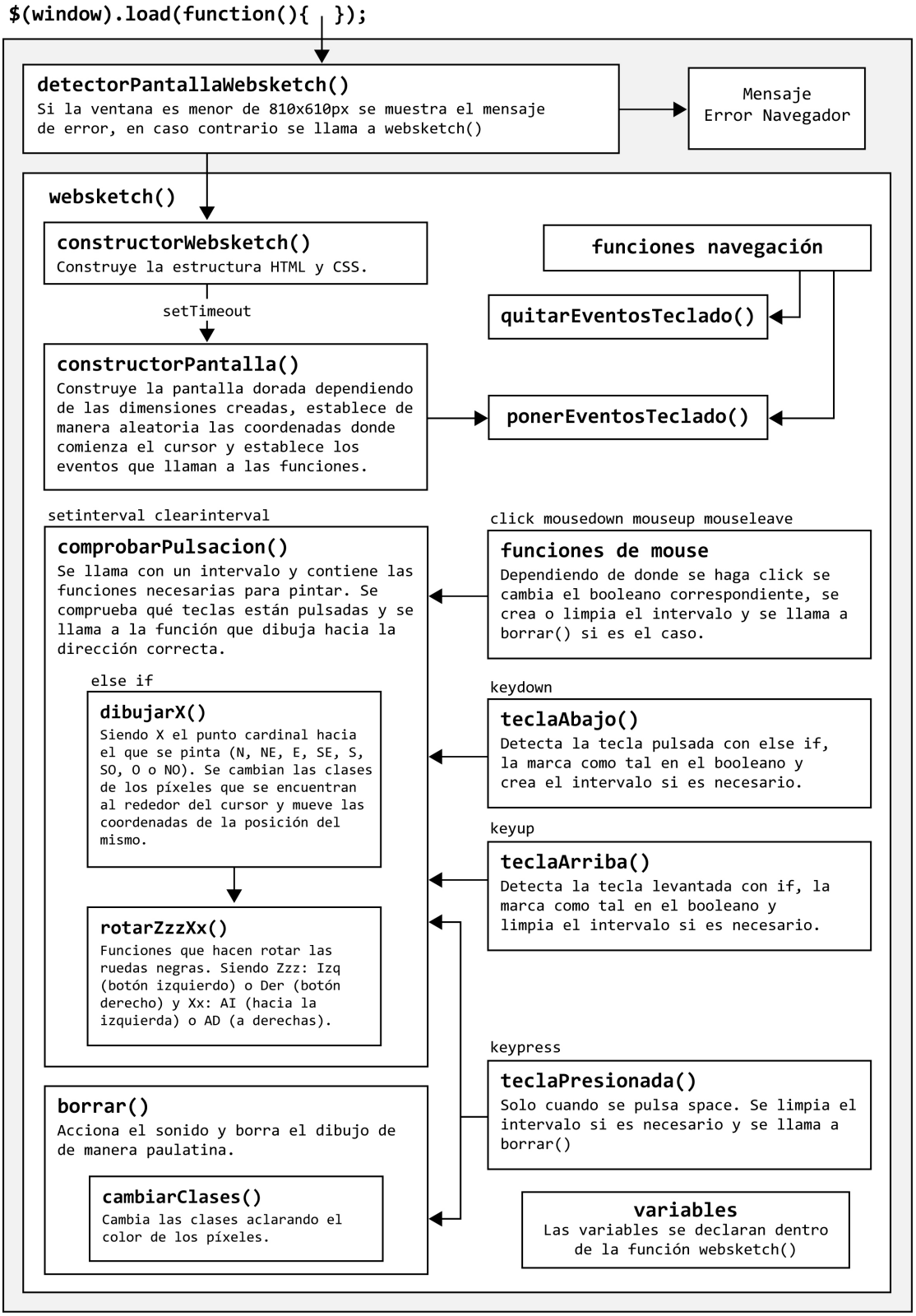
Por el momento Websketch esta pensado para trabajar con unas determinadas dimensiones. Es por eso que incluí esta función al inicio para detectar las medidas de la pantalla en la que se cargaba el Websketch y mostrar un mensaje de error si estas dimensiones eran inferiores a 810px de ancho o 610px de alto. JQuery nos permite adquirir estas dimensiones fácilmente con `$(document).width();` y `$(document).height();`

WEBSKETCH ( )

Esta función es el alma del Websketch y marca los límites del mismo en cuanto a JavaScript se refiere. Esta función sirve para encapsular todas la funciones que dan vida al Websketch y a sus variables.

MAPA GENERAL DE FUNCIONES

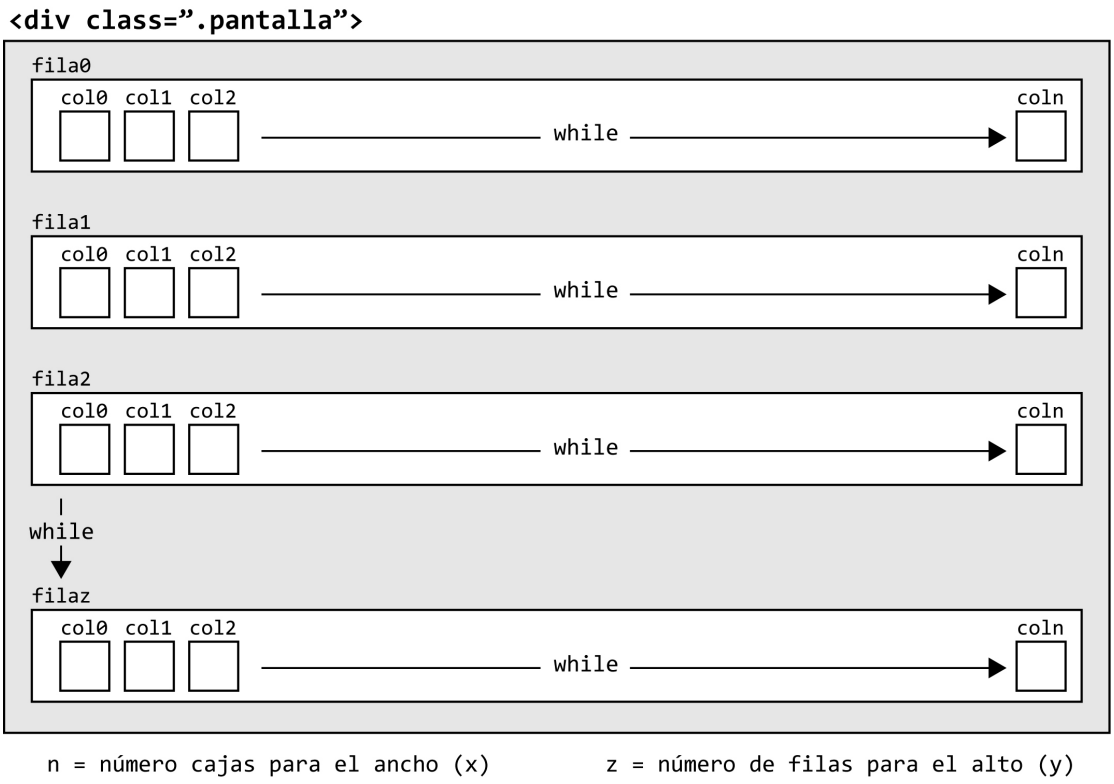
En este gráfico se encuentran las principales funciones que componen el código JavaScript, su utilidad y su situación dentro del mismo:



CONSTRUCTORWEBSKETCH( )
Esta función construye todo el Websketch trabajando sobre el CSS utilizando `.ccs()` y diferentes fórmulas matemáticas que nos brinda JS. La razón de ser de esta función es tener la capacidad de construir todo el aspecto y dimensiones del Websketch de manera dinámica a partir del tamaño del container. Se podría decir que es innecesario y que podríamos incluir directamente en el CSS esta información. Pero debemos tener en cuenta que:

- Al inicio del desarrollo no tenía claras las dimensiones del container y necesitaba poder jugar con estas medidas fácilmente
- En un futuro debería ser posible construir el Websketch a partir de las dimensiones del dispositivo en el que se carga

Por estas dos principales razones decidí incluir esta función y así construir el Websketch de manera dinámica.



CONSTRUCTORPANTALLA( )
Esta función construye la pantalla dorada donde vamos a dibujar a partir de las dimensiones creadas por la anterior función. La pantalla donde dibujamos no es más que una serie de cajas `<div>` de 1px de altura y anchura que incluimos en el HTML mediante el método `.append()` y que construimos de manera dinámica por dos razones:

- El número de filas y columnas viene determinado por el tamaño del elemento `.pantalla` cuyas dimensiones se crean de manera dinámica por la función `constructorWebskecth()` por razones anteriormente indicadas

- En cualquier caso, resulta mucho más fácil que JS escriba todas estas cajas en el HTML por nosotros. Hay que tener en cuenta que cada fila tiene un id diferente y cada caja dentro de esta fila también

Este es el sencillo código que nos pinta las 222.308 cajas de 1px dentro de las 373 cajas que conforman las filas:

```
ancho_pantalla = $('.pantalla').width();
alto_pantalla = $('.pantalla').height();

x_largo = ancho_pantalla - 1;
y_alto = alto_pantalla;

while (y < y_alto) {
      $('.pantalla').append('<div class="fila" id="fila'
      + y + '"></div>');

      while (x < x_largo) {
            $('#fila' + y).append('<div class="pixel"
            id="col' + x + '"></div>');
            x++;
      }

      x = 0;
      y++;
}
```

Los propios contadores que utilizamos para `while` los utilizamos para incluir las id de las cajas. `while` no es más que una manera más elegante que `if` para que una función se repita hasta que se cumpla una condición.

En esta misma función incluimos la posición aleatoria del cursor donde vamos a empezar a dibujar, es decir, determinados los valores de las variables `x` e `y` para comenzar a dibujar desde ahí asignando a esa caja la clase `puntero`.

También incluimos los eventos de mouse que sirven para dibujar y navegar, y llamamos a la función `ponerEventosTeclado()` que permite dibujar con las teclas.

Finalmente en esta función, retiramos el mensaje de loading que se muestra mientras el navegador realiza todas las tareas anteriormente enumeradas.

SETTIMEOUT(CONSTRUCTORPANTALLA, 1000)
La anterior función `constructorPantalla()` es llamada con un retardo de 1000 milisegundos con el método `setTimeout()`. En JavaScript las funciones son ejecutadas de manera asíncrona por el navegador, lo que implica que las tareas no se realizan una detrás de la otra y en ocasiones estas se pisan. Y que una función o sentencia esté por delante en el código JS no quiere decir que los resultados se muestren en el mismo orden en la pantalla del navegador.

Es tan grande el trabajo que `constructorPantalla()` pide al navegador que podríamos decir que eclipsa el resto de tareas. En ciertos navegadores, no se llega a mostrar el mensaje de loading o se muestra a medio construir el Websketch, dando la sensación que funciona mal el programa. Este problema es solucionado con `setTimeout()` que retrasa la llamada a `constructorPantalla()` dando tiempo al navegador a cumplir las anteriores tareas.

Me gustaría indicar que dar con la solución a este problema me llevo mucho tiempo de investigaciones y pruebas ensayo-error.

FUNCIONES DE NAVEGACIÓN

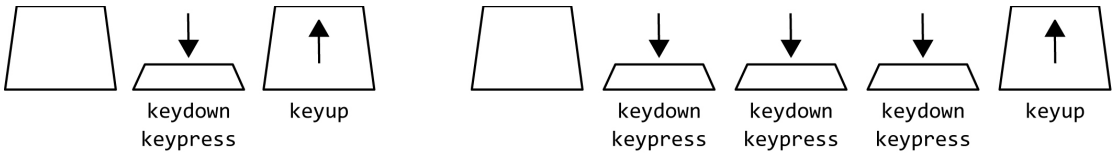
Llamadas por eventos click() muestran los diferentes mensajes de información adicional del Websketch o nos devuelven a nuestro dibujo.

La necesidad de poder inhabilitar el dibujo mientras se mostraban estas pantallas fue lo que me obligó a crear las funciones `quitarEventosTeclado()` y `ponerEventosTeclado()` ya que antes de entrar a construir el tema de la navegación no era necesario impedir el dibujar.

EVENTOS TECLADO

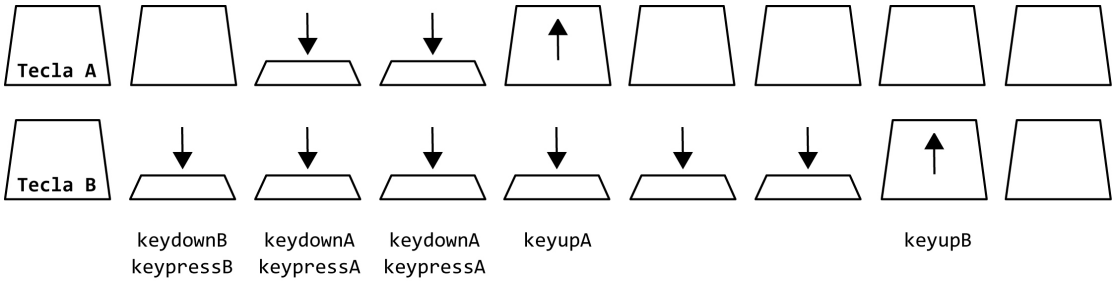
Uno de los principales retos de este proyecto ha sido la comprensión de los eventos de teclado que saltan cuando presionamos y soltamos las teclas y que pueden llamar a una función de nuestro código JS y que utiliza la librería JQuery. Cada vez que se pulsa una tecla, además de llamar a una función asociada a ese evento, nos devuelve un código en forma de número que se identifica con una tecla determinada. Esto nos permite identificar qué tecla se ha pulsado.

Para comprender mejor el funcionamiento y conocer el código que devuelve cada tecla, tome de http://www.desarrolloweb.com/aticulos/eventos-teclado-jquery.html un codigo HMTL que contenía un script que dibujaba en el HTML y en la pantalla el tipo de evento y el código de la tecla. Este HTML que utilice se puede encontrar en el proyecto de NetBeans adjunto a este proyecto con el nombre de `funcionamiento_eventos_teclado.html`



Al pulsar una tecla se crea un evento `keydown`, seguido de un `keypress` y al soltar se origina un evento `keyup`. Y al pulsar de manera continuada se van generando eventos `keydown` y `keypress` hasta que es levantada.

Este comportamiento podría valer en el caso de que solo quisiéramos dibujar arriba, abajo, izquierda o derecha, pero es necesario que nuestro Websketch dibuje también en diagonal. Por lo que debemos observar que comportamiento siguen los eventos al pulsar más de una tecla.



Al pulsar la segunda tecla A solo se devuelven eventos de esta tecla y al levantarla los eventos cesan y no hay noticia de la primera tecla hasta que esta es levantada.

Resulta evidente que estos eventos no nos permiten programar de una manera sencilla el dibujo en diagonal y también continuo. No podemos asignar una función a un evento, identificar que tecla ha sido pulsada y actuar sobre las cajas. Pero aun se complica más el asunto si analizamos el código de tecla que nos devuelve cada evento:

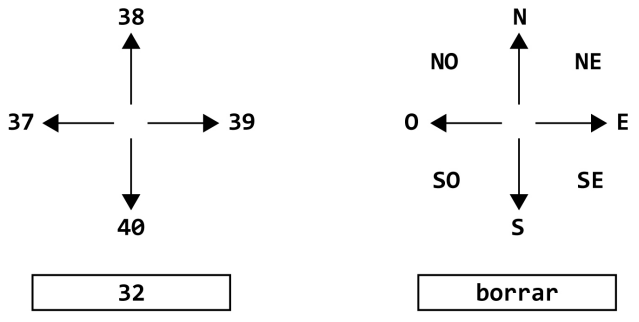
- Keydown – Keycode: Código Interno
- Keypress – Keycode: Código Carácter
- Keyup – Keycode: Código Interno

Dependiendo del evento, se nos devuelve un código diferente: 'Código Carácter' o 'Código Interno' siendo números diferentes.

Pero no terminan aquí los problemas relacionados con el comportamiento de estos eventos, existen una serie de peculiaridades asociadas a las llamadas teclas especiales como son ctrl, mayúsculas, alt y las teclas del cursor, estas últimas las elegidas para que se pudiera dibujar. Estas teclas no devuelven evento `keypress`.

La tecla elegida para borrar era la barra espaciadora que también tenía sus propias peculiaridades. Si presionas las teclas arriba y abajo al mismo tiempo y pulsas la barra, esta última no emite ningún evento. También había que tener en cuenta que el código carácter devuelto era diferente en Mozilla Firefox, punto a tener en cuenta a la hora de construir el JS.

En el siguiente gráfico se muestran los códigos internos devueltos por las teclas que nos interesan. Llegados a este punto decidí utilizar en el código JS los puntos cardinales para construir los nombres de las funciones, los comentarios y las diferentes variables. Como se ve en el gráfico, cada dirección para dibujar viene nombrada por los puntos cardinales.



Esta es la solución que he aplicado para poder dibujar en diagonal y de manera continua y que se entrevé en el Mapa General que se mostraba más atrás:

Existe una variable booleana que nos indica en todo momento si una de las teclas que nos interesan está pulsada o no. La función `teclaAbajo()` se encarga de realizar esta labor siempre que se pulsa en el teclado `keydown` y la función `teclaArriba()` realiza el trabajo en sentido contrario cada vez que se suelta una tecla y salta el evento `keyup`.

Siempre que hay alguna de las teclas que nos interesan pulsadas se crea un intervalo que llama a `comprobarPulsacion()`, función que dibuja teniendo en cuenta qué teclas están pulsadas.

Para borrar utilizamos el evento `keypress`, que sí devuelve nuestra barra espaciadora, y que solo debe impedir que se continúe pintando mientras está pulsada.

Con este sistema de booleanos, funciones y un intervalo, conseguimos dar vida a nuestro Websketch. Las pruebas del sistema las realicé con los archivos `evento_teclas_1.html` y `evento_teclas_2.html` incluidos en el proyecto de NetBeans adjunto a esta memoria.

IF Vs ELSE IF

Me gustaría explicar la diferencia entre las fórmulas `if` y `else if`. La primera va recorriendo todos los elementos hasta el final sin tener en cuenta si se han cumplido las condiciones anteriores y ejecutado el código que se incluye en su interior si es el caso. La segunda, deja de recorrer cada uno cuando el anterior elemento es verdadero, cuando cumple con sus condiciones.

En muchos casos, en el código de Websketch, en el momento que damos con las condiciones verdaderas ya no necesitamos comprobar el resto por que sabemos que ninguna va a encajar si una anterior ya lo ha hecho. El código funcionaría igual con `if`, pero es más eficiente si no cargamos a nuestro navegador con la tarea de seguir comprobando cada `if`, sobre todo cuando se le está pidiendo que realice muchas tareas como es el caso de nuestro Websketch.

TECLAABAJO( )

Esta función es llamada cuando una tecla es pulsada y detecta si es alguna de las teclas que nos interesan. Si es así, crea el intervalo de `comprobarPulsacion()` si no estaba ya creado (puede que otra tecla ya estuviera pulsada anteriormente).

Con `else if` va comprobando el código de la tecla, cambiando el booleano y si el intervalo no está activo, se crea.

TECLAARRIBA( )

Siempre que se suelta una tecla, el evento `keyup` llama a esta función. Con `else if` va comprobando el código de la tecla, cambiando el booleano a false. El intervalo se elimina en `comprobarPulsación()` y por eso no se incluye aquí.

TECLAPRESIONADA( )
Esta función se llama con el evento `keypress` que solo es devuelto por la barra espaciadora, por lo que es utilizada para borrar nuestro dibujo. Cambia el booleano para la barra espaciadora, elimina el intervalo si es necesario y llama a `borrar()`. No es necesario ningún intervalo para comprobar si la barra esta pulsada y así borrar, ya que para que actúe no se tiene que combinar con ninguna otra tecla y emite el evento `keypress` mientras esté pulsada (siempre que no se pulse las flechas arriba y abajo al mismo tiempo, cosa que es raro que suceda).

BORRAR( )
Esta función borra paulatinamente el dibujo del Websketch gracias a la función `cambiarClases()` y lanza el sonido de arena.

La información sobre el sistema para incluir audio y lanzarlo mediante JS la saqué de esta página web: www.storiesinflight.com/html5/audio.html y el sonido es un golpe de maraca de una biblioteca del programa de edición de sonido Pro Tools.

Comentado en el código JS aparece la siguiente línea comentada dentro de la función `borrar()`:

```
$(".container").animate({left:      "-1px"},      "10").animate({left:
"+1px"}, "10", cambiarClases());
```

Esta línea es una pieza básica de código con la que he jugado con tiempos y distancias para hacer que todo el Websketch se agitara cuando se borrase, imitando así la acción que realizamos con el juguete real. Pero el efecto no era el deseado. El navegador realizaba este efecto a trompicones y afectaba al efecto del borrado paulatino y al sonido.

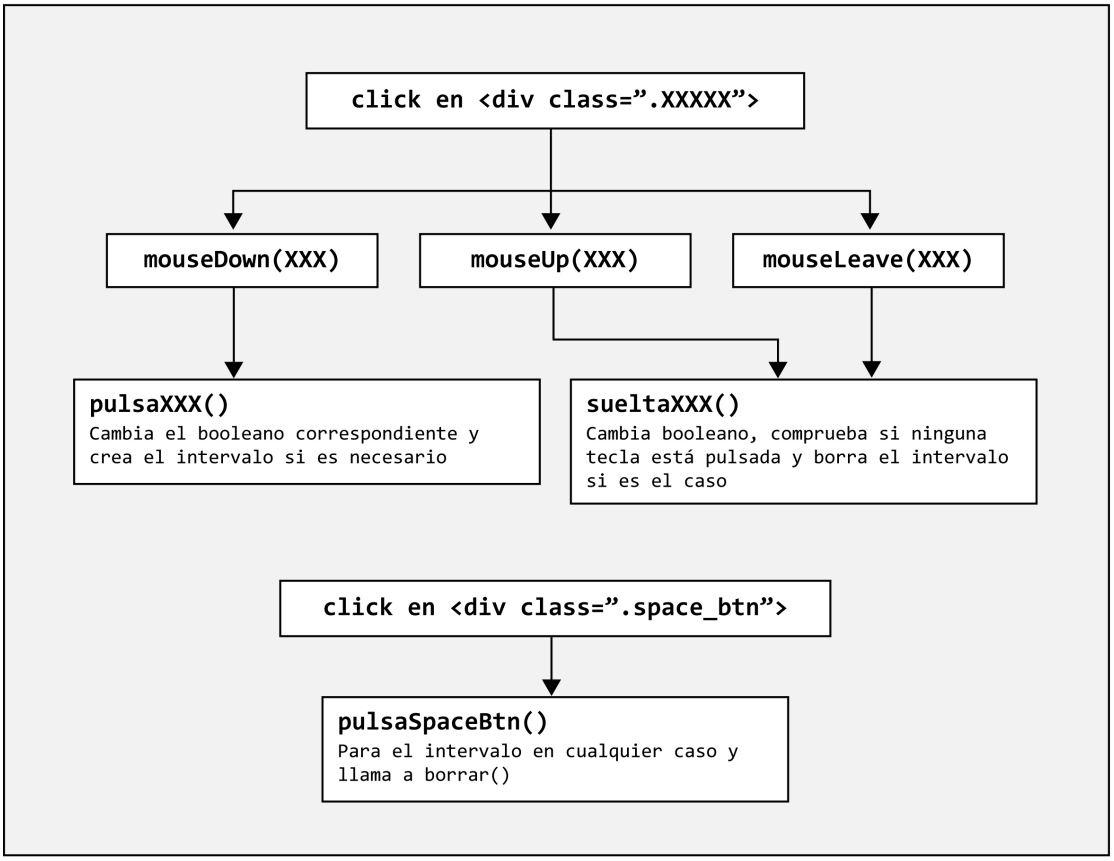
Analizando el problema, llegue a la conclusión que estaba pidiendo al navegador que moviese rápidamente no solo en marco rojo y la pantalla, sino también las miles de cajas de 1px que contiene la pantalla dorada. Los navegadores son incapaces de realizar esta animación con fluidez, por lo que la dejé comentada para implementarla en un futuro.

Una de las funciones y partes del código de la que estoy más orgulloso en cuanto a su estética al ser observada en el código JS es `cambiarClases()`. Para conseguir el efecto de un borrado paulatino debía ir cambiando las clases de las cajas en cascada de más claras a más oscuras. Los nombres de las clases más claras tienen el mismo formato solo que con más ceros, esto hace que el código sea hasta estético. Mediante `removeClass` y `addClass` se consigue el efecto al cambiar la opacidad en las propiedades de `background-color` en el CSS.

EVENTOS Y FUNCIONES DE MOUSE

A parte de las funciones de `click` que permiten la seudo-navegación al mostrar los mensajes de *'Instructions'*, *'About'* y *'Save'*, también utilizo eventos de mouse en los botones negros y en el marco superior para posibilitar el dibujo y el borrado pulsando con el ratón. Que no se pueda presionar con el ratón dos cajas al mismo tiempo facilita mucho la gestión de estos eventos. Este es un primer acercamiento para poder utilizar los eventos táctiles que permitan el dibujo en dispositivos de este tipo.

funciones de mouse



Utilizamos la misma función comprobar pulsación y los mismos boolenaos para las teclas que hemos creado para los eventos de teclado. Esto nos quita trabajo y permite utilizar el ratón sobre los botones al mismo tiempo que utilizamos el teclado.

Al cliquear sobre un elemento `<div class=".XXXX">` se genera un evento `mousedown` cuando se pulsa y otro `mouseup` cuando se suelta. Estos eventos cambian el booleano correcto que utilizamos para las teclas y crea o borra el intervalo que llama a `comprobarPulsacion()`. Así se continúa dibujando hasta que se deja de pulsar el ratón.

Al mantener pulsado el ratón en una de las cajas destinadas a dibujar y al desplazarlo fuera de esta, el programa entraba en bucle y no dejaba de dibujar hasta que fuera pulsado otro botón. Esto ocurría al no tener lugar el evento `mouseup`. Por ello utilicé el evento `mouseleave` que salta cuando el puntero

abandona un elemento determinado, para así llamar a la función que se utiliza cuando se deja de pulsar esa caja.

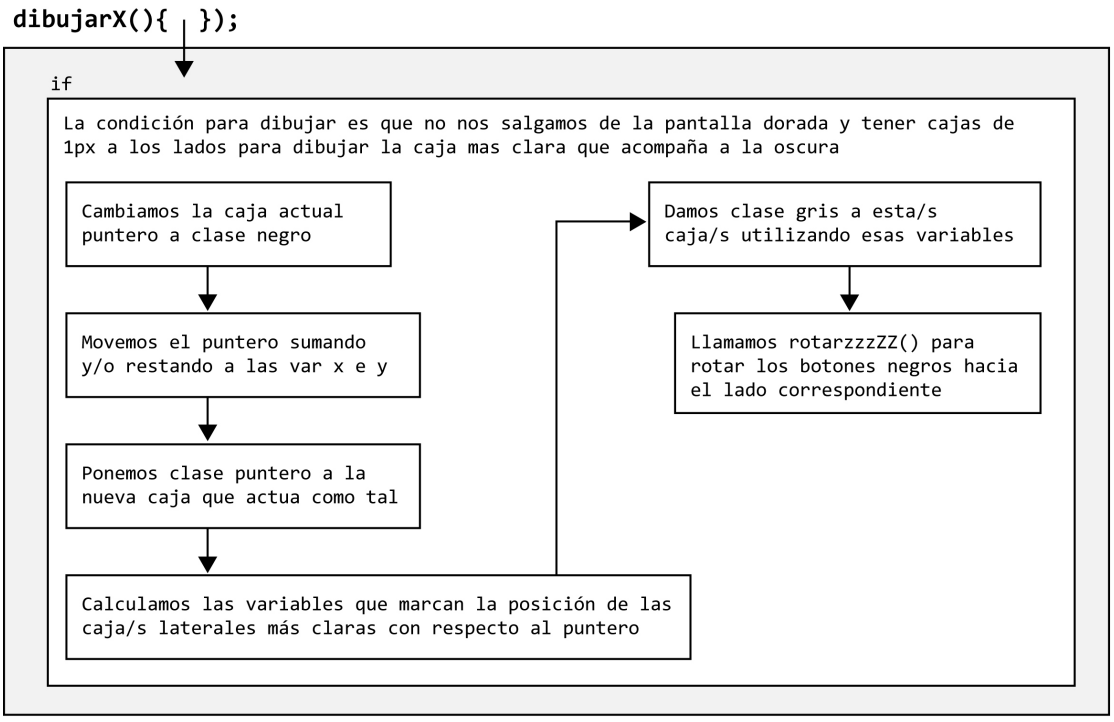
El evento `click` se produce cuando se pulsa y suelta dentro de un determinado elemento. Es suficiente para nuestro efecto de borrar al no necesitar saber cuando se pulsa y cuando se levanta. Cada vez que cliqueas se borra.

Esta es una solución económica en tiempo y código para permitir cierta interacción con el juguete a través del ratón. Y que demuestra que la arquitectura de las funciones de Websketch y su encapsulado permiten y facilitan la incorporación de nuevos elementos y que en el futuro podrá incorporar los eventos táctiles necesarios para los dispositivos tablet y móvil.

COMPROBARPULSACION( )
Esta es la función más importante de todo Websketch, ya que es la encargada de realizar el dibujo. Una vez se pulsan las teclas o se cliquea en los botones ya se sabemos hacía donde quiere dibujar el usuario y se llama en un intervalo a esta función que comprueba hacía donde se quiere dibujar y actúa en consecuencia.

Aquí se declaran una serie de funciones anidadas que hacen rotar los botones, que se llaman en las funciones para dibujar y que se desarrollan más adelante.

Una serie de `else if` comprueban que teclas están pulsadas incluyendo al final uno con la condición de que si ninguna lo está, se borre el intervalo. Dentro de cada `else if` se incluye la función dibujar correspondiente cuya estructura explico en el siguiente punto.

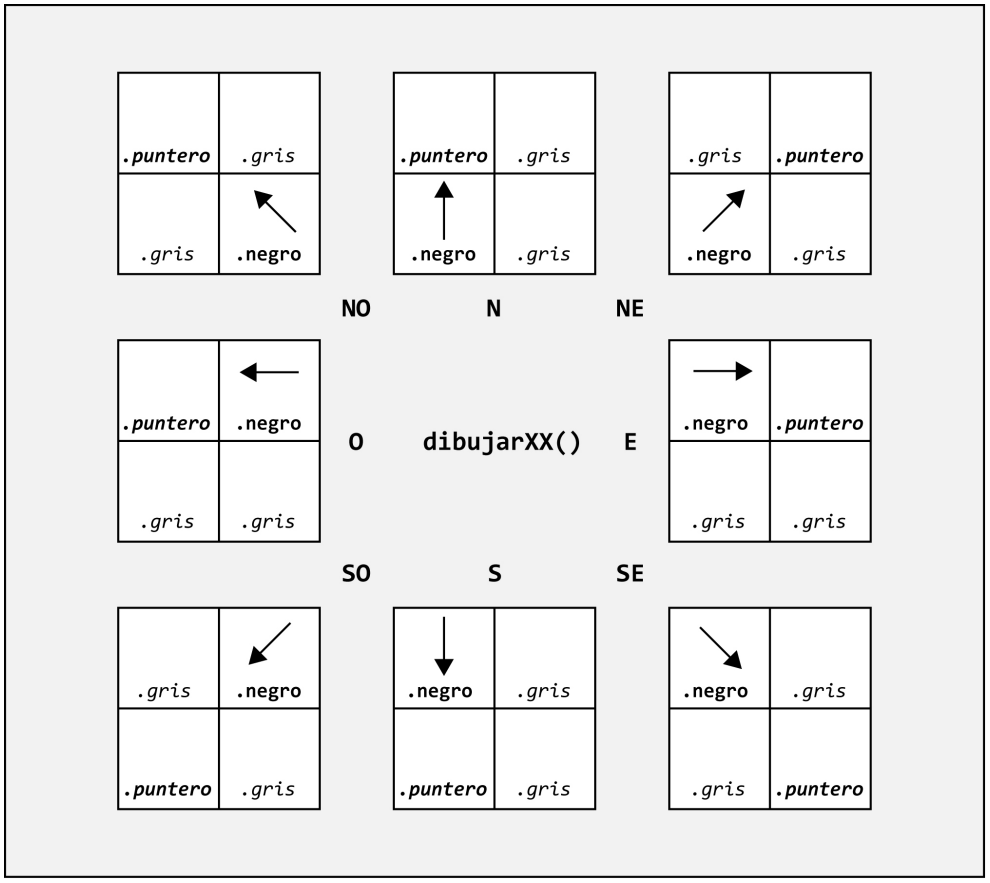


DIBUJARX( )
Al iniciar el desarrollo del Websketch comencé a trabajar en cómo se vería la línea negra que debería estar compuesta por necesidad de cajas de 1px. Trabajaba con clases del CSS que iba cambiando con JS.

Pronto me di cuenta que una línea de una sola caja resultaba demasiado fina y una de 2px o dos cajas demasiado gruesa, sin tener en cuenta el efecto escalera cuando se pintaba en diagonal. La solución de crear cajas de 1,5px no era válida, la pantalla reaccionaba mal a este tamaño ya que está compuesta de píxeles y no era capaz de mostrar ese medio pixel sin crear un efecto de sierra muy poco estético.

La solución pasaba por crear una línea compuesta por una principal de 1px y una secundaria a su lado mas clara. Esto también solucionaba el problema estético de las líneas diagonales, ya que podríamos dibujar cajas claras a los lados de las cajas oscuras evitando el efecto escalera.

En el caso de pisarse alguna clase, prevalece el orden que se les ha asignado en el CSS. De este modo, la clase `.puntero` prevalece sobre `.gris` y `.negro` y ocurre lo mismo con `.gris` que prevalece sobre `.negro` que es más oscuro. Esta decisión fue tomada tras analizar el aspecto de las amalgamas de líneas en el juguete real.



La elección de números a modo de eje de coordenadas para nombrar las filas y columnas se mostró un acierto una vez más al crear las funciones para dibujar. Operaciones aritméticas simples nos daban fácilmente el número que debía ser utilizado para identificar la caja a la que queríamos poner la clase. Concatenando construíamos el selector de la caja que debía recibir la clase.

ROTARZZZXX( )
Estas cuatro funciones (una para cada rueda y cada sentido del giro) vienen a simular el giro de las ruedas negras que controlan el pintado en el juguete real. Si bien son la causa del movimiento lento o entrecortado en alguno de los navegadores, he decidido mantenerlas.

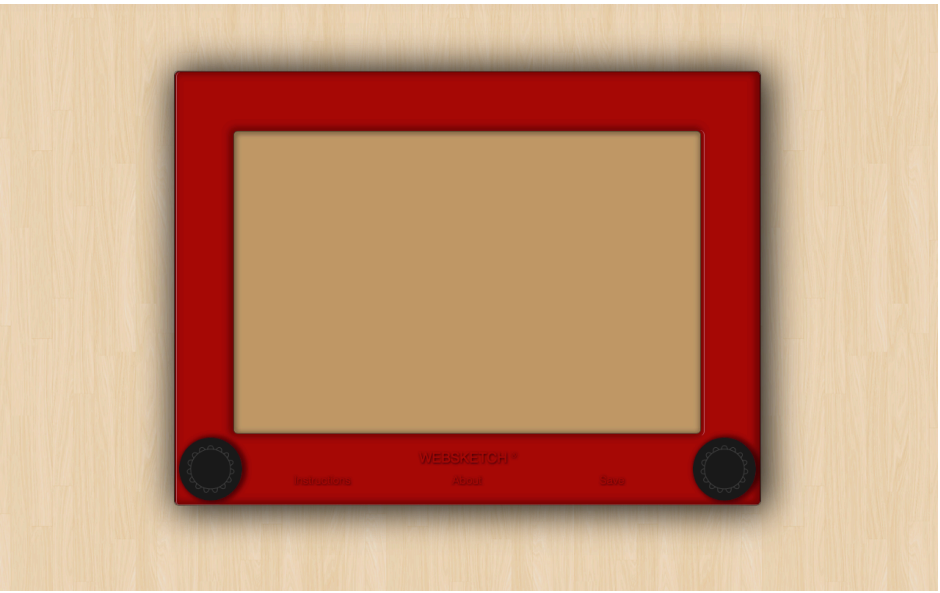
Utilizamos la propiedad `transform: rotate` de CSS para hacer girar las cajas que contienen las imágenes de los botones. Una variable controla el número de grados positivos o negativos que se gira. De este modo el movimiento es continuo.

CONCLUSIONES

Se podría decir que Websketch es como un pequeño Frankenstein al que he dado vida. Los eventos de teclado serían los ojos u oídos que envían la información al cerebro que almacena los datos en booleanos. Sus manos dibujan gracias a la sangre que bombea el corazón que es comprobarPulsación() y que late al ritmo del intervalo.

No creo que sea necesario decir lo complicado que fue encontrar salida a ciertos problemas y conseguir que este pequeño monstruo pudiese caminar y emitiese algunos gruñidos. Siempre que conseguía solucionar un problema surgía otro o me daba cuenta que algo se podía mejorar o comprendía que cierto código podía escribirse de manera más simple y elegante.

Aunque en estas conclusiones también me gustaría hacer referencia a algunos problemas que me siguen torturando y que tal vez solucione en el futuro.



NAVEGADORES

La mayor pesadilla ha sido ver como Websketch funcionaba de manera muy diferente dependiendo del navegador y su versión, la máquina en la que trabajaba y el sistema operativo en el que funcionaba. El trabajo que se le pide al navegador es muy grande y ha habido ciertos efectos, como es el agitado al borrar, que no he podido incluir por esta razón.

Una solución hubiese sido detectar el tipo de navegador y el sistema operativo y quitar ciertas funcionalidades o reducir tamaño para que el Websketch funcionase mejor. Pero finalmente decidí dejar para el futuro este posible desarrollo.

DESARROLLO FUTURO

Sin duda el principal reto es el de crear una versión para dispositivos táctiles. Una tablet es el tipo de dispositivo que más se asemeja al juguete real y por lo tanto Websketch debería funcionar correctamente en los ellos. Esto implica

incluir eventos táctiles que permitan pulsar y girar los botones. He realizado algunas investigaciones en este sentido pero debería profundizar en el tema. Aunque es posible que la respuesta al funcionamiento táctil pase por la creación de una aplicación.

Lo expuesto en el párrafo anterior va ligado a la detección del tamaño del dispositivo y la construcción de manera dinámica de todo el Websketch. Si bien las bases de este desarrollo están marcadas, necesita de investigación y desarrollo que tal vez pasen también por la creación de una aplicación.

Hubiese deseado contar con alguna tutoría con mi profesor y recibir consejo para poder profundizar en el desarrollo de lo expuesto en los dos párrafos anteriores. Pero esto ha sido imposible y ha retrasado en cierto modo la finalización de mi proyecto.

Otro aspecto importante que he dejado aparcado es la posibilidad de poder guardar los dibujos. Aunque no me resulta prioritario, ya que no encaja con el espíritu del juguete original. Pero si veo necesario que podamos compartir nuestros dibujos en las redes sociales. Esto influiría positivamente en el conocimiento de la existencia de Websketch y fomentaría la potencial viralidad de la nostalgia que genera.

VALORACIÓN FINAL

Estoy bastante satisfecho con el proyecto Websketch y espero poder trabajar en los desarrollos futuros a corto plazo. Como he comentado anteriormente, me hubiese gustado poder contar con alguna tutoría de mi profesor para así haber desarrollado algunas partes inconclusas de mi proyecto. Pero en líneas generales estoy contento con el curso que he realizado en la escuela CICE.

Este proyecto me ha servido para descubrir lo mucho que me atrae la lógica de las máquinas y que tal vez pueda lanzar mi carrera en este sentido. He visto posibilidades de realizar gráficos informativos interactivos con los conocimientos adquiridos, actividad relacionada con los medios de comunicación que tanto echo de menos. Aunque también veo la posibilidad de seguir creciendo, aprendiendo otros lenguajes y poder llegar a construir videojuegos.

Recuerdo cuando era niño, con mi ordenador de disquetera de tres y medio, ejecutando los programas desde C y haciendo mis pinitos en Basic. Aun tengo el libro de elige tu propia aventura '*El Expreso de los Vampiros*' con anotaciones de código Basic escritas a lápiz en sus páginas. Conseguí construir el libro en este lenguaje, se podría decir que fue el primer videojuego que hice, una aventura gráfica… se perdió con el viejo Amstrad 086.

¿Quién sabe donde estaría ahora si hubiese seguido ese camino? ¿Quién sabe donde llegaré ahora que lo he retomado?

AGRADECIMIENTOS

Gracias a Cristina, mi madre, sin su apoyo emocional y económico no hubiese podido realizar este curso que ha cambiado por completo mi carrera profesional y mi vida.

Gracias a mi hermana Cristina y mi cuñado Pablo por todo el apoyo que me han dado.

Gracias a David, mi amigo y compañero de piso, por aguantarme y conseguirme el sonido de la maraca.

Gracias a mis compañeros de clase en CICE, en especial a Andrés que me leía el horóscopo todas las mañanas.

Gracias a Brian de www.spotahome.com por echarle un vistazo a mi código y darme valiosos consejos.