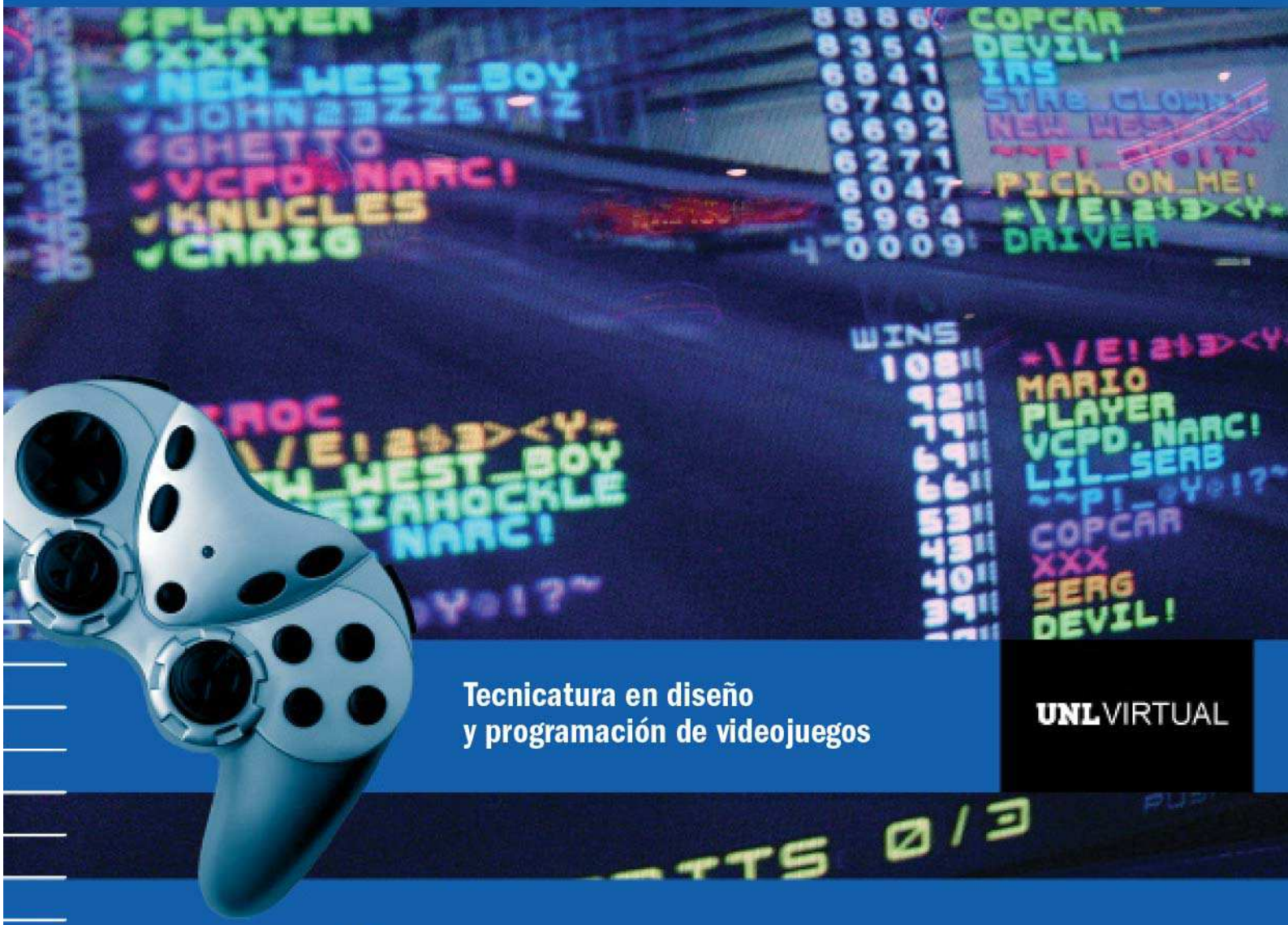
UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS



Tecnicatura en diseño
y programación de videojuegos

UNL VIRTUAL

Programación para videojuegos III

Desarrollo de videojuegos web

Unidad 1
Introducción a html y java script

Docentes
Alejandro Fort Villa
Walter Sotil

CONTENIDOS

1. Introducción a HTML y JavaScript

A continuación veremos algunos conceptos de manera muy breve, que nos servirán para entender algunos elementos claves de la programación de videojuegos web.

El mundo web es bastante complejo porque está lleno de conceptos, protocolos, mitologías, lenguajes, etc. Sin embargo, nos enfocaremos sólo en aquellos que se necesitan para poder desarrollar juegos. Primero, estudiaremos el concepto de web, por qué trabajaremos con ciertas tecnologías y de qué manera lo haremos. Luego, veremos los lenguajes que vamos a utilizar.

Antes de adentrarnos estrictamente en los lenguajes, debemos comprender –a menos superficialmente– qué es el modelo cliente-servidor.

Modelo cliente-servidor

La web se maneja por medio de un modelo de aplicación distribuida, llamada *modelo cliente-servidor.*

Este modelo es básicamente una forma de distribuir el trabajo entre los proveedores de recursos y servicios y los clientes demandantes.

Modelo cliente-servidor
Se utiliza en web y constituye un modo de distribuir el trabajo entre los proveedores de recursos y servicios y los clientes demandantes.

Funciona de la siguiente manera: el cliente realiza una petición y el servidor responde; luego, el cliente ejecuta la petición. Por ejemplo, cuando el cliente pide una página web al servidor, éste ve los archivos que tiene que mandar y se los envía al cliente, quien luego los muestra en pantalla. Un dato interesante para destacar es que mientras no hay comunicación entre el cliente y el servidor, la conexión permanece inactiva y sólo se reanuda cuando se trasfieren los recursos; después de esto, se vuelven a desconectar.

Es importante entender el modelo cliente-servidor para comprender a qué se dedica y limita cada parte. En nuestro desarrollo de juegos web trabajaremos sobre el cliente, pero no nos desentenderemos de lo que ocurra en la parte del servidor.

¿Por qué en el lado del cliente?

La mayoría de las aplicaciones (juegos, sobre todo) trabajan con la parte del cliente porque tienen continua interacción con el usuario. Imaginemos los eventos del mouse. Si el evento estuviera en manos del servidor, cada vez que hacemos click sobre la pantalla, el cliente tendría que mandarle la respuesta al servidor (previa conexión), el cual tendría que decidir qué hacer con el evento y responderle al cliente para que ejecute las acciones necesarias. Además, estaríamos estableciendo continuamente comunicación con el servidor, ralentizando nuestro juego.

Por lo tanto, ¿qué existe en el lado servidor?

El lado servidor se dedica, por lo general, a administrar recursos y archivos persistentes, como las bases de datos de los puntajes y jugadores. Sin embargo, dejaremos esto para otra oportunidad. Nos concentraremos pura y exclusivamente en desarrollar el juego. Sabiendo esta diferencia, podemos concentrarnos en aprender las tecnologías que vamos a usar.

HTML

HTML
Significa HyperText Markup Language y es el lenguaje mayormente utilizado para la realización de páginas web. Se estructura mediante tags, que tienen una sintaxis muy simple: un tag de apertura contenido entre <...> y uno de clausura contenido entre </...>.

HTML es la sigla de HyperText Markup Language y constituye el *lenguaje* mayormente utilizado para la realización de páginas web.

Un gran número de desarrolladores no lo considera un lenguaje, sino una forma de estructurar los datos contenidos en un documento web.

HTML se estructura mediante *tags,* que son las etiquetas utilizadas para demarcar los elementos de una página web. Estos tags tienen una sintaxis muy simple: un tag de apertura contenido entre <...> y uno de clausura contenido entre </...>.

Existen muchísimos tags. Algunos de los más utilizados son:

<DIV> </DIV> Demarcan una región o división en la página.

<A> </A> Definen un link (ancla).

<IMG/> Define una imagen (sólo es un tag de apertura).

 Inserta un salto de línea (sólo es un tag de apertura).

<P> </P> Definen un párrafo.

<CANVAS></CANVAS> Se utilizan para diseñar aplicaciones interactivas.

Nosotros vamos a utilizar sólo el último (CANVAS), que es un elemento HTML que fue incorporado en HTML en su versión 5 (también llamado HTML5) que permite la generación de gráficos dinámicamente por medio del scripting, que generalmente es JavaScript.

Documento web
Está compuesto por una *cabecera*, que contiene todos los metadatos necesarios para enriquecer y estructurar la página, y un *cuerpo*, donde se encuentran todos los elementos del documento.

Un *documento web* se compone de dos partes estructurales bien definidas: la cabecera y el cuerpo.

En la *cabecera* se colocan todos los metadatos necesarios para enriquecer y estructurar la página. Por ejemplo, los tags de búsqueda para los navegadores, la codificación del texto, el título del documento, la información del servidor o de la base de datos, etc. Como no nos interesa esto, nuestra cabecera quedará vacía.

El *cuerpo* contiene todos los elementos del documento: formularios, tablas, párrafos, imágenes, links, etc. Aquí se va a situar nuestro juego.

```
<HTML>
    <HEAD>
    </HEAD>

    <BODY>

        <CANVAS>
        </CANVAS>

    </BODY>

</HTML>
```

Figura 1. Script 1.

En este ejemplo de documento web tenemos el documento HTML con la cabecera *(HEAD)* vacía y el cuerpo *(BODY)* que sólo contiene el elemento *CANVAS.*

Cabe mencionar que HTML no diferencia entre mayúsculas y minúsculas. Sin embargo, los elementos suelen escribirse con mayúscula para diferenciarse del contenido estricto de la página. HTML tampoco distingue tabulaciones ni saltos de líneas. Esto significa que podemos escribir todo en un mismo renglón mientras respetemos la correcta apertura y clausura de los tags.

Los tags poseen atributos que definirán el formato del elemento, como el ancho, el alto, el color, la fuente, etc. Algunos atributos son particulares del tag. Así, por ejemplo, el tag *IMG* tiene un atributo *src* que define la ruta de la imagen a insertar. Otros atributos son genéricos, como el *ID*. Éste es utilizado, entre otras cosas, para hacer referencia al objeto. Es decir, cuando trabajemos con el código JavaScript y queramos hacer referencia a nuestro CANVAS, la forma de capturar este objeto es definiéndole un *ID* y, en el script, capturar el objeto que tiene ese ID (más adelante detallaremos esto).

Para definir un atributo, debemos escribirlo dentro del tag de apertura y escribirlo con un string entre comillas dobles o simples (tampoco hay diferenciación al respecto en HTML).

Por ejemplo, si queremos darle el ID *"mi_canvas"* a nuestro elemento, lo hacemos así:

<CANVAS id = "mi_canvas"></CANVAS>

De esta manera, el documento web completo que trabajaremos queda de la siguiente manera:

```
<HTML>
    <HEAD>
    </HEAD>

    <BODY>

        <CANVAS id="mi_canvas"></CANVAS>

    </BODY>

</HTML>
```

Figura 2. Script 2.

Si copiamos este texto y lo guardamos bajo la extensión *.html,* podremos abrirlo con cualquier navegador.

JavaScript

Java Script
Es un lenguaje de programación interpretado, por lo que no requiere un compilador. Está basado en prototipos y es extensible a objetos. Se utiliza comúnmente en el lado *cliente* de la programación web.

Es un lenguaje de programación interpretado, por lo que no requiere un compilador. Está basado en prototipos y es extensible a objetos. Se utiliza comúnmente en el lado *cliente* de la programación web.

Salvo muy leves diferencias, la sintaxis de JavaScript es muy similar a la de su pariente Java y su antecesor C. JavaScript es un lenguaje estándar, aunque algunos navegadores poseen instrucciones especiales.

Para insertar código JavaScript en nuestra página, debemos hacerlo dentro de los tags SCRIPT en el archivo .html. El script es ajeno a la interpretación del código HTML, por lo que es indiferente dónde ubiquemos los tags. Por una cuestión de comodidad, la declaración de librerías y variables suele colocarse dentro del HEAD y, al final del documento, el resto del código. Lo veremos con la práctica.

Algunas características de JavaScript:

- Las variables no se definen con tipo específico.
- Una misma variable puede contener diferentes tipos de datos a lo largo del script.
- Todas las variables se establecen por referencia.
- Es sensible a mayúsculas y minúsculas.

En el siguiente código se pueden observar algunos ejemplos de declaraciones de variables y arreglos:

```
<HTML>
    <HEAD>
    </HEAD>

    <BODY>
        <CANVAS id="mi canvas"></CANVAS>

    </BODY>

<SCRIPT>


// comentario de simple linea

/*
 Comentarios
 de multiples
 lineas

 */

 // el ; es util para que sea legible el codigo pero no es necesario

 x = 5;
 y = 5

/* Todas las variables son Strings que puede parsearse con algun tipo de
dato*/

A = 10;

// A vale 10

A = A + 1;

// A vale 11


 A = A + ""; // el + esta sobrecargado para la concatenacion de strings

 // a A le sumamos un caracter vacio, ahora A es un string que tiene la
cadena "11"

A = A + 1;

 /* Al String A le sumamos un 1, ahora A es un String que vale "111"

  /*
 Si declaramos una variable con la palabra reservada var, la misma sera
de ambito
 local.

 */

 var t;


 /*** Los Arreglos ***/

 // Todos los Strings son arreglos de caracteres

 Arr1 = "Carlos";

 // Arr1[0] = "C";
 // Arr1[1] = "a";
 // Arr1[2] = "r";
 // Arr1[3] = "l";
 // Arr1[4] = "o";
 // Arr1[5] = "s";

 /* TODAS ESTAS SON FORMAS VALIDAS DE DECLARAR UN ARREGLO */
```

```
 /* Creando un arreglo vacio y definiendolo manualmente */

    var Autos=new Array();
        Autos[0]="Saab";
        Autos[1]="Volvo";
        Autos[2]="BMW";

/* Instanciandolo en el mismo constructor del arreglo */

    var Autos=new Array("Saab","Volvo","BMW");

/* Creando el arreglo de forma literal*/

    var Autos=["Saab","Volvo","BMW"];



 /* En un arreglo se puede predefinir la cantidad de datos*/

 var arreglo0 = new Array(10);

 /* Sin embargo, son dinamicos, podemos referenciar la posicion 11 */

    arreglo0[10]=1;

/* La funcion "alert()" dispara una ventana de dialogo, es muy util,
incluso
muchas veces utilizada como depurador */


 alert("Hola Mundo");

 x = 1;

 alert (x);

 alert(Autos[2]);


</SCRIPT>
</HTML>
```

Figura 3. Script 3.

A continuación se presentan algunas estructuras más utilizadas:

```
        // ciclo FOR

        for(i=0;i<10;i++){

            alert(i); // muestro i

        }

        // ciclo WHILE

        i=0;

        while(i<10){

            alert(i); // muestro i

            i++;

        }


        // selector switch
```

```
i = 3;

switch (i){

    case (0) : alert("vale 0"); break;
    case (1) : alert("vale 1"); break;
    case (2) : alert("vale 2"); break;
    case (3) : alert("vale 3"); break;

}

// Condicional IF

i = 3;

if (i<3){
    alert("vale menos de 3");
}
else {
    alert("vale mas de 3 o igual");
}
```

Figura 4. Script 4.

Ahora veremos cómo declarar funciones y objetos.

Para declarar una función, debemos utilizar la palabra reservada *function* y respetar la siguiente sintaxis:

function nombre_function (parámetro_1, parámetro2, ...){

...

}

Lo que sigue es un ejemplo de cómo crear una función para sumar dos valores:

```
function suma(a,b){

    return (a+b);
    // si la funcion no tiene un return es void
}


alert(suma(3,5));
```

Figura 5. Script 5.

La palabra reservada *function* no se utiliza sólo para declarar funciones, sino también para declarar objetos y sus métodos.

Si bien en esta asignatura no utilizaremos objetos, tendremos que modificar algunos ya creados, por lo que será muy útil saber cómo luce un objeto y sus métodos.

Para crear un objeto, debemos generar una variable y asignarle una función.

Para declarar un atributo, tenemos que crear una variable dentro del objeto; para declarar un método privado, debemos declarar una función dentro del objeto, y para declarar un método público tenemos que declarar una variable y asignarle una función (utilizando la palabra reservada *this*). El ejemplo que se observa a continuación ayudará a aclarar este tema:

```
/* Objeto "mamifero" con dos parametros en su constructor:
patas: el numero de patas del mamifero
color: el color del mamifero
```

```
*/

var mamifero = function (patas, color){

// atributo privado (declarado con var)
var orejas = 2;
var ColorMamifero = color;

/* Asi se declara un metodo privado */

    function mamar(){
        alert("me estoy alimentando en "+patas+"patas");
    }

/* Asi se declara un metodo publico (con el apuntador this) */

    this.amamantar = function(){
        // puedo acceder a los elementos privados
        mamar();
    }

};

/* Instancio el objeto */
var gato = new mamifero(4,"marron");

// llamo al metodo publico
gato.amamantar();
```

Figura 6. Script 6.

Si bien no vamos a declarar objetos, utilizaremos objetos creados. Es muy común en JavaScript redeclarar los métodos de los objetos creados para que realicen algún algoritmo deseado.

A esta metodología la usaremos con mucha frecuencia, ya que vamos a crear nuestros propios elementos en el juego, explicitándoles el comportamiento que deseamos que realicen.

Para redefinir un método de un objeto, debemos instanciar el objeto y asociarle un nuevo comportamiento al método, definiéndole una nueva función.

En el siguiente script modificaremos el objeto creado en el script 6.

```
/* Instancio el objeto */
var gato = new mamifero();

// llamo al metodo publico
gato.amamantar();

/* Redeclaro el metodo amamantar */

gato.amamantar = function(comida){
    alert("ahora creci y como "+comida);
}
gato.amamantar();
```

Figura 7. Script 7.

Inclusiones

Al igual que en otros lenguajes, en JavaScript a veces resulta muy incómodo tener todo el código en el mismo archivo, por lo que conviene generar varios archivos e ir incluyéndolos de acuerdo a nuestra necesidad.

Supongamos que tenemos el script 3 en un archivo aparte, llamado *S3.js* (js es la extensión JavaScript), y el script 4 en otro archivo, llamado *S4.js*. A su vez, en una misma carpeta, tenemos el archivo de ejemplo.html (script 2), el S3.js y el S4.js.

Podemos, entonces, hacer una inclusión de ambos JavaScript de la siguiente manera:

En ejemplo.html:

```
<HTML>
    <HEAD>
    </HEAD>

    <BODY>
        <CANVAS id="mi_canvas"></CANVAS>

    </BODY>

    <SCRIPT type="text/javascript" src="S3.js"></SCRIPT>
    <SCRIPT type="text/javascript" src="S4.js"></SCRIPT>

 </HTML>
```

Figura 8. Script 8.

Ejemplo completo

Antes de mostrar un ejemplo integrador para comprender los conceptos que hemos visto, nos explayaremos en algunas cuestiones que quedaron pendientes.

Anteriormente dijimos que en JavaScript podíamos referenciar los objetos HMTL y modificarlos, pero no explicamos cómo. Mencionamos los ID, pero no explicamos cómo se usan.

A continuación, analizaremos estos temas.

El *ID* es un atributo de los tags HTML. Su valor es un string y no es único, lo cual significa que varios elementos HTML pueden tener el mismo ID. Esto es ventajoso para aplicarle a varios elementos la misma función o estilo con una sola asignación.

Para referenciar un elemento HTML en JavaScript, debemos dirigirnos al objeto *document* (en minúscula). *document* es un objeto del documento que tiene la referencia a todo lo que contiene la página; sólo debemos saber cómo buscar. Si quisiéramos, podríamos hacer una analogía en C++ al objeto *System*.

document tiene infinidades de métodos que nos permiten desplazarnos por los elementos de un documento, incluso buscar aquellos elementos anónimos que no tienen definido un ID. Sin embargo, por una cuestión de practicidad, nos basaremos en un ID definido.

Utilizaremos un método de *document* llamado *getElementById* (obtener el elemento a partir de su ID). Esto nos devolverá la referencia al objeto cuyo ID pasemos por parámetro. En caso de que hubiera muchos elementos con ese ID, nos devolverá la referencia del primero que encuentre (de arriba para abajo en el documento).

La sintaxis es:

var varName = document.getElementById(ID);

En el ejemplo que estamos usando, sería:

```
var Mi_objeto_canvas = document.getElementById("mi_canvas");
```

Así nos queda en la variable *Mi_objeto_canvas* una referencia al elemento CANVAS del HTML.

Por último, explicaremos a grandes rasgos cómo dibujar un rectángulo en el objeto canvas, ya que este tema abordará con mayor detalle en las unidades correspondientes.

El elemento canvas tiene un objeto interno llamado *context 2D;* aquí es donde se dibuja. En el siguiente ejemplo haremos un cuadrado azul de 100 x 100 píxeles:

```
// obtenemos la referencia

var Mi_objeto_canvas = document.getElementById("mi_canvas");

// obtenemos el context

ctx = Mi_objeto_canvas.getContext('2d');

//definimos un color para el llenado

ctx.fillStyle   = '#00f'; // azul (en hexadecimal es equivalente a decir
000 000 255)

//dibujamos un cuadrado desde 0 en x, 10 en y, con un ancho de 100 y un
alto de 100

// Dibujamos el cuadrado.

ctx.fillRect  (0,   10, 100, 100);
```

Figura 9. Script 9.

Guardamos el script 9 en un archivo llamado *cuadrado.js.* Luego, en ejemplo.html incluimos el archivo cuadrado js y obtenemos lo siguiente:

```
<HTML>
    <HEAD>
    </HEAD>

    <BODY>
        <CANVAS id="mi_canvas"></CANVAS>

    </BODY>

    <SCRIPT type="text/javascript" src="cuadrado.js"></SCRIPT>

 </HTML>
```

Figura 10. Script 10.

Si abrimos ejemplo.html desde un navegador, veremos un cuadrado azul en pantalla.

Bibliografía

Goodman Danny. *Javascript Bible: Gold Edition*. John Wiley & Sons, julio de 2001. ISBN-10: 0764547186 | ISBN-13: 978-0764547188.