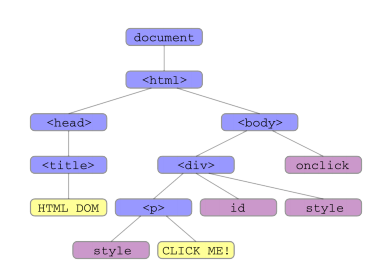
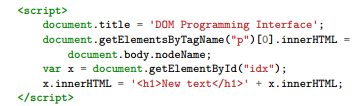
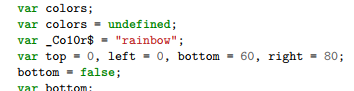
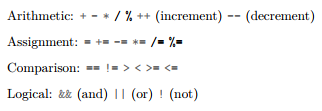
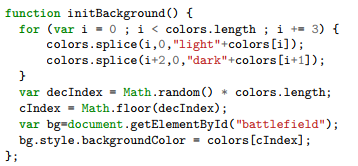
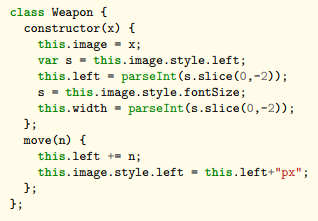
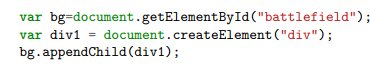
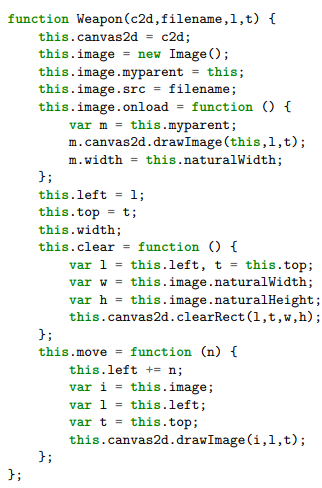
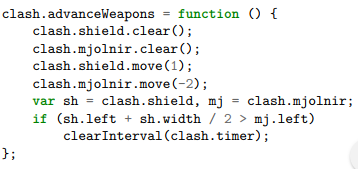
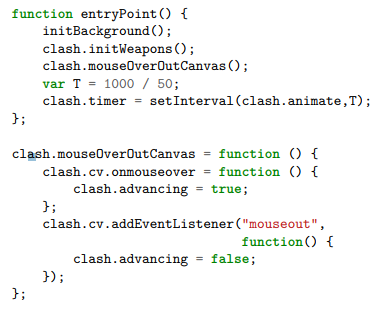
***Por Aythami Godoy Rodríguez*.**

Gracias al ***Document Object Model*** (DOM) los programadores y desarrolladores web son capaces de acceder, cambiar y eliminar cualquier elemento de un documento *HTML*. Esto puede ser logrado mediante desarrollo de programas en el languaje *Javascript* (JS).

*Javascript* es un lenguaje de programación de alto nivel, dinámico y desvinculado. Éste ha sido estandarizado en las especificaciones del lenguaje *ECMAScript*, siendo concretado del todo con otros lenguajes como *ActionScript* (AS). JS puede acceder a etiquetas de *HTML*, así como a propiedades y contenidos a través de *DOM* y su manejo dinámico del *runtime*.

En esta práctica nos encargaremos de introducirnos en algunos conceptos básicos tanto de *JS* como de *DOM*, combinando ambos en una serie de proyectos de dificultad sencilla.

1. **Qué es *HTML DOM*.**El *HTML DOM* es un estándar de modelo de objeto e interfaz de programación para *HTML*. Se trata de un árbol de objetos creado por el navegador cuando se carga una página web, dando pie a que se permita a *JavaScript* (u otros lenguajes de programación) realizar múltiples cosas.  
   **1.1 árbol de objetos de DOM.**Para seguir el fragmento de código de la figura 1 y mostrarlo de forma conceptual se hace de un esquema de *árbol de objetos*.  
     
   Su equivalente en árbol sería:  
     
     
     
     
   **1.2 Interfaz de programación de *DOM*.**La interfaz de programación, que sirve de envoltorio para nuestro modelo (el árbol de objetos), es proveída por la *DOM* para acceder al total de los elementos almacenados en dicho árbol, teniendo en cuenta su diferente naturaleza: elementos de HTML/CSS, propiedades (características de los objetos), métodos (acciones ejecutables sobre el objeto), eventos (sucesos o cambios que afectan al objeto),...  
     
   Cualquier elemento del documento HTML puede ser accedido a través de un *document object* predefinido en *JavaScript*, que ofrece métodos para encontrar elementos concretos en el *DOM*, como ***getElementById()*** or ***getElementsByTagName()***, o propiedades para directamente obtener elementos seleccionados, como ***body*** o ***title***.  
     
   La propiedad *title* devuelve el título en formato cadena del documento, mientras que *body* es una propiedad representa la sección *<body>* de cualquier documento de formato HTML. Cada elemento HTML representado en DOM tiene una propiedad *nodeName* que contiene el *tagname* en *capitals*. El método ***getElementsById()***   
   devuelve el objeto identificado por sus argumentos. Por otro lado, ***getElementsByTagName()*** devuelve una colección (como un array) compuesto por todos los objetos que contengan el argumento *tagname*. Las dos búsquedas invocan a los respectivos elementos involucrados. Los errores de sintaxis de *JavaScript* son considerado menores, pero las frases de este ejemplo pueden ser ilegibles.  
     
   **1.3 Conectando DOM con CSS.**DOM también permite accesos y cambios de estilo de elementos de HTML. Por ejemplo, en el caso de las propiedades CSS de cualquier elemento perteneciente a HTML.  
   Cualquier elemento de HTML proporciona una propiedad de estilo, que mantiene un objeto que tiene propiedades DOM para todas las posibles propiedades de estilo de CSS. Esas propiedades DOM almacenan valores de *string*, que podemos enviar con tal de modificar cualquier aspecto del estilo de los elementos.  
     
   Algunos nombres de propiedades de estilos de CSS contienen una raya, como por ejemplo *background-color*. Como los nombres son difíciles de trabajar en *JS*, éstos deberían se escritos como *style[“background-color”]*. De manera alternativa, nosotros usamos nombres de propiedades *DOM* teniendo sus rayas quitadas y su nombre en *camel*. Como por ejemplo en *style.backgroundColor*).
2. **Conceptos básicos de JavaScript.**En HTML, el código de JavaScript debe ser insertado entre *<script>* y *</script>*. Cualquier número de *scripts* puede ser reemplazado en un documento HTML, o bien en el *body* o en la sección de *head* (o incluso en los dos). Sin embargo, el reemplazo en la sección de *body* es mucho más recomendado.  
   Además, para facilitar la reusabilidad, se puede disponer de un fichero externo que almacene los *scripts*, siendo llamado mediante la siguiente línea:  
     
   ***<script src="js/shieldhammer.js"></script>*2.1 Valores, tipos y operadores.**Hay 6 tipos básicos de valores en *JavaScript*: number, string, Boolean, undefined, function y object, siendo categorizadas las primeras 4 como **simples** y las otras 2 como **complejas**. Las simples siguen unas reglas comunes y de sentido común: los números pueden ser enteros o decimales, los strings son textos delimitados por doble comilla, los booleanos disponen de dos valores (true y false) y undefined solamente se tiene a sí mismo.  
   Las variables se constituyen escribiendo *var*. Al mismo tiempo que se declara la variable, como es lógico también se deberá rellenar con el =. Evidentemente, que esté declarada y que tenga valor no es lo mismo.  
     
     
   A la hora de operar se utilizan los siguientes caracteres:  
     
     
   **2.2 Objetos y arrays.**Un objeto literal (en JavaScript) permite definir y crear un objeto solamente declarado. Es una lista de propiedades escritas como *name: value*, separados por comas y delimitados por *curly braces*. Un objeto vacío está especificado por el {}:  
     
   ***var clash = {shield:undefined, mjolnir:undefined}; clash.timer = undefined;***La propiedad de un objeto puede ser accedida como *clash.shield*. Nuestro proyecto de ejemplo usa el objeto principal *clash* para representaer y administrar la propia escena que va a ser animada.  
   Los elementos sobrantes son los *métodos*, que realizan acciones sobre los objetos y que realmente son propiedades almacenadas en funciones definidas. Ellas, como propiedades, pueden ser añadidas a objetos existentes después de su creación.  
   Para invocar un método se coloca *objeto.método(parámetros)*.  
   Por otra parte, una array se crea con la estructura *tipo nombre\_variable=[elemento1,elemento2]*El tamaño que tiene dicho array se comprueba con el método *.length;*  
     
   El objeto *window* está soportado por todos los navegadores y representa la pantalla del mismo. Entonces, todas las variables, objetos y funciones de *Javascript* pasan a ser automáticamente miembros, propiedades y métodos del objeto *window* (BOM), preferible con respecto al DOM. Esperar al BOM es más seguro que esperar al DOM para esperar a correr código *JS*, pues éste requiere más contenido (imagen,css, scripts, etc.) para ser cargado y preparado.  
   Entonces,en todos nuestros ejemplos de programas de HTML/CSS/JS en el *file* principal siempre vamos a colocar la siguiente línea:  
    *window.onload = entryPoint;*Entonces, la función llamada *entryPoint* debe ser definida, conteniendo el inicio de las acciones de nuestro programa. Por otra parte, los programas de JS usan punto y coma para separar declaraciones, aunque es opcional en muchos casos. Para evitar pensar acerca de los casos que lo requieren o no, acabar cada declaración con punto y coma está recomendado.
3. **JavaScript: control de declaraciones.**Como era de esperar, JS tiene sentencias que controlan el flujo de la ejecución. Puedes encontrar que son las típicas estructuras condicionales y de bucle similares a las de otros lenguajes.  
     
   3.1 Ejecución de condicionales.  
   Estaríamos hablando del *if*. Su funcionamiento es exactamente el mismo que hemos visto a lo largo de la carrera, así que poco se puede añadir.  
   También tenemos el *switch*, que se trat  
     
   switch ( /\*expression\*/ ) {   
   case /\*expression 1\*/ : // code block for expression 1   
    break;   
   case /\*expression 2\*/ :  
    // code block for expression 2  
    break; ...   
   default:   
    // code block default value  
   }   
     
   3.2 Bucles.  
   Están los habituales: *for, while, do-while,...*  
   3.3 Funciones.  
   Las funciones se declaran mediante la siguiente estructura:  
     
   Cuando la función sea invocada con los parámetros adecuados, el código de la misma será ejecutada.   
     
   3.4 Clases.  
     
   Básicamente funciona como en C#, con la diferencia de que se utiliza el *this* con tal de referirse a un hipotético objeto de la clase.
4. **OOP con JavaScript.**  
   JavaScript provee a las clases de una función estándar denominada *prototype*.
5. **Creando nodos en el DOM.**HTML DOM puede ser completamente editado mediante cambios, adiciones o eliminaciones de cualquier tipo de nodo. No todos los elementos pero sí atributos o contenido de elementos. Cada nodo editado ha de ser gestionado a través de su nodo padre en orden de preservar la integridad del árbol DOM.  
   Para añadir un nuevo elemento, el elemento nodo primero debe ser creado y entonces ha de ser añadido al elemento padre seleccionado. La creación de un elemento está ligado a *createElement*, método del objeto *document*, mientras que el método *appendChild* añade un elemento hijo a otro padre. La estructura que sigue es algo tal que así:  
     
   Añadir contenido de texto en un elemento nodo debe seguir los siguientes pasos:  
   
6. **Animando en el *canvas*.**Primeramente se coloca este código con tal de declarar el *canvas*:  
     
   Luego usamos el método *initWeapons* en el *main* de la clase *clash*:  
     
   Entonces, el constructor weapon crea dos objetos: clash.shield y clash.mjolnir.  
   Después pasa esto:  
     
     
   Las llamadas se producen así:  
   
7. **Eventos y su mantenimiento.**Un evento, en palabras simples, es algo que ocurre. Para programar eventos, se utliliza el formato de programación *event-driven programming*. Para ello hay qué saber qué clase de evento se va a realizar y cuál va a ser la respuesta del programa.  
   HTML DOM y BOM en navegadores te ofrecen herramientas para satisfacer los dos requerimientos. Cada elemento tiene atrbituos *onload, onclick, onmouseover, onkeypress*.  
    **window.onload = entryPoint;***addEventListener* es un método que permite añadir cualquier número de *handlers* por nodo. Esta disponible para cualquier elemento.  
   *removeEventListener* elimina el *handler* del elemento.  
   

**---------------------------------------------------------------------------------------------------------------------------**

**Va, ahora los apuntes de verdad.**

Para randomizar un array:***Math.ceil((Math.random()\*array.Length)-1);***

Este es el código del ejercicio 3 (el del DOM). Guárdalo como oro en paño porque te va a ser útil de la polla:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

var colors = ["cyan","green","gray","blue"];

var cIndex = 0;

function randomi(){

var longitud=colors.length;

return Math.ceil((Math.random()\*longitud)-1);

}

function initBackground() {

for (var i = 0 ; i < colors.length ; i += 3) {

colors.splice(i,0,"light"+colors[i]);

colors.splice(i+2,0,"dark"+colors[i+1]);

} // colors === ["lightcyan","cyan","darkcyan","lightgreen","green","darkgreen"...

cIndex = randomi();

var bg=document.getElementById("battlefield");

bg.style.backgroundColor = colors[cIndex];

};

function Weapon(x) {

this.image = x;

var s = this.image.style;

this.left = parseInt(s.left.slice(0,-2));

this.width = parseInt(s.fontSize.slice(0,-2));

this.move = function (n) {

this.left += n;

this.image.style.left = this.left + "px";

};

};

// Entry point

window.onload = entryPoint;

function entryPoint() {

initMyDOM();

initBackground();

clash.initWeapons();

var T = 1000 / 50;

clash.timer = setInterval(clash.animate,T);

};

function initMyDOM(){

var bg=document.getElementById("battlefield");

var div1=document.createElement("div");

bg.appendChild(div1);

var letraO=document.createTextNode("O");

div1.appendChild(letraO);

var att=document.createAttribute("id");

att.value="capshield";

div1.setAttributeNode(att);

document.getElementById("capshield").style.font="italic bold 60px arial,serif";

document.getElementById("capshield").classList.add("superheroweapon");

document.getElementById("capshield").style.left="60px";

document.getElementById("capshield").style.top="270px";

var div2=document.createElement("div");

bg.appendChild(div2);

var letraT=document.createTextNode("T");

div2.appendChild(letraT);

var att=document.createAttribute("id");

att.value="mjolnir";

div2.setAttributeNode(att);

document.getElementById("mjolnir").style.font="italic bold 60px arial,serif";

document.getElementById("mjolnir").classList.add("superheroweapon");

document.getElementById("mjolnir").style.left="760px";

document.getElementById("mjolnir").style.top="280px";

document.getElementById("mjolnir").style.transform="rotate(270deg)";

}

var clash = {shield:undefined, mjolnir:undefined};

clash.timer = undefined;

clash.initWeapons = function () {

var sh = document.getElementById("capshield");

clash.shield = new Weapon(sh);

var mj = document.getElementById("mjolnir");

clash.mjolnir = new Weapon(mj);

};

clash.animate = function () {

clash.advanceWeapons();

clash.changeBackgroungColor();

};

clash.advanceWeapons = function () {

clash.shield.move(1);

clash.mjolnir.move(-2);

var sh = clash.shield, mj = clash.mjolnir;

if (sh.left + sh.width / 2 > mj.left)

clearInterval(clash.timer);

};

clash.changeBackgroungColor = function () {

var period;

var sh = clash.shield, mj = clash.mjolnir;

if (sh.left + 2 \* sh.width > mj.left)

period = 1;

else

period = 20;

if (sh.left % period === period - 1) {

cIndex = (cIndex + 7) % colors.length;

var bg = document.getElementById("battlefield");

bg.style.backgroundColor = colors[cIndex];

}

};

