2. DOM y JavaScript

Diseño y desarrollo de juegos en red (VJ1217), *Universitat Jaume I*

Tiempo estimado: 3 horas (+ 4 horas de ejercicios)

> "La teoría es cuando se sabe algo, pero no funciona. La práctica es cuando algo funciona, pero no se sabe por qué. Los programadores combinan la teoría y la práctica:. Nada funciona y que no saben por qué"- Desconocido (@CodeWisdom)

DOM en resumen, los programadores y desarrolladores web son capaces de acceder a la HTML Document Object Model[1], un documento HTML. Esto puede lograrse mediante el desarrollo de programas en lenguajes de programación del lado del cliente, tales como el lenguaje de programación JavaScript. JavaScript (JS) es un alto nivel, dinámica, sin tipo, e interpretado lenguaje de programación en tiempo de ejecución. Se ha estandarizado en el lenguaje ECMAScript especi fi cación, estando así estrechamente relacionada con otra ECMAScript similar a idiomas como el Ac tionScript (AS). JS puede acceder a las etiquetas HTML, propiedades y contenidos a través de DOM y es capaz de gestionar de forma dinámica todos ellos en tiempo de ejecución. En esta sesión de laboratorio, se le presentó a algunos conceptos básicos de la JS y DOM, y combinará tanto para el desarrollo de animaciones básicas y proyectos de manejo de eventos.

Contenido

[1] https://en.wikipedia.org/wiki/Document_Object_Model

1 ¿Qué es el código HTML DOM

Tsucesivamente, es un modelo de objeto y programación interfaz estándar para HTML. Documento Modelo de objetos, o DOM para abreviar- cuando se carga una página web, y su propósito es permitir JavaScript (u otros lenguajes de programación) para acceder, cambiar, añadir y borrar cualquier elemento de un documento HTML, así como sus propiedades , métodos y eventos.

1.1 DOM árbol de objetos

A medida que el siguiente fragmento de código y la fig. 1 deben ilustrar, el DOM es principalmente un modelo de objeto estructurado como una *árbol de objetos*.

```
<! DOCTYPE html>
<Html>
    <Head>
        <Title> HTML DOM </ title> </ head> <body onclick = "This.innerHTML =
    '<p> Listo </ p>'" >

        <div id = "Idx"
            = estilo "Position: absolute; izquierda: 50px" >
            <p = estilo "color rojo" > ¡HAZ CLICK EN MI! </ p> </ div> </
    body> </ html>
```

Cada HTML, CSS, JavaScript e incluso, elemento in- cluded en una colección de documentos de la construcción de una página Web que se muestra por un navegador se representa como un nodo (objeto) en un árbol como la mostrada en la Fig. 1 .

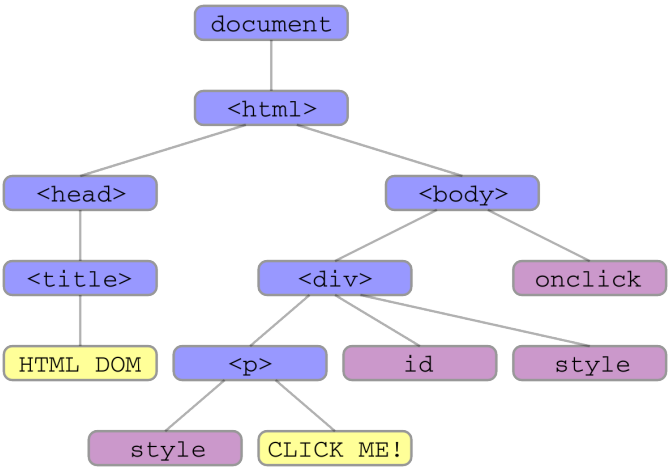

Figura 1: *Árbol de objetos para el código HTML en la Sección. 1.1 .*

1.2 Interfaz de programación de DOM

UN *interfaz de programación* envolver un modelo de este tipo, el árbol de objetos, es proporcionada por el DOM para acceder y gestionar todos los elementos almacenados en el árbol, teniendo en cuenta su di ff naturaleza Erent: HTML / elementos CSS, propiedades (que especifican características de los objetos), métodos (acciones de fi nir eje- cutable sobre los objetos), eventos (objeto de notificación o cambios en el entorno). . .

Cualquier elemento de un documento HTML (nodo de un árbol DOM) puede ser accesado a través de la prede fi nida documento objeto en JavaScript, que o ff ers met- ods a fi nd elementos de hormigón en el DOM, tales como getElementById (), getElementsByClassName () o getElementsByTagName (), o propiedades a los elementos directamente ser seleccionado, tales como cuerpo o título.

```
<Script>
        documento .título = 'DOM Interfaz de programación' ;
        documento .getElementsByTagName ( "pag" ) [ 0 ] .innerHTML =
                documento .body.nodeName;
        dejar X = documento .getElementById ( "Idx" );
        x.innerHTML = '<H1> Nueva texto </ h1>' + x.innerHTML;
</ Script>
```

los título conjuntos de propiedades o devuelve la cadena de título del documento actual, mientras que el cuerpo propiedad es un objeto que representa la sección HTML < body> . Cada elemento HTML representado en el DOM tiene una nombre del nodo pro- piedad que contiene su nombre de etiqueta en las capitales. El método getElementById () Devuelve el objeto identificado por su gument Ar-, y getElementsByTagName () devuelve un ción colec- (array-como contenedor), compuesto de todos los objetos que coinciden con el argumento tagname. Ambas búsquedas se ejecutan dentro del alcance del objeto invocante. cuestiones de JavaScript sintaxis se consideran a continuación, pero las frases de este ejemplo deben ser comprensibles.

Aprender más acerca de. . .

- documento .getElementById ()
- documento .getElementsByClassName ()
- documento .getElementsByTagName ()

1.3 Conexión DOM con CSS

DOM también permite acceder y cambiar el estilo de los elementos HTML, es decir, sus propiedades de CSS. Cada elemento HTML (objeto de la DOM) proporciona una estilo propiedad, que sostiene un objeto que tiene propiedades DOM para todas las posibles propiedades de estilo CSS. Estas propiedades DOM almacenan los valores de cadena, que podemos establecer para modificar cualquier aspecto del estilo del elemento.

```
<Script>
        dejar w = documento .getElementById ( "Idx" );
        w.style.left = '200px' ;
        w.style.backgroundColor = 'verde claro' ;
        dejar y = documento .getElementsByTagName ( "pag" ) [ 0 ];
        y.style.color = 'púrpura' ;
        dejar z = documento .getElementsByTagName ( "H1" ) [ 0 ];
        z.style.fontFamily = 'mensajero' ;
</ Script>
```

Algunos nombres de las propiedades de estilo CSS contienen guiones, tales como " color de fondo" . Tales nombres son torpes Para trabajar con en JavaScript y que deberían ser diez ESCRITO como estilo [" color de fondo"] . Como alternativa podemos utilizar los nombres de propiedades DOM que sus guiones que se habían quitado y las letras que les siguen en mayúsculas (style.backgroundColor).

2 Conceptos básicos de JavaScript

JavaScript (JS) fue desarrollado por Brendan Eich en 1995, sólo para mejorar la interactividad del usuario para las páginas web de Netscape Navigator, el navegador más popular en ese momento. Desde su estandarización por ECMA Internacional en 1997, como ECMA-262, que ha ido evolucionando para mejorar la navegación web funciona- lidad y se ha incorporado a muchos otros navegadores modernos. ECMAScript es el o fi cial nombre de la lengua, siendo por tanto un sinónimo de JS. Por el contrario, a pesar de su similitud con el lenguaje de programación llamado Java, no tienen nada que ver entre sí.

En HTML, JS código debe ser insertado entre < script> y </ script> las etiquetas. Cualquier número de secuencias de comandos se puede colocar en un documento HTML, ya sea en el < body> o < head> secciones, o incluso en ambos. Sin embargo, colocándolos en la parte inferior de la < body> elemento es muy recomendable para acelerar la visualización de páginas web.

Para facilitar la reutilización del código, en lugar de incrustar scripts en archivos HTML, el código JS también se puede colocar en archivos externos, cuyos nombres debe terminar con el expediente de extensión. js. En este caso, el nombre del script fi l debe ser escrito en el src (fuente) atributo de una < script> etiqueta:

```
<script src = "JS / shieldhammer.js" > </ script>
```

Varios fi guión les puede añadir a la misma página, utilizando una < script> etiqueta para cada archivo. Además, nota que el código JS en un script fi l no puede contener < script> etiquetas, al igual que < style> las etiquetas no están incluidos en una CSS fi l.

Tu turno 1

Abrir el proyecto CapShieldVsMjolnir y todos sus archivos. Ejecutar el proyecto de código de Visual Studio y comprobar su funcionalidad. Para ello, haga clic en el

index.html fi l y selecciona la opción “Abrir con Live Server”. Vamos a estudiar su código más adelante en esta sesión de laboratorio. Tenga en cuenta la estructura del sitio y comprobar las referencias a archivos externos en index.html. Además, lea el código de la fi les index.html y clashscene.css y tratar de relacionar las etiquetas y atributos a lo que se muestra en la ventana del navegador.

Valores, tipos y operadores

Siete tipos básicos de valores se proporcionan en JS: Number, String, Boolean, fi Unde definido, Null, Símbolo, y el objeto, por lo que la primera de seis se clasifican como (valores inmutables) Primitivos, y el último como Complex. Los literales para los tres primeros tipos primitivos se pueden escribir obedeciendo cierto, más o menos, las reglas comunes: Los números se escriben con (3.14159) o sin (20) decimales, o con c notación científica (exponencial) (3141.59 mi -3); Las cadenas se de- texto limitados por comillas simples o dobles (" rojo" , 'azul claro');

y, sólo hay dos valores booleanos (cierto y falso).
Por otro lado, sólo hay un valor fi nida Unde (indefinido , para las variables que existen, pero no tienen nada como- firmado a ellos) y sólo un valor nulo (nulo , para las variables que existen y tienen nulo asignado intencionalmente a ellos).

Obviamente, los valores de datos se almacenan en variables, que se declaran utilizando la dejar palabra clave. Al mismo tiempo que una variable se declara un valor puede ser asignado a la misma, a través de la asignación (=) Del operador, pero esto no es obligatorio. Muchas variables pueden ser declaradas en una sola declaración, separándolos por comas.

```
dejar colores; // <=> dejar que los colores = indefinido;
const _ Co10r $ = "arco iris" ;
dejar parte superior = 0 , abajo a la izquierda = 60 , derecho; fondo = falso
;
```

Una variable declarada simplemente no tiene valor; es decir, técnicamente, tiene el valor indefinido . las variables tienen JS tipos dinámicos: la misma variable puede contener di ff Erent tipos de datos. De este modo, las asignaciones anteriores a fondo son validos.

Las variables declaradas usando const son *constantes:* sus ues Val- no se pueden cambiar una vez establecido. Por esta razón, cada constante se debe formatear en la declaración. Pero, a excepción de un valor tan inmutabilidad, const declaraciones son tratados como dejar declaraciones.

(... variables, funciones) Reglas para nombres legales identi fi cador en JS son similares a la mayoría de los lenguajes de programación: una se- cuencia de letras, dígitos, guiones bajos (_), o signos de dólar incluso ($), que no parta de un dígito. código JS es sensible a mayúsculas y el uso de codificación charset Unicode.

barras dobles, // , Iniciar un comentario que se extiende hasta el final de la línea actual. Múltiples líneas de comentarios se escriben entre / * y * /.

operadores disponibles en JS, que forman expresiones, también son ampliamente compartidos entre los lenguajes de programación: Aritmética: + - * /% (Módulo) ++ (Incr). - (Decr.) Asignación: = + = - = * = / =% = Comparación: ==! = > < > = <=

JS operadores que abordan la comprobación de tipos

operadores de tipo tipo de y en vez de vamos a sabido de tipos de variables y objetos. Por ejemplo,
tipo de [] dice Objeto pero [] en vez de Formación
es cierto , ya que [] es la matriz vacía y Formación es un tipo especial de Objeto .
operadores de comparación fi c JS especí === y ! == deter- minar la igualdad o di ff rencia tanto en valor y el tipo de sus operandos, en contraste con los operadores compara- comunes == y ! = , Dicho control sólo para la igualdad o di ff rencia en valor. Por ejemplo, 0 == "0" es

cierto , gracias a las reglas de conversión de tipo automático, pero
0 === "0" es falso .

Lógico: && (y) || (o) ! operadores (no) Otros JS son los JS especí fi co operadores de comparación === y ! == , El operador ternario condicional (?:), los operadores de tipo (tipo de y en vez de), y bit a bit eral y bit a bit de asignación de los operadores de SeV. La precedencia de operadores de JS también se puede inferir su mayoría de precedencias conocidos en lenguajes de programación estándar. Recuerda que siempre se puede romper por medio de paréntesis.

Aprender más acerca de. . .

operadores Asignación Las comparaciones

Estar al tanto de las normas permisivas de comprobación de tipos en JS expresiones, especialmente en condiciones y expresiones involv- ing números y cadenas. Con frecuencia, no será notificada una coincidencia de tipos, pero usted podría experimentar re- sultados inesperados. Pero a veces, puede utilizarlo para su propio beneficio. Por ejemplo, basado en el código de nuestro proyecto de ejemplo, frases siguientes cambian la posición izquierda de un objeto DOM:

```
dejar sh = documento .getElementById ( "Capshield" );
dejar yo = parseInt (Sh.style.left.slice ( 0 , -2 )); yo + = 23 ; sh.style.left = yo +
'Px' ;
```

La posición de la izquierda (propiedad de estilo de tipo de cadena) de una < div> elemento, identi fi ed con la cadena " capshield" , se corta (a través de un built-in método de cadena) para conseguir sus dígitos solamente, convierten entonces en el número correspondiente mediante la función global JS parseInt , siguiente incrementa en 23 unidades como un número, y finalmente convertidos de nuevo a cadena donde este número se concatena con la cadena ' px' y asignada como una cadena a la propiedad de estilo correspondiente.

Recordemos que el operador + agrega números sino cadenas Nates concatenación, y si sus operandos son un número y una cadena, el resultado será una cadena.

Objetos y arrays

En JS, una *objeto literal* que permite tanto definen y crear un objeto en una sola declaración. Se trata de una lista de *propiedades*
Escrito como nombre : valor pares, separados por comas y delimitados por llaves.
En su forma más simple, una *objeto vacío* se especi fi cada por la {} objeto literal.
Después de la creación, una

nueva propiedad se puede agregar a un objeto existente simplemente dando un valor. Las siguientes dos frases crear el objeto
choque con dos propiedades iniciales, proteger y mjolnir,
y luego añadir una tercera, Temporizador, al objeto.

```
dejar choque = {proteger : indefinido , mjolnir : indefinido };
clash.timer = indefinido ;
```

La propiedad de objeto proteger a continuación, se puede acceder como
clash.shield. Nuestro proyecto de ejemplo se utiliza el principal ob- ject choque para representar y gestionar toda la escena a ser animado.

Otros elementos en objetos son el *métodos,* que per- acciones de formulario en los objetos y en realidad son propiedades de la función de fi niciones de almacenamiento. Ellos, como propiedades, también se pueden añadir a los objetos existentes después de su creación.

Posponemos la explicación del método especí fi cación hasta secta. 4 , Pero podemos introducir invocación de método ya que ciertos métodos están siempre disponibles para todos los objetos (por herencia). Por ejemplo, cualquier objeto puede ejecutar Encadenar, un método para convertir a sí mismo una representación de cadena. Para invocar este método en choque objeto, por ejemplo, escribimos

clash.toString () (Se requieren paréntesis). Además, los métodos que necesitan argumentos tienen que ser invocado proporcionar valores adecuados para ellos dentro de los paréntesis.

Los objetos también se pueden declarar constantes, por medio de
const . Pero, en tal caso una, sólo la unión con el objeto es inmutable; es decir, que no será capaz de asignar un nuevo objeto al er fi cación. Sin embargo, podríamos cambiar ninguna de sus propiedades y métodos, o añadir otras nuevas. UN const
declaración en un objeto realmente impide modi fi cación de la unión y no del valor en sí.

Similares a los objetos, una *literal de matriz* crea una matriz en un estado de cuenta, a través de una lista de valores (expresiones), separados por comas y delimitado por corchetes.

```
dejar colores = [ "Cian" , "verde" , "gris" , "azul" ];
```

Como de costumbre, se accede a cualquier matriz completa JS a través del nombre de la matriz y cualquier elemento de la matriz es especi fi cada por un número de índice entre corchetes, como colores [i]. JS array indización se basa-cero, lo que significa que el primer elemento tiene índice 0, y una *matriz vacía* se define por la [] literal array.

la matriz colores tiendas, en nuestro proyecto de ejemplo, las etiquetas de color disponibles que se van a utilizar para cambiar el color de trasfondo en la escena animada.

JS matrices son un tipo especial de objetos. De hecho, el
tipo de operador devuelve ' objeto' cuando se aplica a una matriz. Por otra parte, las matrices JS permiten para almacenar di ff Erent tipos de elementos en la misma matriz.

JS matrices o ff er muchas propiedades integradas y métodos que facilitan la escritura de código. Una de tales propiedades es
longitud, que tiene el número real de elementos de matriz. En nuestro proyecto de ejemplo, la siguiente frase utiliza la
longitud propiedad de la matriz colores, a los índices de cómputo en el rango permitido.

```
const SALTO = 7 ;
. . .
CINDEX = (CINDEX + SALTO) % colors.length;
```

A incorporada en el método matriz útil es empalme, que añade o artículos saca a / desde la matriz. Su primer parámetro de fi ne la posición en la que las operaciones se llevan a cabo, es decir, donde se añaden nuevos elementos o elementos existentes se eliminan de. El segundo parámetro específico fi ca el número de elementos a eliminar. Y el resto de los parámetros de proporcionar los nuevos elementos que se añadirán. Tenga en cuenta que los parámetros dos primeros pueden ser 0 y sólo ellos son obligatorios. Trate de entender las frases siguientes, extraídos de nuestro proyecto de ejemplo.

```
dejar yo = 0 ; colors.splice (i, 0 , "ligero" + colores [i]); colors.splice (i 2 ,
0 , "oscuro" + colores [i 1 ]);

// colores === [ "lightcyan", "cian", "Darkcyan", //
                                "Verde", "gris", "azul"]
```

Desde una posición inicial yo, el " ligero" versión del color almacenada en el índice yo se inserta antes de que el color existente en la matriz, y luego su " oscuro" versión se inserta después de ella.

Estructura del programa

No existen normas o fi ciales para el objeto Browser Modelo (BOM), pero es el nombre común a menudo se utiliza para referirse a los métodos y propiedades para JS interactividad que los navegadores modernos han puesto en práctica y que son (casi) la misma entre todos los navegadores.

los ventana objeto es soportado por todos los navegadores y representa la ventana del navegador. Entonces el documento
(HTML DOM) objeto se proporciona como una propiedad de la
ventana objeto, es decir, ventana . documento es lo mismo que
documento . Tenga en cuenta que no se puede sobrescribir cualquier er fi cación estándar global, como ventana o documento , utilizando dejar o
const declaraciones: sólo se puede sombra de ella. Para cada dejar
y const Declaración en el ámbito global, una nueva unión se crea en este ámbito, pero ninguna propiedad se añade a la
ventana objeto.

De esta manera, vamos a iniciar la ejecución de nuestro código JS basado en la disponibilidad de la ventana objeto (BOM) en lugar de la de la documento objeto (DOM). Espera- ción de la lista de materiales es más seguro que sólo se espera para el DOM, para iniciar la ejecución de código JS, ya que requiere más contenido (imágenes, css, guiones, etc.) para ser cargado y listo.

Por lo tanto, en nuestros proyectos CSS / HTML / JS, JS en el principal archivo, siempre vamos a ofrecer una oración como:

```
ventana .onload = punto de entrada;
```

Entonces, una función llamada punto de entrada (o lo que sea el nombre que desea emplear) también debe ser de fi nido, que contiene las acciones a partir de nuestro programa. En la Sección. 4 vamos a aprender cómo dE funciones ne fi y llamadas. Por otra parte, en la Sección. 8
vamos a conocer el significado de las onload propiedad / método en el ventana objeto y, en general, en cualquier objeto.

Por otra parte, los programas de JS utilizan puntos y comas a las declaraciones del tipo de la SEPA, a pesar de que son opcionales en la mayoría de los casos. Para evitar pensar en qué casos no se requiere o no, todas las declaraciones finales ing con punto y coma es muy recomendable. Además, las palabras clave son palabras reservadas JS, y esto significa que no pueden ser utilizados como los identificadores.

depuración sencilla

Aunque las estrategias y herramientas de depuración avanzadas se pueden utilizar, por el momento, vale la pena mencionar una rudimentaria todavía uno útiles: **imprimir mensajes a la consola del navegador. Por ejemplo, podríamos querer saber el valor de la variable energía antes de la función checkForCollisions () se** llama. Podríamos entonces simplemente escribir:

```
console.log ( "Antes de checkForCollisions (), energía =" ,energía);
```

Dado que los mensajes de error también se muestran en la consola, es una valiosa herramienta para el desarrollador web (juego). Dentro de un navegador, pulse Ctrl + Shift + I para mostrar (u ocultar) la consola y otras herramientas de desarrollo. El contenido de los objetos JS se pueden visualizar fácilmente con el método Encadenar():

```
console.log (colors.toString ());
```

También puede encontrarla útil para visualizar un cuadro de alerta con un poco de información. Es más perjudicial que console.log, porque tiene que ser cerrado de forma explícita, pero este inconveniente puede convertirse en una ventaja cuando se desea esto.

```
alerta( "Existen " + stars.length + " estrellas" );
```

3 JavaScript: instrucciones de control

Como se podría esperar, JS ofrece frases de control- ling el flujo de ejecución. Te encontramos que son estructuras condicionales y bucles típicos, muy similares a los de otros lenguajes de programación.

ejecución condicional

La sentencia condicional típico de JS es **Si** , que nos permite ejecutar acciones di ff Erent en base a una condición (expresión), delimitado por paréntesis y que se evalúa como un valor booleano. Cuenta con una sección opcional **más** , y requiere de ambos bloques de oraciones dependientes estar delimitados por llaves si contienen más de una frase. Las llaves son opcionales, cuando sólo contiene una declaración.

Mira el siguiente ejemplo para identificar todos por encima de men- ciona partes. En este código, comprobamos si la posición izquierda de un objeto más dos veces su anchura va más allá de la posición izquierda de otro objeto. En nuestro proyecto de ejemplo, estas frases cuando se detectan las armas son lo suficientemente cerca de aumentar la frecuencia de cambio de los colores de fondo.

```
const PERÍODO LARGO = 20 ;
const PERÍODO CORTO = 1 ;
dejar período;
dejar sh = clash.shield, mj = clash.mjolnir;
Si ( sh.left + 2 * sh.width > mj.left)
    período = PERÍODO CORTO;
más
    período = PERÍODO LARGO;
```

JS sintaxis permite la escritura sucesiva **else if** secciones, entre **Si** y **más** , para la gestión de varias pruebas con más de dos opciones.

los **cambiar** declaración proporciona una forma alternativa para seleccionar uno entre muchos bloques de código a ser ejecutado.

```
cambiar ( /* expresión*/ ) {
  caso /* la expresión * 1 / :
    // bloque de código para la expresión 1
    descanso ;
  caso / * expresión 2 * / :
    // bloque de código para la expresión 2
    descanso ;
  . . .
  defecto :
    // valor predeterminado bloque de código
}
```

los **cambiar** la expresión se evalúa una vez. Entonces, su valor se compara en orden de aparición, por medio de === , Con los valores de las expresiones de cada uno **caso** etiqueta. Si hay una coincidencia, se ejecuta el bloque asociado de código. los **defecto** palabra clave específica es el código para ejecutar cuando no hay ninguna coincidencia. los **descanso** palabra clave se detiene la ejecución de más de código y pruebas dentro del caso **cambiar** declaración. Por lo tanto, no es necesario en el último caso.

Aprender más acerca de. . .

Si ... más **cambiar**

bucles

JS soporta tipos di ff Erent de bucles: lo normal **para** y **mientras** frases, y la menos frecuente **para - en** y **hacer - mientras** estructuras. Al igual que las sentencias condicionales, todos sus bloques de frases dependientes deben estar delimitados por llaves si contienen más de una frase, pero las llaves pueden ser omitido es que sólo hay una.

Observe el siguiente **para** ejemplo y tenga en cuenta los tres expresiones dentro de los paréntesis y separados por punto y coma:

```
para ( dejar yo = 0 ; yo < colors.length; yo + = 3 ) {Colors.splice (i, 0 , "ligero" + colores [i]); colors.splice (i 2 , 0 , "oscuro" + colores [i 1 ]); }
```

La primera expresión Inicializa las variables a utilizar en el código dependiente. Se evalúa sólo una vez, antes de que comience el bucle. La segunda expresión es la condición para ejecutar el bucle y se evalúa antes de cada iteración. Y el tercero de los cambios en las variables fi nes después de cada iteración, es de esperar que con el tiempo termina el bucle. En nuestro proyecto de ejemplo, completa este bucle colores matriz para inserto

"ligero" (antes) y " oscuro" (después) de versiones para los colores inicialmente de fi nido.

los **mientras** (*/ * condición*/)* ... sentencia de bucle a través de un bloque de código dependiente, mientras que la condición es verdadera. La expresión de esta condición debe evaluarse como un valor booleano y se evalúa antes de cada iteración.

los **hacer ... mientras** (*/ * condición*/)* frase es una variante del bucle while que pone a prueba su condición *después* escorrentías Ning su bloque de código. Esto significa que al menos una iteración se garantiza la ejecución.

los **para** (*/ * variable*/ **en** / * objeto*/)* ... tence sen- sirve para recorrer a través de las propiedades de un objeto.

Aprender más acerca de. . .

para mientras

cuestiones de bloques y alcance

Variables y constantes declaradas a través **dejar** y **const** son *declaraciones a nivel de bloque,* similar a la de lenguajes basados en C: fuera inaccesible del bloque de declaración o función, tanto delimitada por llaves (véase el cuadro ING Hoist-). Cada variable o constante se crea en el lugar donde se produce su declaración, lo que significa que no se puede utilizar antes de que se declara. Por lo tanto, si está destinado a estar disponible para el bloque entero, la función o el alcance global, entonces se debe declarar a principios del alcance. Por otro lado, una **dejar** o **const** declaración puede crear una nueva variable con el mismo nombre que una variable existente en su ámbito de aplicación que contiene. Esta nueva variable sombras de la existente.

Por otra parte, la sección de inicialización de una **para** bucle es parte del bloque subsiguiente; es decir, una **dejar** declaración dentro de ella hace que la declaró variable disponible sólo en el interior del bucle, y en otros lugares ya no es accesible una vez que el bucle se ha completado.

4 funciones JavaScript y clases

Funciones permiten la reutilización de código, que es, por de fi nir el código una vez, y usarlo muchas veces, probablemente con di ff Erent argumentos con el fin de producir di ff Erent resultados. Las clases permiten generalizar el comportamiento de los objetos y, sobre todo en JS, su definición está estrechamente relacionada con la función de fi nición.

funciones

Una función de JS se define o se declara la **función** palabra clave, seguido por su nombre y los paréntesis, dentro de la cual los nombres de parámetros separados por comas pueden ser incluidos. reglas de identi fi cador se aplican tanto a los nombres de funciones y parámetros, y los parámetros se comportan como variables locales dentro del ámbito de la función. El código que se ejecuta

izado

Las declaraciones de variables utilizando el **var** palabras clave y la función JavaScript en declaraciones son tratados como si estuvieran colocados en la parte superior de la función o el alcance mundial (si es declarado fuera de una función), independientemente del lugar donde se produce la declaración real. Este comportamiento por defecto se conoce como ***izado.*** Debido a esto, las funciones y este tipo de variables (**var**) Puede ser utilizado antes de que se declaran. Pero tenga en cuenta que sólo se iza JavaScript declaraciones, no Inicializaciones. Si ***izado*** es desconocida, se pasa por alto o mal entendido por un desarrollador, el código podría correr con los bugs (errores). Para salir de ella, mientras se trabaja con **var** , Siempre declarar todas las variables al comienzo de cada ámbito, dado que este es el comportamiento predeterminado de JavaScript. Alterna- tivamente, sólo para uso **dejar** y **const** palabras clave para declarar variables y constantes, y el trabajo siempre en un tratamiento más estándar, a nivel de bloque de declaraciones, que es algo que recomendamos encarecidamente.

por la función se coloca dentro de llaves. Compruebe todas estas características, a excepción de los parámetros, en el siguiente ejemplo.

```
función initBackground () {
    para ( dejar yo = 0 ; yo < colors.length; yo + = 3 ) {Colors.splice (i, 0 , "ligero" + colores
      [i]); colors.splice (i 2 , 0 , "oscuro" + colores [i 1 ]); };


    dejar decIndex = Matemáticas .aleatorio() * colors.length; CINDEX = Matemáticas
    .floor (decIndex);
    dejar bg = documento .getElementById ( "campo de batalla" );
  bg.style.backgroundColor = colores [CINDEX]; };
```

Esta función inicializa el color de fondo en nuestro proyecto ejem- plo. Recordar la finalidad de la **para** , explica en la sección bucles, y observar cómo se genera un índice entero aleatorio para seleccionar un color entre los disponibles en el colores formación. los aleatorio() método de la **Matemáticas** objeto (no un constructor) devuelve un número aleatorio entre 0 (inclusive) y 1 (exclusivo), que luego es a escala en el número de colores y trunca al número entero inferior más cerca de él.

Aprender más acerca de. . .

matemáticas objeto y su método aleatorio

Cuando se invoca una función, proporcionando argumentos para sus parámetros, si es necesario, se ejecuta su código. Aparte de algunos casos especiales, una función JS se llama normalmente cuando se ejecuta una instrucción de llamada explícita en el código o se produce un evento. En el siguiente ejemplo, una declaración llamada explícita a la anterior de fi nido initBackground () aparezca la función. Del mismo modo, una llamada explícita al método initWeapons () del choque objeto se ejecuta. Este método es de fi nida y se explica a continuación. Más tarde, se discutirá la invocación de una función en el contexto de la gestión de eventos.

```
función punto de entrada() {
    initBackground ();
```

```
clash.initWeapons ();
const T = 1000/50 ; // 20 ms
clash.timer = setInterval (clash.animate, T); };
```

Tenga en cuenta que la función punto de entrada() lleva a cabo algunas tareas iniciales en nuestro proyecto de ejemplo, la organización de fondo y las armas. además, el setInterval () función, en realidad un método del objeto ventana , pone en marcha un temporizador para el método animar() del objeto choque, para ser ejecutado cada 20 milisegundos, es decir, 50 veces por segundo. Haciendo animar() para generar y mostrar un (posiblemente di ff Erent) fotograma cada vez que se ejecuta, una escena animada se puede mostrar en el navegador.

Aprender más acerca de. . .

ventana .setInterval ()

Así, el método animar() deben ser declarados como si- mínimos, que también ilustra que la definición de un método de objetos JS es simplemente la asignación de la definición de una función anónima a una propiedad.

```
clash.animate = función () {
    clash.advanceWeapons ();
  clash.changeBackgroungColor (); };
```

Tareas gestionadas por animar() relacionarse con el avance de las armas y el cambio de color de fondo, y se llevan a cabo de forma independiente por otros dos choque métodos.

Además de presentar un segundo ejemplo de ración método Decla-, el código de advanceWeapons () destaca algunos hechos impor- tante. En primer lugar, en general advanceWeapons () divisiones de operación en micro-movimientos individuales de los dos elementos de la escena, uno de ellos el aumento y la disminución de la otra. Entonces, después de modificar su posición en la escena, su eventual choque se prueba y, si tiene éxito, anima- ción se detiene llamando al clearInterval () función con el Temporizador obtenido al principio.

```
clash.advanceWeapons = función () {
    const SHIELD_MOVE = 1 ;
    const MJOLNIR_MOVE = -2 ; clash.shield.move
    (SHIELD_MOVE); clash.mjolnir.move
    (MJOLNIR_MOVE);
    dejar sh = clash.shield, mj = clash.mjolnir;
    Si ( sh.left + sh.width / 2> mj.left)
        clearInterval (clash.timer); };
```

Funciones menudo calculan un valor que debe ser devuelto a su llamador. En JS, el regreso palabra clave se utiliza en el código de una declaración de función para colocar el punto de devolverle el valor. Cuando el fl ujo de ejecución alcanza una regreso declaración, la función se detiene su propia ejecución y el flujo continúa en la siguiente línea de código que invoca. Invocaciones a funciones que devuelven un valor pueden ser colocados en todas partes una expresión para ese tipo de valor se puede utilizar.

Tu turno 2

En shieldhammer.js definen una función, llamada randomArrayIndex, para generar y devolver un índice entero dom ran- dentro de la gama actual de la matriz que esta función tiene que recibir como parámetro. Con este fin, utilizar las frases correspondientes de InitBackground () y, después de haberlo escrito, reemplazar estas frases por una llamada a la función adecuada.

clases

Además de declarar y la creación de objetos individuales, JS declaremos *tipos de objetos* que luego se puede utilizar para crear muchos objetos reales a partir de ellos. La forma estándar de hacerlo en ECMAScript 6 es por medio de *declaración de la clase.*
Observar la posterior declaración:

```
clase arma {
    constructor (x) {
        esta . imagen = X;
        dejar s = esta . image.style.left;
        esta . izquierda = parseInt (S.slice ( 0 , -2 )); s = esta . image.style.fontSize;

        esta . anchura = parseInt (S.slice ( 0 , -2 ));
    }; mover (n) {

        esta . izquierda + = norte;
        esta . image.style.left = esta . izquierda + "Px" ; }; };
```

Arma es un *declaración de la clase:* que comienza con la clase palabra clave seguida por el nombre de la clase, un método structor con- objeto especial llamado constructor y el uso de la esta palabra clave en sus oraciones. Nombrando a clases con se recomienda un fi mayúscula primera letra, con el fin de que coincida con JS-construido en los constructores de nomenclatura estilo. *propiedades propias,* los que ocurren en la instancia (objeto) en lugar de la proto- tipo (clase), solamente se pueden crear en el interior de un método de clase. Por lo tanto, la creación de todas las propiedades propias dentro de la constructor método es aconsejable. De esta manera, un solo lugar en la clase será responsable de todos ellos. Los métodos de clase deben ser colocados dentro de la declaración de la clase; no hay manera directa por adición de ellos fuera del alcance de esta declaración, como lo fue el caso de los objetos (véase el cuadro Clase-como las estructuras en ECMAScript 5).

Darse cuenta de esta no es una variable: no se puede cambiar su valor. Es una palabra clave que actúa como un marcador de posición para el futuro de referencia real de cualquier objeto construido. En el momento de creación del objeto, esta será sustituido por la referencia de objeto real. En otras palabras, esta representa el objeto que “posee” el código JS.

Dentro de nuestro proyecto de ejemplo, los objetos construidos a través de esta clase servirán para el modelo en el código de las armas que se mueven en la escena. Por lo tanto, el objeto DOM (< div>), en que cada uno está conectado, se pasa como un argumento, y propiedades para almacenar los datos necesarios y un método para actualizar que se proporcionan. los imagen propiedad se mantiene una referencia de acceso al propio objeto DOM, requerido para la gestión de la style.left propiedad del objeto DOM,

Estructuras en ECMAScript Clase 5 similar

ECMAScript 6, también conocido como ECMAScript 2015, in- troduced un nuevo ***sintaxis*** para la clase de estructuras similares a espec- i fi cado en ECMAScript 5 que no se estableció un nuevo modelo de herencia orientada a objetos para JavaScript, pero no es más que una renovación sintáctica de la herencia basado en prototipos existentes.

Comparación de la sintaxis clásica (abajo) con el nuevo:

```
función Arma (x) {
    esta . imagen = X;
    dejar s = esta . image.style.left;
    esta . izquierda = parseInt (S.slice ( 0 , -2 )); s = esta . image.style.fontSize;

    esta . anchura = parseInt (S.slice ( 0 , -2 ));
};

Weapon.prototype.move = función ( n) {
    esta . izquierda + = norte;
    esta . image.style.left = esta . izquierda + "Px" ; };
```

En este código, Arma es una ***función de objeto constructor*** que crea tres propiedades. los moverse como- método se firmó con el Arma prototipo, por lo que el mismo código de función será compartido por todos los objetos creados a partir de la Arma función constructora. Esto también puede utilizarse como una manera indirecta para la adición de un método fuera del alcance de una declaración de clase,

Descripciones más detalladas se pueden encontrar en:

- MDN documentos web - MOZ: // a. Las clases de JavaScript underst.
- ECMAScript 6 - Presentación de las clases JS

Tu turno 3

Copiar todo el proyecto CapShieldVsMjolnir a uno nuevo llamado CapShieldVsMjolnirC A continuación, cierre todos los archivos desde el primer proyecto y abrir todos los del proyecto duplicado. Ahora en

shieldhammer.js, declarar una clase, llamada Choque, capaz de construir un nuevo objeto choque plenamente equivalente al objeto actual choque. Por último, sustituir en el código del objeto actual choque por la nueva declaración de la clase y un nuevo objeto choque construido a partir de Choque.

5. programación orientada a objetos con JavaScript

JavaScript proporciona algunas características de algún tipo de programación orientada a objetos por medio de *prototipos,* que de hecho son objetos acoplados automáticamente como una propiedad predeterminada prototipo a cada objeto creado. Así, por ejemplo, la declaración de clase Arma hereda su prototipo de propiedad Función .prototipo, y cada objeto creado a partir de Arma, como clash.shield, hereda su prototipo de propiedad Weapon.prototype.

las relaciones entre los prototipos de objetos JavaScript forman una estructura en forma de árbol, con raíces en Objeto .prototipo. Proporciona unos métodos que están disponibles en todos los objetos, tales como Encadenar, que convierte un objeto a una representación de cadena. Este tipo de herencia limitada se conoce como *basada en la herencia de prototipo.*

Los objetos pueden JavaScript o interfaces de ff er para comunicarse con sus entornos, con la intención de *ING encapsulat-* los detalles internos que los componen. Además, JavaScript también permite la escritura de código con *polimorfismo,* de modo que di ff Erent objetos se desarrollan para exponer la misma interfaz y otras partes de la obra de código en cualquier objeto que tiene que interfaz común. Y, por otra parte, una especie de *la especialización de clase* puede llevarse a cabo en JavaScript a través de la herencia de prototipo de los constructores de objetos, lo que permite un tipo para la obtención de su prototipo de la de otra y después añadiendo y anulando propiedades según sea necesario.

Todas estas características ayuda para escribir clara y más organizado y código reutilizable. Se le anima a ampliar su conocimiento sobre estos temas en JavaScript. Es fuera del alcance de este material de introducción a la cubierta en profundidad estas características adicionales. Capítulo 6 del libro de M. Haverbeke es una buena lectura sobre este tema.

responsable de la ubicación de la pantalla del objeto en el navegador. También, número entero versiones numéricos, izquierda y anchura, de los valores de cadena correspondientes en el objeto estilo se almacenan, a condiciones adecuadas de prueba en otras frases del código. los moverse() método actualiza la propiedad numérica izquierda y luego lo utiliza para la creación de la style.left propiedad del objeto DOM.

Una vez que una declaración de la clase está disponible, nuevos objetos del mismo tipo se pueden crear. Esta operación requiere el uso de la palabra clave **nuevo** .

```
clash.initWeapons = función () {
    dejar sh = documento .getElementById ( "Capshield" ); clash.shield = nuevo Arma
    (sh);
    dejar mj = documento .getElementById ( "Mjolnir" ); clash.mjolnir = nuevo Arma
  (mj); };
```

El método initWeapons () del objeto choque se encarga de la creación de estos objetos que representan las armas en el objeto choque y conectándolos con los objetos DOM que muestran las armas en la ventana del navegador.

6. Creación de nodos en el DOM

HTML DOM puede ser completamente editado, al cambiar, añadir o eliminar cualquier tipo de nodo. No sólo los elementos, sino también atributos o contenidos de los elementos. Cada nodo editada tiene que ser manejado a través de su nodo padre con el fin de preservar la integridad del árbol DOM.

Para añadir un nuevo elemento, el nodo elemento es creado primero y luego anexado al elemento principal seleccionado. ación Cre- de un elemento se lleva a cabo por el createElement () método de la documento objeto, mientras que el añadir Niño()

método del elemento padre se encarga de conectar el nuevo elemento. Las acciones necesarias son así:

```
dejar bg = documento .getElementById ( "campo de batalla" );
dejar div1 = documento .createElement ( "Div" ); bg.appendChild
(div1);
```

Adición de contenido textual a un nodo de elemento, ya sea nuevo o existente, debe seguir los mismos pasos que antes, pero usando el específica createTextNode () método de la documento
objeto. Estas acciones pueden escribirse así:

```
dejar LetterO = documento .createTextNode ( "O" ); div1.appendChild
(LetterO);
```

Por otra parte, para añadir un atributo a un nodo de elemento, ya sea nuevo o existente, una secuencia similar de acciones debe ser seguido, pero los detalles di ff er. Creación del atributo es realizada por el createAttribute () método de la documento
objeto, mientras que la específica setAttributeNode () método del elemento padre une el nuevo nodo atributo al nodo de elemento. Por otra parte, el valor del atributo tiene que ser asignado por medio de la valor propiedad del nodo de atributo. Estas acciones se ven así:

```
dejar att = documento .createAttribute ( "carné de identidad" );
att.value = "Capshield" ; div1.setAttributeNode (att);
```

Un montón de métodos y propiedades están disponibles para ING navigat-, administrar y editar los elementos DOM, atributos, contenidos y objetos de estilo. Familiarizarse con ellos le ayudará a desarrollar el código sea más funcional y precisa JS en HTML5 juegos.

Aprender más acerca de. . .

HTML DOM Referencia

Tu turno 4

Copiar todo el proyecto CapShieldVsMjolnir a uno nuevo llamado CapShieldVsMjolnirCreation.

A continuación, cierre todos los archivos desde el primer proyecto y abrir todos los del proyecto duplicado. Ahora en index.html, elimine el interior < div> etiquetas, identi fi cada como " capshield" y " mjolnir" , y todas sus tiendas de campaña con-. A continuación, en shieldhammer.js, introducir una llamada a initMyDOM (); como la primera frase de la definición de la función punto de entrada(), y luego declarar la función initMyDOM (). Esta función tiene que crear todos los elementos HTML, atributos y contenidos acaba de eliminar en index.html. Por último, pruebe el proyecto resultante para comprobar que ambos enfoques son funcionalmente equivalentes.

7 Animating en el *lona*

En la fi sesión de laboratorio primero que hemos aprendido cómo utilizar el lona elemento DOM, que proporciona una interfaz de programación para dibujar formas o incrustar imágenes en una imagen de trama zona. Cada elemento pintado en el lienzo se convierte inmediatamente en píxeles de colores en un mapa. Por lo tanto, la tela no retiene ninguna función conocida previamente de la forma o la imagen dibujada, tales como la posición o el contorno. Para manejar la información geométrica de los elementos dibujados en el lienzo, estos datos deben mantenerse separados, en una estructura de datos asociada en el código. Por lo tanto, si queríamos mover cualquier elemento dibujado en el lienzo, tendríamos que claro (eliminar) el área del lienzo que incluye el elemento y volver a dibujar en una nueva posición.

Para ilustrar el proceso de mostrar una animación en una <Canvas> elemento, vamos a sustituir a los dos < div> ements El- incluyendo las letras O y T, en nuestro proyecto de ejemplo, con dos imágenes que se dibuja en el < canvas> .

Tu turno 5

Copiar todo el proyecto CapShieldVsMjolnir a uno nuevo llamado CapShieldVsMjolnirC

A continuación, cierre todos los archivos desde el primer proyecto y abrir todos los del proyecto duplicado. Ahora en index.html, sustituir el código dentro de la <Body> elemento por la del fragmento de código primero se muestra a continuación, y en shieldhammer.js, sustitute la completa definición de la clash.initWeapons () y clash.advanceWeapons () métodos y de la Arma declaración de clase por los de los fragmentos de código restantes se muestra en esta sección. Por último, pruebe el proyecto resultante para ver la nueva animación que aparece en el navegador.

En index.html, el < div> elementos identificados por "Capshield" y " mjolnir" se sustituyen con una < canvas> elemento y se hace una carné de identidad atributo de modo que pueda acceder más tarde a partir del código JS.

```
<div id = "campo de batalla" >
    <canvas width = "800" = altura "600" id = "escena" >
        Su navegador no soporta HTML Canvas. Por favor, cambiar a
        otro navegador.
    </ Canvas> </
div>
```

Después, en el método initWeapons () del principal choque objeto, el coordinador de la animación, obtenemos el objeto que representa el < canvas> elemento de la DOM y, a partir de este elemento, el contexto 2D se requiere proporcionar una interfaz de dibujo. Este contexto se pasa a lo largo de cada constructor de armas con su correspondiente fi nombre de archivo de imagen para que pueda ser responsable de su propia muestra. Además, la posición inicial de su imagen en el < canvas> se pasa a cada constructor arma.

```
clash.initWeapons = función () {
    clash.cv = documento .getElementById ( "escena" );
    dejar CTX = clash.cv.getContext ( "2D" );
    dejar sh = "Images / CaptainAmericaShield.png" ; clash.shield = nuevo Arma
    (CTX, sh, 0 , 260 );
    dejar mj = "Images / ThorMjolnir.png" ; clash.mjolnir = nuevo Arma
    (CTX, mj, 651 , 250 );
};
```

Observan que dos objetos se crean a partir de la Arma clase: clash.shield y clash.mjolnir. Cada una almacena el contexto 2D y, también, las coordenadas izquierda y superior usada para colocar la imagen en la < canvas> . Un objeto ima- gen, almacenada en la propiedad imagen, es creado por la Imagen() constructor, que es funcionalmente equivalente a documento .createElement ('Img'). Este objeto está a cargo de mapa de bits que representa el arma asignada. Por lo tanto, su src puntos de atributo al expediente que contiene la imagen y su onload () método se codifica para ejecutar el primer dibujo del mapa de bits arma en el lienzo. Este método se ejecuta tan pronto como se carga el mapa de bits de imagen en la memoria principal (más detalles se dan en la siguiente sección).

Además, dentro de la onload () código, el anchura propiedad de la Arma objeto se establece desde el homenaje en- naturalWidth, que es leído de la imagen fi l. Esta asignación es necesaria si queremos que los ods met advanceWeapons () y changeBackgroungColor ()

del choque objeto para el trabajo todavía, ya anchura es una propiedad de Arma casos que se utiliza en ambos métodos. Sin embargo, conseguir esta asignación de trabajo, requiere primero la grabación en el Mis padres propiedad ad hoc de la imagen objetar una referencia por su padre Arma objeto y, a continuación, esperar a que el mapa de bits de imagen que se ha cargado completamente para tomar el e ff valor reflexivo de la naturalWidth propiedad.

Tenga en cuenta que la esta palabra clave que aparezca en el ámbito de la Arma declaración de la clase se refiere a un objeto ff on diferente que el esta de palabra clave escrita en el ámbito de la onload
método para la imagen objeto. Tómese su tiempo y analizar el código siguiente para Arma declaración de la clase, con el fin de captar todas las ideas que acaba de presentar.

```
clase arma {
    constructor (C2D, nombre de archivo, l, t) {
        esta . Canvas2D = C2D;
        esta . imagen = nuevo Imagen();
        esta . image.myparent = esta ;
        esta . image.src = nombre del archivo;
        esta . image.onload = función () {
            dejar metro = esta . Mis padres;
            m.canvas2d.drawImage ( esta , l, t);
            M.Width = esta . naturalWidth; };

        esta . izquierda = l;
        esta . parte superior = t;
        esta . anchura; };

    claro() {
        dejar l = esta . izquierda, t = esta . parte superior;
        dejar w = esta . image.naturalWidth;
        dejar h = esta . image.naturalHeight;
        esta . canvas2d.clearRect (l, t, w, h); };

    mover (n) {
        esta . izquierda + = norte;
        dejar yo = esta . imagen;
        dejar l = esta . izquierda;
        dejar t = esta . parte superior;
        esta . canvas2d.drawImage (i, l, t); }; };
```

Objetos Event

Un argumento se puede pasar a un evento manejador fun- ción: el objeto de evento, que proporciona la formación de in- adicional sobre el evento, como lo fueron las coordenadas del ratón o la tecla que se ha pulsado. Una gran cantidad de acciones de montacargas para requerir la gestión de objetos de eventos.

javascript.info Los objetos de evento

Una vez que los datos de imagen para Arma Los objetos son cargados, la animación se inicia y su moverse() y claro() métodos de hacer la imagen micro-movimientos. Estos métodos utilizan los métodos de lona drawImage () y clearRect () para ING pintura- y la limpieza de la tela, respectivamente. los moverse()

conjuntos de parámetros método hasta cuántos píxeles tiene cada arma para avanzar hacia la izquierda o hacia la derecha.

Leer por encima de los detalles del código de estos métodos, y por debajo de la secuencia de llamadas a ellos. Después de los movimientos se hacen, nuevas posiciones de armas se ponen a prueba para detectar su instantánea 'choque'. Cuando la prueba tiene éxito, la animación se detiene.

```
clash.advanceWeapons = función () {
    const SHIELD_MOVE = 1 ;
    const MJOLNIR_MOVE = -2 ; clash.shield.clear ();
    clash.mjolnir.clear (); clash.shield.move
    (SHIELD_MOVE); clash.mjolnir.move
    (MJOLNIR_MOVE);

    dejar sh = clash.shield, mj = clash.mjolnir;
    Si ( sh.left + sh.width / 2> mj.left)
        clearInterval (clash.timer); };
```

Tu turno 6

Cambiar el orden de las cuatro frases después de la const declaraciones de la clash.advanceWeapons () método, como este:

```
clash.shield.clear ();
clash.shield.move ( 1 );
clash.mjolnir.clear ();
clash.mjolnir.move ( -2 );
```

A continuación, ejecute de nuevo la animación y analizar las diferencias. ¿Puede usted explicar lo que le está pasando al fi nal de visualización de imágenes de armas?

Aprender más acerca de. . .

HTML5 Tutorial de la lona y Referencia

8 Eventos y su gestión

Un clic del usuario, un tiempo de espera del temporizador, o una pulsación de tecla son ejemplos de los muchos *eventos* que puede suceder en cualquier programa interactivo (ica graph-). En pocas palabras, un evento es "algo que

Propagación evento

Por defecto, un evento ocurring en un elemento DOM o lista de materiales se propagan a sus antepasados en el árbol (contenedores, en términos de etiquetas HTML), y todos ellos tienen una oportunidad para controlar el evento. comportamiento predeterminado de propagación puede ser modi fi cado o se detiene, y cada manejador propagado puede determinar dónde ocurrió realmente el caso.

javascript.info Burbujeante y capturando

sucede". A diferencia de la programación secuencial pura, donde está claro en que las cosas para producir, eventos en un entorno interactivo llegan en momentos arbitrarios. Por ejemplo, no sabemos cuando un usuario haga clic en un botón, pero cada vez que esto sucede, el programa debe responder adecuadamente a ese clic. Un paradigma de programación ff Erent di, *programación orientada a eventos,* que se requiere para desarrollar este tipo de programas. En la programación orientada a eventos, debemos decir básicamente dos cosas:

- cuyo caso nos interesa (*registro de eventos),*
  y
- cómo el programa debe responder a ese evento (*de fi n- ing el controlador de eventos).*

HTML DOM y lista de materiales en los navegadores proporcionan mecanismos para hacer frente a los dos requisitos, que permiten a los programadores de JS para registrar funciones como manejadores de eventos especí fi cos.

Dado un evento, tal como " carga" , "hacer clic" , "ratón sobre" o " pulsación de tecla" , cada elemento DOM y lista de materiales proporciona un atributo, llamado " onload" , "al hacer clic" , "el ratón por encima" o " onkeypress" , listos para el registro de una función de controlador para asistir al evento, cada vez que ocurre. Ya hemos presentado uno de los siguientes atributos:

```
ventana .onload = punto de entrada;
```

los " carga" evento en el ventana objeto es normalmente despedido cuando toda la página se ha cargado, incluyendo el contenido (imágenes, css, guiones, etc.). Por lo tanto, en nuestros proyectos CSS HTML / JS /, queremos que el código para empezar a correr cuando se genera este evento en el navegador.

atributos tales evento, en elementos DOM y lista de materiales, puede registrar sólo un controlador por nodo. Pero a veces más de un manejador tiene que estar registrado en un nodo dado. Para este fin, la addEventListener () método permite añadir cualquier número de manipuladores por nodo, y también está disponible para cada elemento de DOM y lista de materiales.

Después, los manipuladores registrado para un evento se pueden mover a través del método re- removeEventListener (), también disponible en todos los elementos DOM y lista de materiales. Se llama con argumentos similares a los utilizados en addEventListener (),
sino que requiere la función de controlador que tiene un nombre, el mismo nombre que debe utilizarse para registrarla para la fi vez primera.

El código siguiente muestra cómo iniciar y detener la animación que nos ocupa, en base a la detección cuando el puntero del ratón se encuentra sobre o fuera de la lona. En el punto de entrada() función, el mouseOverOutCanvas () método de choque objeto está dirigido a registrar manejadores de eventos " ratón sobre" y " mouseout" en el CV atributo (objeto de lona) de choque. Tenga en cuenta la di ff erencies cuando el el ratón por encima atributo o la addEventListener ()
se utiliza el método.

```
función punto de entrada() {
    initBackground (); clash.initWeapons ();
    clash.mouseOverOutCanvas ();

    const T = 1000/50 ; // 20 ms
    clash.timer = setInterval (clash.animate, T); };


clash.mouseOverOutCanvas = función () {
    clash.cv.onmouseover = función () {
        clash.advancing = cierto ;
    };
    clash.cv.addEventListener ( "Mouseout" ,
                                            función () {
        clash.advancing = falso ;
    }); };
```

Los controladores de estos eventos de ratón, sobre o fuera del lienzo, basta con establecer un atributo booleano de la choque objeto. Tiene que ser inicialmente definido en la declaración objeto y, en cualquier momento, se comprueba si o no permitir el avance armas.

```
clash.advancing = falso ;

clash.animate = función () {
    Si ( clash.advancing)
        clash.advanceWeapons ();
  clash.changeBackgroungColor (); };
```

Tu turno 7

En el expediente shieldhammer.js del proyecto CapShieldVsMjolnirCanvas, reemplazar el código de la función punto de entrada() y el método clash.animate () con el código mostrado anteriormente en esta sección. Además, introducir en el JS fi l del nuevo código de choque escrito más arriba: la declaración y initiali- zación de la avanzar la propiedad y la definición del método mouseOverOutCanvas (). Finalmente, probar el proyecto resultante para ver la interacción entre el registro de eventos y la manipulación y la animación.

9 ejemplo final: cambiar pantallas

Concluimos nuestra introducción a JS y HTML DOM, mostrando cómo tratar con varias pantallas. Este segundo proyecto también refuerza y se extiende ya cubierto contenidos tales como la animación lienzo o el control de eventos, aunque nuevos detalles merecen cierta atención.

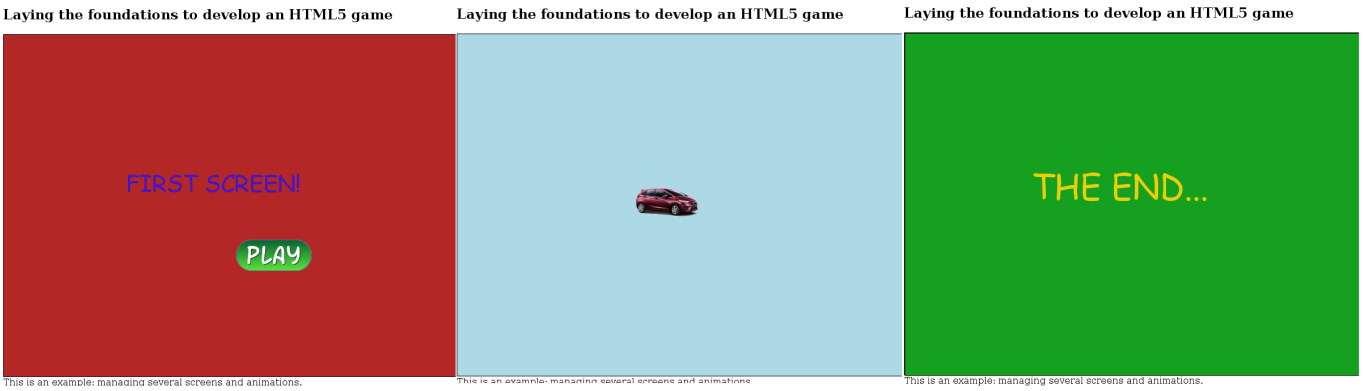

Figura 2: *ScreenCarTravel proyecto se ocupa de varias pantallas y anime un coche.*

Tu turno 8

Abrir el proyecto ScreenCarTravel y todos sus archivos. Ejecutar el proyecto y probar su funcionalidad antes de comprobar el código en el HTML, CSS y JS archivos. Al hacer clic en el ***Jugar*** botón, verá la secuencia de las pantallas se muestra en la figura 2 . Puede mover o parar el coche en la segunda pantalla haciendo clic sobre ella. Si el automóvil está en movimiento se detendrá; de lo contrario, se moverá. La tercera pantalla aparece cuando el coche se mueve más allá del límite derecho de la tela.

En index.html hay dos < div> elementos que se utilizan para "almacenar", tanto el inicio y el final pantallas -employing la identi fi ers gamestartscreen y endingscreen, respectivamente- y una < canvas> elemento para dibujar el coche y llevar a cabo la animación en la segunda pantalla.

```
<div id = "Gamecontainer" >
    <canvas width = "800" = altura "600"
        id = "GameCanvas" class = "Gamelayer" >
        Su navegador no soporta HTML5 Canvas. Por favor, cambiar a otro
        navegador.
    </ Canvas> <div id = "Gamestartscreen" class = "Gamelayer" >

        <img src = "Images / play.png" alt = "Jugar un juego"
            onclick = "myGame.showCanvas ()" >
    </ Div> <div id = "Endingscreen" class = "Gamelayer" >

    </ Div> </
div>
```

Tenga en cuenta que la misma clase ha sido de fi nida para todas estas etiquetas HTML - gamelayer - por lo que todos ellos se pueden manejar a la vez si es necesario. También, observar que una llamada a una myGame.showCanvas () método ha sido incluido en el al hacer clic atributo de la *Jugar* -una imagen del botón < img> etiqueta en HTML se ha utilizado.

La fi l screens.css establece las dimensiones del escenario, la posición de todas las pantallas, las imágenes de fondo, la posición de la *Jugar* botón, y el color de fondo del lienzo. Tenga en cuenta el uso del atributo cursor para especificar la forma que tomará el cursor del ratón cada vez que se coloca en la parte superior de la imagen.

En fi l carmoves.js, queremos llamar su atención sobre los métodos myGame.init (), myGame.showCanvas () y myGame.showEnd (), donde las transiciones entre pantallas son codificada por primera ocultar todos ellos y luego mostrando el apropiado. Esto se lleva a cabo por medio de un valor asignado a la style.display propiedad, ' ninguna' para ocultar y 'en línea' para mostrar.

El segundo punto de interés en este fi l es la "ratón hacia abajo" función de controlador. Tenga en cuenta el objeto de evento pasado y la conversión de la posición del puntero del ratón desde el navegador para coordenadas de lona.

El resto del código debe ser comprensible recordando todos los temas que hemos abordado a través de esta sesión de laboratorio.

referencias

Para preparar este documento hemos encontrado inspiración en, básicamente, estos libros:

- Marijn Haverbeke. *Elocuente JavaScript: Un Mod-ern Introducción a la programación,* Tercera edicion, https://creativecommons.org/licenses/by-nc/3.0/ , 2018
- David Flanagan. *JavaScript: Guía de la de fi nitiva,* Sexta Edición, O'Reilly Media, Inc., 2011
- Nicholas C. Zakas. *La comprensión de ECMAScript 6: Guía para la de fi nitiva JavaScript Desarrolladores,* Sin Almidón Press, Primera edición (septiembre 3, 2016) http://freecomputerbooks.com/Understanding- ECMAScript-6.html

downloadLinks ; Leanpub, 2015 Un buen lugar para empezar en JavaScript y HTML DOM es https://www.w3schools.com :

- JavaScript Tutorial
- JavaScript HTML DOM .
- JavaScript y HTML DOM Referencia .

10 Ejercicios

1. De fi ne una primera 800x600px < canvas> y luego un JS / - función de DOM para la elaboración de una *Pila de Poo emoji* similar a la mostrada en la Fig. 3 , Que se ha hecho desa- OpEd a través de métodos de lona. Su función debe:

Figura 3: *Pila de Poo emoji dibujado por una función JS / DOM en el lienzo.*

- declarar como parámetros: un contexto 2D lienzo, y *X* y *y* coordenadas de un punto de referencia para ING ° de dibujo;
- considerar (*x, y)* la parte superior punta de un cuadro delimitador de la imagen hacia la izquierda;
- la posición de todos los elementos de la imagen debe referirse a la *X* y *y* coordenadas, de tal manera que el cambio de los valores de *X* y *y* como resultado sólo en un cambio de la imagen en el lienzo.

2. De fi ne una animación para el *Pila de Poo emoji* en semejante forma en que (en realidad su cuadro delimitador error u omisión) hace un paseo desde la esquina inferior izquierda hasta la superior derecha del lienzo. Utilice la función desarrollada antes, con los argumentos adecuados, para hacer la animación.

3. Desarrollar un proyecto JS / DOM para implementar una *morir de seis pantallas* cuyas transiciones están restringidos a su adyacencia en la matriz y gestionado por presionando las cuatro teclas de flecha. También tener en cuenta las directrices siguientes:

- Combinar HTML, CSS y JS archivos para estructurar las acciones y la apariencia.
- De fi ne una 800x600px < div> antecedentes y otra an- < div> para cada una de las seis pantallas.
- En cada pantalla de incluir la imagen adecuada, entre las seis imágenes dadas: morir[123456] .Png.
- Coloque cada imagen en el centro de la escena y lo mostrará en anchura natural.
- De fi ne un controlador de eventos de teclado para la gestión de pulsaciones de teclas, tratar sólo con las cuatro teclas de flecha. este controlador *deben estar registrados* Para el documento objeto y debe comprobar la " keydown" evento.