Introducción.

Este es el primer Lab para la creación de nuestro videojuego HTML 5.

Para la creación del videojuego utilizaremos HTML 5 para definir el contenido visual, CSS para definir la apariencia de los objetos y JavaScript para la lógica del juego y las animaciones.

En este primer Lab vamos a crear la estructura básica de archivos y contenido del documento HTML y mediante JavaScript cargaremos y mostraremos las imágenes.

Es MUY IMPORTANTE que no copiéis y peguéis los fragmentos de código que tenéis en este documento, ya que si lo escribís vosotros mismos lo entenderéis mucho mejor, no os llevará mucho más tiempo y os surgirán dudas que os ayudarán a vosotros y a vuestros compañeros.

Paso previo: Archivos

En la carpeta Letters que acompaña a este Lab podéis encontrar los archivos básicos con los que comenzaremos el desarrollo del videojuego.

Analicemos estos archivos:

- letters.html: Es el documento HTML donde se construirá el juego. Por el momento está prácticamente vacío.
    - Define el título y charset.
    - Incluye los 3 archivos JavaScript necesarios por el momento
    - Define una variable global game
    - Define una función que se invocará cuando toda la estructura esté creada.
    - Define una vacía con un id concreto, que indica que esa div será el tablero del juego.
- images: Es la carpeta que contiene las imágenes necesarias. Dentro están las imágenes de las 5 vocales que, además, tienen diferentes tamaños.
- js: Carpeta que contiene todo el código JavaScript
    - letters: Carpeta que contiene el código JavaScript específico del juego.
        - core.js: Es el archivo núcleo del juego. Es el primero que se debe incluir en el documento HTML porque es el que define el namespace que contendrá todos los elementos del juego. También definirá constantes y funciones de utilidad generales al juego.
        - board.js: Este archivo define un nuevo namespace anónimo para aislar toda la lógica JavaScript que aquí se escriba del resto del juego. Añade un nuevo símbolo al namespace que es Board. Board es una clase que representará el tablero virtual del juego.
            - Su prototype define una propiedad el. Como veremos todas las clases que tengan una representación visual

HTML tendrán una propiedad llamada el, que guardará una referencia a su elemento HTML.

- letter.js: Este archivo contiene la definición de la clase Letter, que representará cada una de las letras que manejaremos en el juego. Como es una clase que representa un objeto visual del juego, también define su propiedad el.

Y con esto es suficiente para empezar a desarrollar nuestro juego. Es importante que vayáis entendiendo detalladamente todo lo que tenemos hasta aquí y cada paso que demos de aquí en adelante, así que, como siempre, si hay algo que no entendéis o alguna duda, preguntad inmediatamente.

Paso primero: Carga de imágenes.

La carga de imágenes la podríamos hacer escribiendo el código HTML directamente, pero para hacerlo más flexible y menos repetitivo la carga va a ser dinámica, es decir, cargaremos y mostraremos las imágenes usando JavaScript.

Cada imagen irá asociada a un objeto Letter y será colocada en el tablero (Board).

Empezamos por instanciar el tablero desde el documento HTML. Esto tenemos que hacerlo dentro de la función loadGame para asegurarnos de que cuando empecemos a trabajar con JavaScript, todo el contenido del documento ya está disponible. Este es el código de la función loadGame()

```
game = new LETTERS.Board("game_board");
game.loadLetters(['images/A.jpg', 'images/E.jpg', 'images/I.jpg',
'images/O.jpg', 'images/U.jpg']);
```

Con este código creamos un nuevo tablero, le indicamos cuál es el id de la donde debe mostrar su contenido y le indicamos que cargue las letras necesarias.
Debemos ahora modificar el código de la clase Board en consecuencia.
El constructor quedará así:

```
LETTERS.Board = function(htmlId) {
        var div = document.getElementById('#' + htmlId);
        if (!div) {
            div = document.createElement('div');
            div.id = htmlId;
            document.body.appendChild(div);
        }
        this.el = div;
    };
```

Hemos añadido el parámetro necesario para que reciba el id del elemento HTML donde mostrará su contenido y intentamos conseguir ese elemento del documento HTML. Si no existiera lo crearíamos nosotros mismos y lo añadiríamos al documento. Finalmente asignamos la propiedad 'el' al Nodo HTML que representa el tablero. Así nos evitaremos tener que seleccionarlo cada vez que lo necesitemos.

A continuación debemos implementar una función que se llame loadLetters y reciba un array por parámetro que le indicará la ruta a cada una de las imágenes que debe cargar como contenido de las letras.

```
LETTERS.Board.prototype = {
    el: null,    /* HTML Element */
    letters: null,

    loadLetters: function(images) {
    }
};
```

Así es como quedará el prototype de nuestra clase tablero. Daros cuenta que hemos añadido una nueva propiedad, que será un array donde se guardarán referencias a todas las letras que formen parte del juego. Este array debéis inicializarlo en el constructor de la función:

```
this.letters = [];
```

Ahora debemos rellenar el código de la función loadLetters para que cree un nuevo objeto Letter para cada letra.

```
loadLetters: function(images) {
    var self = this;
    images.forEach(function(img) {
        var letter = new LETTERS.Letter(img);
        self.letters.push(letter);
        self.el.appendChild(letter.el);
    });
}
```

Analizando este código hemos hecho un bucle funcional forEach que ejecutará una función anónima para cada elemento del array images. En esta función, por ser anónima, el valor de this es el objeto global, por tanto debemos utilizar el scope léxico de las funciones para tener la referencia a nuestro objeto Board dentro de la función anónima.

Si no hay ninguna duda, pasamos a analizar el contenido de la función anónima. Recibe por parámetro cada elemento del array. Creamos un nuevo objeto Letter para cada imagen, lo añadimos al array local de imágenes y añadimos al documento HTML el elemento HTML de cada objeto Letter que, como veremos a continuación, va a ser una imagen.

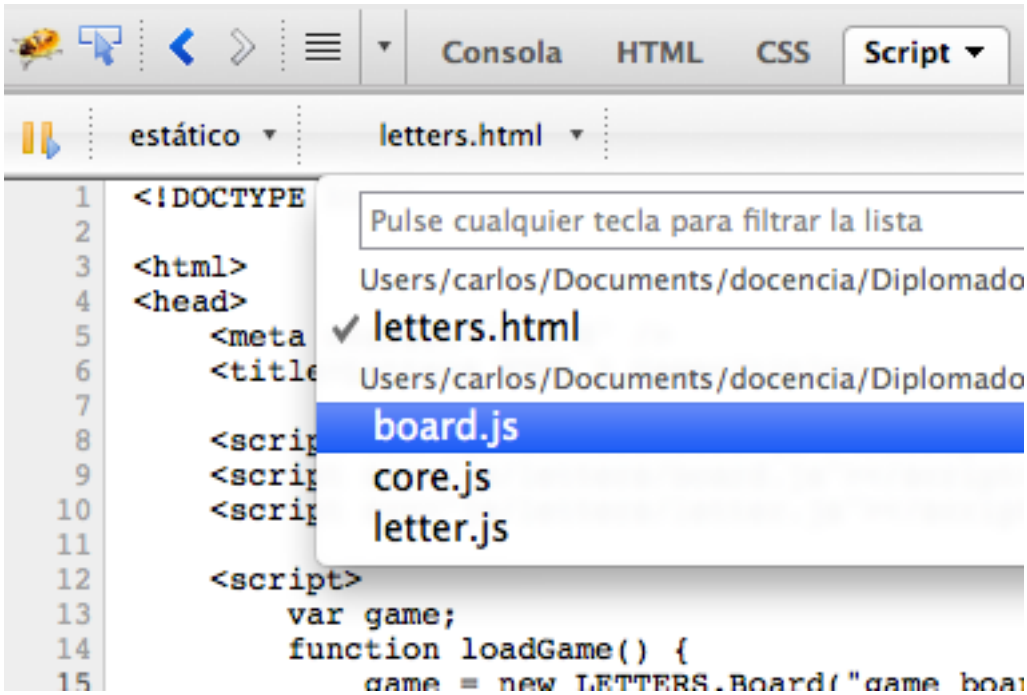
Ya solo nos queda la lógica de la clase Letter. Según lo que hemos visto hasta ahora, la clase Letter se tendrá que encargar de crear un nuevo objeto HTML imagen y cargar la imagen que se le indique por parámetro.

```
LETTERS.Letter = function(source) {
    this.el = document.createElement('img');
    this.el.src = source;
};
```

Este será el código de nuestra clase Letter, como hemos dicho antes, crea un nuevo elemento de tipo imagen y le asigna su atributo src al valor que recibe por parámetro, que es la ruta en disco de la imagen que debe mostrar.

Si guardamos todos los cambios y abrimos en nuestro navegador el archivo letters.html, podremos ver como las imágenes aparecen en el navegador, algunas muy grandes y otras más pequeñas.

Si no os aparece nada, seguramente habréis cometido algún error al escribir el código, utilizad la consola de Firebug, que os indicará los posibles errores que hayan ocurrido. Si no os aparece nada tampoco en la consola entonces debéis depurar el código usando la pestaña Script de la herramienta Firebug.
Para poner puntos de ruptura en documentos importados por el documento HTML, desplegad el menú de la figura, donde podréis elegir qué documento visualizar en Firebug.



Si aun así no encontráis el error, comparad el código de vuestro proyecto con el que habéis visto en este Lab. Como siempre, cualquier mínima duda que tengáis, ¡¡preguntadla!!.