3. HTML5 / CSS3 / JS: juegos en desarrollo

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

Duración estimada: 3 horas (+ 12 horas de trabajo en casa)

Ta aprendido hasta ahora: hemos desarrollado dos di ff Erent juegos simples para que ha llegado el momento de poner juntos todo lo que tiene de vista: el concepto del juego y los elementos empleados en su ejecución. El objetivo no es sólo para mostrar di ff maneras Erent de combinar las características de HTML5, CSS3 y Javascript para desarrollar un juego simple, pero también para animar a sus propias implementaciones para las versiones de estos juegos. En este sentido, aprenderá a insertar y animados contenidos visuales en el HTML5 *lona* elemento para gestionar eventos con los manipuladores, y dinámicamente cambiar el aspecto y el comportamiento de la página web que aloja su juego. Va a poner en práctica estos conceptos tanto en una en el corredor finito [1] - como juego y en un partido de fútbol-como mesa. ¡Ve a por ello!

Contenido

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

1 Más información sobre acontecimientos y su administración

UN folleto, 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 (gráfica). En pocas palabras, un evento es “algo que 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 o cuando se mueve el ratón fuera de un elemento dado, pero cada vez que esto sucede, el programa debe responder adecuadamente a ese clic o movimiento. Un paradigma de programación dife- rentes, *programación orientada a eventos,* que se requiere para desarrollar este tipo de programas. Recordemos que, en la programación orientada a eventos, que básicamente hay que decir 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).*

Terminamos la última sesión de laboratorio introducción de eventos y su manejo en JavaScript (JS). Ahora vamos a cubrir tanto los problemas a través de los proyectos previstos que están disponibles en el Aula Virtual. Sólo tiene que descargar el archivo 4students.zip correspondiente a esta sesión de laboratorio a su ordenador y descomprimirlo. Una vez descomprimido, se le encontrar una carpeta llamada 4students el cual tiene las carpetas de los dos juegos: *In- infinito Square* y *Futbolín.*

Tu turno 1

Abrir los dos proyectos en ***Código Visual Studio.*** Las instantáneas de ambos juegos se muestran en las Figuras 1 (página 8) y 2 (página 10) Respectivamente. Ejecutarlos y reproducirlos. Una vez que sepa ***qué*** ellos, vamos a aprender ***cómo*** ellas hacen

eso. Hemos de tener en la lectura de este folleto con las explicaciones oportunas.

1.1 Registro de eventos

Tome un vistazo al código almacenado en la fi les iftysquare.js y tablefootball.js y localizar todas las llamadas a la addEventListener () función. Se corresponden con registros de eventos -un evento se registra para un objetivo dado que será manejado por un controlador. Adivinar o averiguarlo [2]

cuyo caso cada llamada es para, y lo que representa cada uno ter paráme-. ¿Usted entiende qué evento *oyentes* hacer y lo que *¿escucha a?*[3] Ahora, completar esta ción estatal en general mediante el suministro adecuado *genérico* nombres de los elementos

MI - YO: manipulador, objetivo, y el evento.

```
MI. addEventListener ( E, i)
```

1.2 De fi nir el controlador de eventos

¿Cuántas manejadores de sucesos no tienen nuestros programas? Si tú dijiste *cinco,* usted está *más o menos*[4] Correcto. Vamos a echar un vistazo de cerca a uno de ellos, handlerOne en el expediente iftysquare.js.

```
ventana . documento .addEventListener ( "Keydown" , HandlerOne);
```

¿Cuál es el objetivo? En este caso lo es ventana . documento en la parte superior del árbol DOM. De todos modos, hay que saber que es típico para los programadores de JS para definir *anónimo* funciones a eventos mango, como en el caso de la *Futbolín* juego:

```
dejar pGround; [...] pGround = documento .querySelector ( "#patio de recreo" );
[...]


pGround.addEventListener ( "Mouseout" , función ( evento) {
        tableFootballData.isPaused = cierto ;
});
```

En este ejemplo, el objetivo es el elemento del árbol DOM que se identifica fi con la etiqueta “parque infantil”. En cualquier caso, si desea quitar o cambiar el manejador más adelante, tendrá que almacenar el manejador en una variable ya que tendrá que hacer referencia a ese controlador.

```
función handlerOne (evento) {
    cambiar ( event.keyCode) {
        caso RIGHTARROW_KEYCODE :
            Si ( ! rightArrowPressed) {
                    rightArrowPressed = cierto ;
                    theSquare.setSpeedX (SQUARE_SPEED_X); } descanso ;


        [...]}}
```

[2] https://www.w3schools.com/jsref/dom_obj_event.asp

[3] Esta y otras preguntas a continuación no están destinados a ser retórica. Eso es importante que hace una pausa en la lectura y pensar en ellos o incluso hallar una respuesta antes de reanudar su lectura. Por favor, no olvide tomar notas para que pueda revisarlas más adelante.

[4] Volveremos a este punto más adelante.

querySelector vs. getElemenById

Si sólo nos fijamos en la evaluación de ambos métodos devuelven el mismo: el elemento del DOM que coincide con lo dado ***selector*** - querySelector - o lo dado ***carné de identidad*** - getElemenById.

Sin embargo, hay di ff erences relación con el impuesto sin y con el apoyo del navegador. En este sentido, querySelector es más útil cuando se selec- tores complejos se van a emplear -por ejemplo si quisiéramos obtener el elemento de la lista primero que es un descendiente de un elemento que es un miembro de la foo clase fue posible emplear el selector. li foo en querySelector, que es algo que no es posible en getElemenById.

Además, hay personajes que tienen un signifi- cado especial para querySelector y no para getElemenById.

Por ejemplo, podría escribir algo así como `documento .getElementById ( "Vista: _id1" )` pero no con querySelector ya que el carácter “:” tiene un significado especial para este procedimiento y que tiene que ser “escapado”:

```
documento .querySelector ( "#View \\: _ ID1" )
```

Además, hay que tener en cuenta que Si utiliza las versiones de estos métodos que devuelven un ***nodo lista*** de elementos coincidentes en el árbol DOM - querySelectorAll y getElementsByClassName - el querySelectorAll método devolverá una ***estático*** Lista -es decir, compuesto por los elementos que se veri fi có el selector ***en este momento se llama el método*** - mientras que el segundo devolverá una ***En Vivo*** Lista -es decir, compuesto por los elementos que se veri fi có el selector ***cuando se utiliza (*** que puede ser di ff Erent desde el momento en que se llama el método). Se podía leer más detalles aquí [un] y allí [si].

[un] https://stackoverflow.com/questions/14377590/queryselector-y-queryselectorall-VS-getElementsByClassName-y-getElemenById

[si] http://www.whatabouthtml.com/difference-between-getElementById-y-queryselector-180

Los oyentes no son manipuladores

Hay un problema importante aclarar la notación. Hemos distinguido ***oyente*** y ***manipulador,*** pero usted hallar que muchos autores o programadores -o accidentalmente, nosotros mismos- no puede hacer una distinción importante. Cuidado con esto y no se confunda cuando lea acerca de esto.

Nuestra handlerOne debe realizar las acciones previstas para mover el cuadrado de la tela cada vez que se pulsa una tecla de flecha. Por lo tanto, tenemos que saber qué tecla que el usuario ha pulsado. Esto se encuentra a través de la evento parámetro. Por lo general, podemos acceder a la información de eventos relacionados -que clave que el usuario ha pulsado, donde ha hecho clic el usuario, cuando una imagen se ha cargado completamente. . .- a través de este tipo de parámetros prácticos relacionados con el evento. El navegador se encarga de transmitir estos datos al programa de JS para que el evento puede ser gestionado adecuadamente.

Del mismo modo, la handlerTwo controlador lleva a cabo las ac- ciones necesarias para detener el cuadrado de la tela cuando se suelta la tecla de flecha correspondiente.

Aprender más acerca de. . .

Eventos básicos

Aprender más acerca de. . .

Referencia evento

Observe que los controladores se define en el expediente tablefootball.js son, todos ellos, funciones anónimas. Estos controladores gestionan, respectivamente, los eventos ratón sobre

- Este evento se produce cuando el puntero se mueve sobre un ele- mento, o en uno de sus hijos-, mouseout -Este evento se produce cuando el usuario mueve el puntero del ratón de un elemento, o de una de sus niños-, y movimiento del ratón
- Este evento se produce cuando el puntero se mueve mientras está sobre un elemento. Recuérdalo hay muchos eventos DOM HTML posible [5].

manejo de eventos es un tema muy amplio

Los aspectos más importantes de eventos ya están cubiertos. Hay, sin embargo, muchos detalles involucrados en el proceso de registro, el envío y el manejo de eventos. evento distinto objetos teclado o el ratón -por ejemplo eventos- tener di ff propiedades Erent, aunque pueden compartir algunas de ellas. Algunas propiedades pueden incluso ser útil para propósitos de depuración. Averiguar qué evento objetos pueden ser relevantes para el desarrollo de juegos en red y tratar de entender qué datos se puede acceder desde sus propiedades. Ver Eventos de referencia cuadro de arriba.

Ahora, ¿por qué decimos que habría sido "más o menos" bien si usted ha dicho que hay *cinco* manipuladores en ambos programas cuando se le preguntó al comienzo de esta sección? Tal vez hay más controladores? La respuesta a esta última pregunta es *de hecho si.*

Para empezar, los puntos de entrada de los dos programas - funciones empezar juego() y en eso(), respectivamente- son invocados cuando el carga caso de la ventana elemento se activa:

```
ventana .onload = empezar juego;

ventana .onload = en eso;
```

Como se explica en la última hoja de laboratorio, ambas frases son *registrarse* manipuladores *en efecto*. Por lo tanto, en este momento, la respuesta correcta habría sido *Siete* no *cinco*. Sin embargo, esta respuesta sigue siendo simplemente *falso*. Sigue leyendo la siguiente sección y descubrirá que hay *cuatro* más manejadores de ambos programas, por un importe total de *once*.

1.3 Animación de un juego: los eventos del temporizador

Como ya saben, la animación es sólo una cuestión de dibujar un objeto, borrarlos, y el dibujo de nuevo a una nueva posición o con una mirada cambiada. La forma más común de manejar esto es, manteniendo una función de dibujo que es llamada varias veces por segundo. También es necesario para ejecutar funciones que se actualizan las variables del juego que dan cuenta del estado interno de los -positions juego del héroe, los enemigos, obstáculos. . .- a una determinada tasa fijada tiempo. Estas funciones son updateGame () en el *En la plaza fi nidad* juego, y gameloop () y hacer() en el *Futbolín* juego.

Por lo tanto, tenemos que llamar a ellos en varias ocasiones, a intervalos regulares, si es posible. La forma más sencilla de lograr este objetivo es utilizar el método de temporizador setInterval (). Esta función le indica al navegador que mantener llamar una función dada -fi primera PARAMETRIZACIÓN repetidamente a intervalos de tiempo fijos -segundo PARAMETRIZACIÓN hasta su contraparte, clearInterval (), se llama. Siempre que se necesite para detener una animación -es decir, cuando el juego está en pausa, o tiene ended-, utilizaremos

clearInterval ().

Como ya saben handout- de -ver última práctica, esta función establece una *evento de temporizador* que se activará cada momento dado. Por lo tanto, el método que se registra en la llamada a esta función es *en realidad un manipulador* para el evento de temporizador que es desencadenado cuando este lapso de tiempo ha pasado. La función setInterval () se utiliza en ambos programas para llamar a las fun- ciones destinadas a actualizar el estado interno de ambos juegos. Y dos de ellos son de hecho los manipuladores. Vamos a echar un vistazo al código de la *En la plaza fi nidad* juego:

```
clase GameArea {
    [...] de inicialización () {

        esta . canvas.width = GAME_AREA_WIDTH;
        esta . canvas.height = GAME_AREA_HEIGHT;
        esta . contexto = esta . canvas.getContext ( "2D" );
        dejar theDiv = documento .getElementById ( "Como se Juega" );
        theDiv.appendChild ( esta . lona);
        esta . intervalo = setInterval (updateGame,
                                        1000 / FPS);
        esta . número de cuadro = 0 ; } [...]


} [...]
```

```
función updateGame () {
    // Comprobar colisión para terminar el juego
    dejar colisión = falso ;
    para ( dejar yo = 0 ;yo < gameArea.obstacles.length; i ++ ) {
        Si ( theSquare.crashWith (gameArea.obstacles [i])) {
            colisión = cierto ;
            descanso ;
        }} Si ( colisión) {
            Final de partida (); } más
    {
        // Aumentar el recuento de los marcos
        gameArea.frameNumber + = 1 ;
        // Vamos a ver si se deben crear nuevos obstáculos
        Si ( gameArea.frameNumber > = FRAME_OBSTACLE)
            gameArea.frameNumber = 1 ;
        // Primero: verificación si el número dado de tramas // ha pasado

        Si ( gameArea.frameNumber == 1 ) {
            dejar oportunidad = Matemáticas .aleatorio();
            Si ( oportunidad < PROBABILITY_OBSTACLE) {
```

[5] https://www.w3schools.com/jsref/dom_obj_event.asp

```
dejar altura = Matemáticas .piso( Matemáticas .aleatorio() *
    (OBSTACLE_MAX_HEIGHT - OBSTACLE_MIN_HEIGHT
     + 1 ) + OBSTACLE_MIN_HEIGHT);
dejar brecha = Matemáticas .piso( Matemáticas .aleatorio() *
    (OBSTACLE_MAX_GAP - OBSTACLE_MIN_GAP + 1 ) +
     OBSTACLE_MIN_GAP);
dejar formar = nuevo SquaredForm (
     gameArea.canvas.width, 0 , OBSTACLE_WIDTH, altura,
  OBSTACLE_COLOR); form.setSpeedX ( - OBSTACLE_SPEED);
  gameArea.addObstacle (forma);

// El obstáculo en la parte inferior solamente se // creado si
hay suficiente espacio
Si (( altura + brecha + OBSTACLE_MIN_HEIGHT) <=
        gameArea.canvas.height) {formulario = nuevo
   SquaredForm (
         gameArea.canvas.width, altura + brecha, OBSTACLE_WIDTH,
  gameArea.canvas.height - altura - brecha, OBSTACLE_COLOR);
  form.setSpeedX ( - OBSTACLE_SPEED); gameArea.addObstacle
  (forma); }}} // Mover obstáculos y eliminar las que va // fuera de la lona
```

```
para ( dejar yo = gameArea.obstacles.length - 1 ;
          yo > = 0 ; yo - ) {
    gameArea.obstacles [i] .Llevar ();
    Si ( gameArea.obstacles [i] .x +
          OBSTACLE_WIDTH <= 0 ) {
      gameArea.removeObstacle (i); }} // Mover
  nuestro héroe

  theSquare.move ();
  // Nuestro héroe no puede ir fuera de la lona
  theSquare.setIntoArea (gameArea.canvas.width,
                gameArea.canvas.height);
  gameArea.clear ();
gameArea.render (); }}
```

El evento de temporizador se establece en el código de la initialise método de la GameArea clase.

```
esta . intervalo = setInterval (updateGame, 1000 / FPS);
```

Aprender más acerca de. . .

Ventana setInterval () Método

La función updateGame () se fija para ser llamada en el número de milisegundos necesaria para que el framer- dado comió -stored en FPS - puede lograrse. Como puedes ver, updateGame ():

1.- cheques para las colisiones entre la plaza y los obstá- culos. Si hay una colisión, el juego se terminó.

2.- En caso contrario, se decide al azar si se presentan nuevos obstáculos son creados con la altura y el tamaño al azar brecha entre ellos.

3.- A continuación, se mueve los obstáculos y elimina aquellas que se encuentran fuera de la lona.

4.- A continuación, se mueve a la plaza, teniendo cuidado de no ponerlo fuera del lienzo.

5.- Por último, se borra y se vuelve a dibujar el área del lienzo. Con respecto al código de la *Futbolín* juego, tenemos:

```
función en eso() {
  // intervalo establecido a la lógica llamada gameloop en 30 FPS
  tableFootballData.timer = setInterval (gameloop,
                                  1000-1030 );
  // vista representación
  tableFootballData.request =
          ventana .requestAnimationFrame (render);
  tableFootballData.isRendering = cierto ;
  tableFootballData.isPaused = falso ;
  // entradas
  handleMouseInputs (); }
```

```
función gameloop () {
  Si ( tableFootballData.isPaused) {
    Si ( tableFootballData.isRendering)
      ventana .cancelAnimationFrame
            (tableFootballData.request);
    tableFootballData.isRendering = falso ;
    regreso ;
  } else if ( ! tableFootballData.isRendering) {
    tableFootballData.request =
          ventana .requestAnimationFrame (render);
    tableFootballData.isRendering = cierto ;
  } MoveBall ();
 autoMovePaddleA (); }
```

De nuevo, setInterval () se emplea para establecer un temporizador que se traducirá en una velocidad de fotogramas de 30 fps. Esto se realiza en el interior del cuerpo de la en eso() función. El temporizador se almacena en la propiedad Temporizador del tableFootballData objeto:

```
tableFootballData.timer = setInterval (gameloop,
                                  1000-1030 );
```

La función gameloop () lleva a cabo di ff Erent acciones si el juego está en pausa o que tiene que ver la prestación de sus elementos di ff Erent. A continuación, se actualiza la posición de la bola [6] y la posición de remo del ordenador.

Hay un tercer prestador de servicios en el *En la plaza fi nidad* juego. En el interior del cuerpo de la empezar juego() la función se hace un llamado a setTimeout:

```
función empezar juego() {
  gameArea.initialise ();
  gameArea.render ();

  ventana . documento .addEventListener ( "Keydown" ,
                handlerOne);
  ventana . documento .addEventListener ( "tecla Arriba" ,
                handlerTwo);

  segundos = 0 ; se acabó el tiempo = ventana .setTimeout (updateChrono, 1000 );
  theChrono = documento .getElementById ( "Crono" );

}
```

[6] Y lo hace algunas otras cosas también, como veremos en la Sección 4 .

los setTimeout se asemeja a la función setInterval ya que también establece un evento de temporizador que será manejado por el ción fun- pasado como el parámetro de primera. El rencia esencial di ff entre ellos es que el temporizador ajustado por setTimeout se usa *sólo una vez*. Esto significa que el evento de temporizador se activa una sola vez y que el controlador se ejecuta sólo una vez. Si queremos que el manejador se ejecuta de nuevo se requiere a la llamada setTimeout de nuevo. Esto es algo que se hace de hecho dentro de la updateChrono función para que el temporizador se ajusta de nuevo una y otra vez.

```
función updateChrono () {
    Si ( Continua el juego) {
        segundos ++ ;
        dejar minutos = Matemáticas .floor (segundo / 60 );
        dejar secondsToShow = segundos 60% ; theChrono.innerHTML = CHRONO_MSG
        + "" +
                    Cuerda (Minutos) .padStart ( 2 , "0" ) +
                    ":" + Cuerda (SecondsToShow) .padStart ( 2 , "0" );
        se acabó el tiempo = ventana .setTimeout (updateChrono, 1000 );
    }}
```

Por lo tanto, esta función, que actualiza el cronómetro que mide el tiempo transcurrido en el juego, se llama varias veces cada segundo.

Aprender más acerca de. . .

Ventana setTimeout () Método

El último manipulador restante es quizás algo más oculto en el código de la *Futbolín* juego. Dentro de en eso() función, hay esta frase:

```
tableFootballData.request =
        ventana .requestAnimationFrame (render);
```

El código de la hacer() función:

```
función render () {
    renderBall ();
    renderPaddles ();
    tableFootballData.request =
            ventana .requestAnimationFrame (render);
    tableFootballData.isRendering = cierto ;
}
```

Esta función dibuja la pelota y después de las paletas gameloop () ha actualizado sus posiciones -bueno, de hecho, la *y* posición del remo del jugador se actualiza handleMouseInputs () - y, como gameloop (), se debe ejecutar repetidamente a intervalos de tiempo dados. Sin embargo, en lugar de utilizar setInterval (), en este caso hemos utilizado otra opción: la función requestAnimationFrame (). Este método tiene que ser llamado cada vez que queremos registrar una petición para animar a los contenidos de la ventana. Esta es la razón por la que se llama inicialmente en en eso() y luego en el hacer() propio código de función.

los requestAnimationFrame () función se utiliza hasta la fecha UP- la vista -el dibujo de la etapa- de acuerdo a los datos -como actualiza por las funciones gameloop () y handleMouseInputs (). Hemos utilizado este método para actualizar la vista porque la vista sólo necesita actualización, en un escenario óptimo, cuando el navegador decide que tiene que hacerlo. El intervalo de requestAnimationFrame ()

No se fi ja a priori: cuando el navegador es visible, requestAnimationFrame () se ejecutará a menudo. Cuando el terio de murciélago esta baja o el navegador se ejecuta en segundo plano, el navegador se ralentizará su frecuencia de ejecución.

De cualquier manera, cada vez que el navegador decide que ha llegado el momento de volver a dibujar la ventana se llamará al "controlador" se pasa como parámetro a requestAnimationFrame (). Tenemos citas usadas en sentido estricto, ya la hacer() función no es un manipulador, sino una *llamar de vuelta,* es decir, una función que se llamará cuando la ventana del navegador decide. En otras palabras, ningún evento se dispara.

Si quisiéramos cancelar una solicitud anterior usaríamos la cancelAnimationFrame () método. Este podría ser el caso si el juego se había puesto en pausa o terminado. Algunas de las ventajas de utilizar esta API en lugar de setInterval ()

son que el navegador puede hacer lo siguiente:

- Optimizar el código de animación en un único ciclo de re fl ujo repinte aftosa, lo que resulta en una animación más suave.
- Detener la animación cuando la pestaña del navegador no es visible, lo que lleva a una menor uso de CPU y GPU.
- reducir automáticamente la velocidad de fotogramas en máquinas que no soportan altas velocidades de cuadro, o, alternativamente, aumentar la velocidad de fotogramas en las máquinas que son capaces de un procesamiento más rápido. Oso

en mente ese nosotros son utilizando requestAnimationFrame () sólo en la lógica vista-relacionada. Sin embargo, empleamos setInterval () para actualizar los cálculos de datos juego porque *siempre hay que actualizar a intervalos de tiempo regulares* -ya que, por ejemplo, puede que hayamos perdido algunas colisiones de otra manera.

En general, la setInterval () función se prefiere realizar los cálculos de la lógica juego mientras requestAnimationFrame () se utiliza para la representación de vista.

Aprender más acerca de. . .

window.requestAnimationFrame ()

1.4 Extracción de los oyentes ya no son necesarios

Cada detector de eventos que tenemos memoria consume tiempo y procesa- miento. Por lo tanto, es habitual para eliminar los teners LIS- mediante el uso de removeEventListener () -con los mismos parámetros que el correspondiente addEventLis () - cuando sabemos que ya no se necesitan. Hacemos esto sólo en el *En la plaza fi nidad* juego por medio del Final de partida () función. Recuerde que esta función se llama en el interior updateGame () siempre que se detecte una colisión entre la plaza y un obstáculo, lo que conduce a la final de la partida.

```
función Final de partida () {
    Continua el juego = falso ;
    clearInterval (gameArea.interval);
    ventana . documento .removeEventListener ( "Keydown" ,
                        handlerOne);
    ventana . documento .removeEventListener ( "tecla Arriba" ,
                        handlerTwo);
}
```

Entre otras cosas, Endgame usos clearInterval para cancelar la llamada a updateGame establecido a través de la setInterval
método, previamente almacenado en la propiedad intervalo de
gameArea, y elimina los oyentes para los eventos de teclado keydown y tecla
Arriba. Observe que, al establecer el operador booleano Continua el juego a falso, sino que también evitará que el
updateChrono () la función de llamar a sí mismo una vez más por el voking in- setTimeout
método, rompiendo así el ciclo de llamadas a este oyente.

El detector de eventos único resto que ser eliminado es el de ventana .onload pero esta eliminación no es necesario, ya que se dispara *sólo una vez:* cuando la ventana del documento se ha cargado completamente en el navegador a primera.

2 Administración de objetos en lienzos

Una novedad importante en HTML5 es la lona. Este es un elemento diseñado para los programadores para poder (mediante programación) dibujar gráficos. El elemento canvas de HTML5 proporciona el contexto para el dibujo y los gráficos y formas reales se extienden mediante la API de dibujo JS. Sin embargo, hay una clave di ff rencia entre el uso de tela y otros elementos DOM HTML habituales, como < div >

o < img >: lienzo es una *inmediato* modo mientras DOM es una *retenido* modo.

Esto significa que nos describen el árbol DOM con elementos y atributos, y hace que el navegador y rastrea los objetos para nosotros, mientras que en la lona tenemos que manejar todos los atributos y haciendo nosotros mismos. El navegador no guarda la información de lo que nos basamos. Es sólo mantiene los datos de píxeles dibujados.

los *En la plaza fi nidad* juego utiliza un lienzo para representar la mayor parte de los elementos del juego. Este lienzo se crea programa- ticamente ya que no es un elemento ya presente en el documento HTML (por favor, revise cuidadosamente el contenido del expediente index.html).

El lienzo se almacena en una propiedad de la gameArea objeto. Sólo hay que ver el código de la clase GameArea 'S constructor:

```
clase GameArea {
    (constructor de lona, héroe, obstáculos) {
        esta . lona = lona;
        esta . héroe = héroe;
        esta . obstáculos = obstáculos;
        esta . contexto = nulo ;
        esta . intervalo = nulo ;
        esta . número de cuadro = indefinido ;
    } [...]}
```

La creación de la lona se lleva a cabo cuando la instancia de gameArea es generado:

```
dejar gameArea = nuevo GameArea (
        documento .createElement ( "lona" ), la plaza, []);
```

Por último, se inserta en el árbol DOM en el initialise método de la clase:

```
clase GameArea {
    [...]
    Inicializar () {
        [...]
        dejar theDiv = documento .getElementById ( "Como se Juega" );
        theDiv.appendChild ( esta . lona); [...]} [...]}
```

Hay dos tipos de elementos en el lienzo: la plaza y los obstáculos. Sin embargo, ambos pertenecen a la misma clase: SquaredForm. Por favor, revise cuidadosamente el código de esta clase.

Dado que estos elementos no se almacenan en el árbol DOM [7] tenemos que mantener una cuenta de ellos para poder prestar adecuadamente en el lienzo. Recuerde que el lienzo es como un mapa de bits: no hay ninguna pista se mantiene de lo que está dibujado en él. Es por eso que suele mantener los elementos que se pueden extraer en el lienzo en una especie de estructura de datos. En este caso, se almacenan en una variable y en una propiedad de la clase GameArea, respectivamente.

```
clase GameArea {
    (constructor de lona, héroe, obstáculos) {
        esta . lona = lona;
        esta . héroe = héroe;
        esta . obstáculos = obstáculos; [...]} [...]


} [...]

dejar la plaza = nuevo SquaredForm ( 0 , GAME_AREA_HEIGHT / 2 , SQUARE_SIZE,
            SQUARE_SIZE, SQUARE_COLOR);
dejar gameArea = nuevo GameArea (
            documento .createElement ( "lona" ), la plaza, []);
```

Nuevos obstáculos se crean cuando algunas condiciones se cumplen en el updateGame () función, cuyo código se muestra en la Sección 1.3 . Prestar atención a estas frases:

```
función updateGame () {
    [...]
        dejar oportunidad = Matemáticas .aleatorio();
        Si ( oportunidad < PROBABILITY_OBSTACLE) {
            dejar altura = Matemáticas .piso( Matemáticas .aleatorio() *
                (OBSTACLE_MAX_HEIGHT - OBSTACLE_MIN_HEIGHT
                + 1 ) + OBSTACLE_MIN_HEIGHT);
            dejar brecha = Matemáticas .piso( Matemáticas .aleatorio() *
                (OBSTACLE_MAX_GAP - OBSTACLE_MIN_GAP + 1 ) +
                OBSTACLE_MIN_GAP);
            dejar formar = nuevo SquaredForm (
                gameArea.canvas.width, 0 , OBSTACLE_WIDTH, altura,
            OBSTACLE_COLOR); form.setSpeedX ( - OBSTACLE_SPEED);
            gameArea.addObstacle (forma);

            // El obstáculo en la parte inferior solamente se // creado si
            hay suficiente espacio
            Si (( altura + brecha + OBSTACLE_MIN_HEIGHT) <=
                    gameArea.canvas.height) {formulario = nuevo SquaredForm
                (gameArea.canvas.width,
                        altura + brecha, OBSTACLE_WIDTH, gameArea.canvas.height -
                        altura - brecha, OBSTACLE_COLOR);
```

[7] Sólo el lienzo se considera un elemento del árbol DOM y por tanto, dictada por el navegador.

```
            form.setSpeedX ( - OBSTACLE_SPEED);
    gameArea.addObstacle (forma); [...]}
```

Como se puede ver, los obstáculos son creados con una altura aleatoria y al azar una separación entre ellos - brecha -por usando el nuevo operador en el SquaredForm clase. Entonces, se insertan en la matriz obstáculos, que es una propiedad de cualquier instancia de la GameArea clase, por medio de la addObstacle método de esta misma clase que, a su vez, utiliza el empujar Método del JavaScript de Formación clase:

```
addObstacle (obstáculo) {
  esta . obstacles.push (obstáculo); }
```

Los números aleatorios están en todas partes

Observe la forma en que nuestro programa crea los obstáculos con una altura aleatoria y una separación aleatoria BE- interpólelas. La función Matemáticas .aleatorio() devuelve un número fl punto flotante pseudo-aleatorio entre 0

(Inclusive) y 1 (exclusivo). Si quisiéramos devolver una ***entero*** de números aleatorios en un intervalo dado, digamos [mínimo máximo], escribiríamos algo como esto:

```
Matemáticas .piso( Matemáticas .aleatorio() * (máx - min 1 ) + min);
```

Por favor, entienda esto bien porque la generación de números dom ran- dentro de un rango dado es importante en muchos programas, juegos incluidos.

A continuación, se actualizan las posiciones de la plaza y los obstáculos a través de la moverse método de la SquaredForm clase. los moverse método actualiza el valor de la X y y Nates coordi- dentro de la tela simplemente teniendo en cuenta la velocidad tanto en el *X* y *Y* ejes. El valor de la velocidad correspondiente se establece por los métodos setSpeedX y setSpeedY. Tenga en cuenta que la plaza no puede quedar fuera del lienzo: la

setIntoArea Método viene al rescate para evitar esta circunstancia suceda.

```
función updateGame () {
  [...]
  para ( dejar yo = gameArea.obstacles.length - 1 ;
              yo > = 0 ; yo - ) {
          gameArea.obstacles [i] .Llevar ();
          Si ( gameArea.obstacles [i] .x +
                OBSTACLE_WIDTH <= 0 ) {
            gameArea.removeObstacle (i); }} // Mover
      nuestro héroe

    theSquare.move ();
    // Nuestro héroe no puede ir fuera de la lona
    theSquare.setIntoArea (gameArea.canvas.width,
                           gameArea.canvas.height);
  [...]}
```

Hay que notar que si un obstáculo se mueve más allá del límite izquierdo del lienzo, entonces se quitan de la matriz a través de la empalme método de la Formación clase.

Extracción de un elemento de una matriz en JS

La eliminación de un elemento de una matriz es una operación muy común en la informática. La clase gama de JS tiene dos métodos que se pueden utilizar en este sentido: rebanada y concat. Vamos a dar por sentado que conocemos el índice, yo, del elemento que desea eliminar de la matriz a. Por lo tanto, podríamos hacer algo como:

```
función eliminar (a, i) {
    regreso una rebanada( 0 , I) .concat (a.slice (i 1 ));
}
```

los rebanada método toma un índice de inicio y un índice final y devuelve una matriz que tiene sólo los elementos entre esos índices. El índice de inicio es inclusivo, el exclusivo índice final. Cuando no se da el índice final, rebanada tendrá todos los elementos después de que el índice de inicio. Cuerdas también tienen una rebanada método, que tiene una semejante e ff ect. los concat método se puede utilizar para arrays de pegamento juntos, similar a lo que el + operador hace por cuerdas. De todos modos, las anteriores declaraciones de funciones

una nueva matriz que es una copia de la matriz original con el elemento en el índice dado eliminado. Lo que si queríamos eliminar un elemento de una matriz ***en su lugar*** -ië cambiando la matriz en sí, no mediante la creación de una copia fi ed modi? En este caso, tenemos que utilizar el empalme método, como es el caso con el código de la removeObstacle método de la GameArea

clase. los empalme método se utiliza para cortar una pieza de una matriz. Se le da un índice y una serie de elementos, y ***muta*** la matriz, la eliminación de que muchos elementos después el índice dado. Si pasa argumentos adicionales a empalme, sus valores serán insertados en la matriz en la posición dada, en sustitución de los elementos eliminados.

Finalmente, el updateGame () la función es responsable de volver a dibujar el contenido del lienzo -es decir el cuadrado y los obstáculos. Con este fin, esta función se llama a los métodos claro() y hacer() del gameArea objeto.

```
función updateGame () {
  [...]
     gameArea.clear ();
  gameArea.render (); }}
```

los claro() método sólo llama a la clearRect () ción fun- de la API de canvas y borra toda la zona del lienzo. Para poder utilizar clearRect (), clear () necesita el contexto de la tela, pero recuerda que este contexto se almacena en la propiedad contexto del gameArea -ver objetar el código de la GameArea clase. En particular, ver el código de la Inicializar () método.

```
clase GameArea {
  [...] claro() {

      esta . context.clearRect ( 0 , 0 , esta . canvas.width,
            esta . canvas.height); }
```

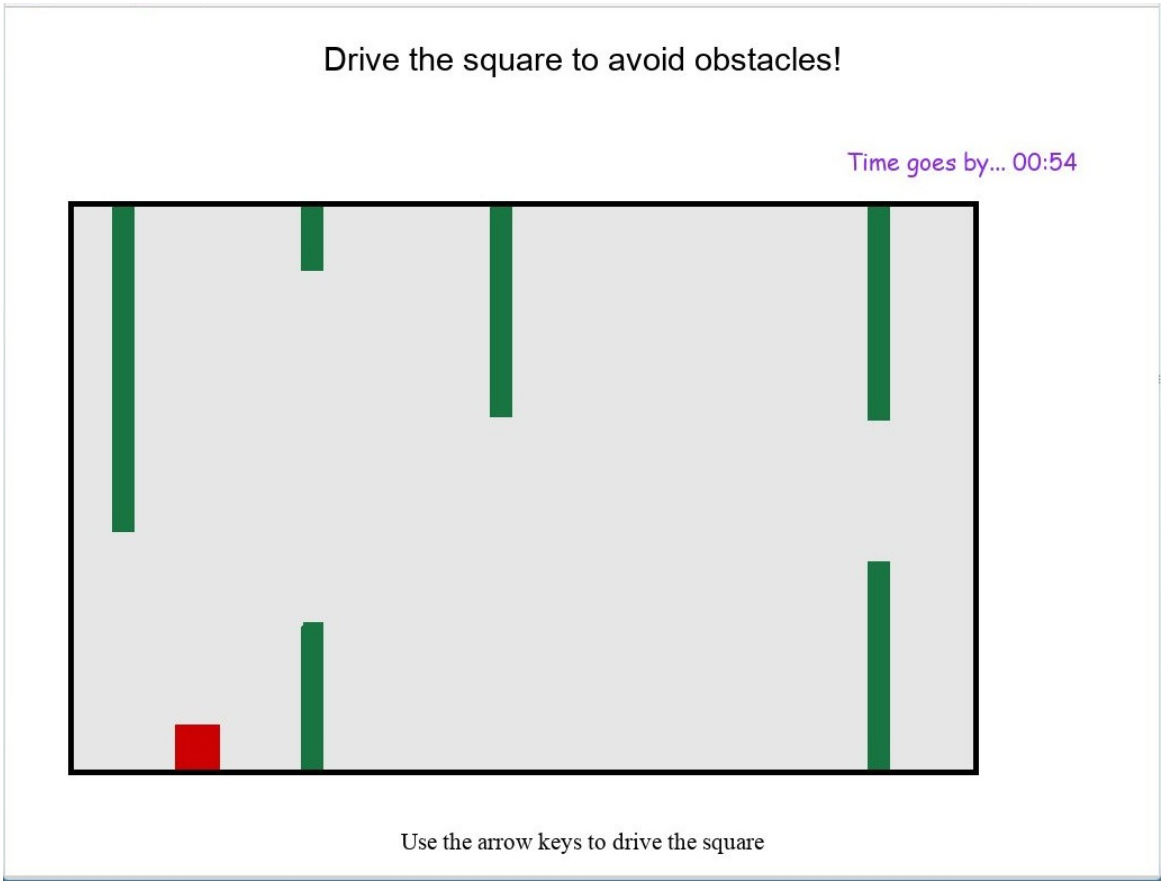

Figura 1: *los En la plaza fi nidad proyecto es un simple juego de fol lowing el estilo "infinito correr", que, por cierto, está de moda la derecha ahora.*

[...]}

los hacer() método llama simplemente el hacer()
Método de cada objeto que se dibuja en el lienzo. Recuerden que este es un método de la SquaredForm clase. Este método requiere, de nuevo, el contexto de la tela con el fin de ser capaz de utilizar los métodos fillStyle () y

fillRect (). Y, de nuevo, este contexto es proporcionada por el gameArea objeto -es se almacena en su contexto propiedad.

```
clase GameArea {
    [...]
    render () {
        para ( const obstáculo de esta . obstáculos) {
            obstacle.render ( esta . contexto); } esta . hero.render

    ( esta . contexto); } [...]}
```

Observamos que en este método se ha utilizado una entre las diversas variantes del para bucle en JavaScript: en este caso, la variante que nos permite recorrer todos los elementos de una matriz.

2.1 Limpieza de datos de hasta

Sólo hemos explicado cómo los obstáculos que se vuelven invisibles en el lienzo se retiran de la estructura de datos que los mantiene en el *En la plaza fi nidad* juego. En todo caso, usted debe saber que, a pesar de que no lo han utilizado, el Eliminar operador es capaz de tomar una referencia a un objeto, elemento o propiedad como un parámetro y la eliminación de su referencia de la memoria. Con sólo decir que porque se puede ver su uso en el código de algunos programas de JavaScript. Pero, por favor, ten en cuenta que este operador *sólo elimina la referencia,* no el objeto, elemento o propiedad en sí de la memoria.

Así que, ¿cuál es el punto en el uso de la Eliminar operador si en realidad estamos eliminando una referencia? El punto es que cuando un objeto, elemento o propiedad obtiene "huérfanos", sin ninguna referencia a ella, se marca como disponible y, por lo tanto, es un candidato para liberar a su memoria en la siguiente JavaScript de
recolección de basura ciclo.

El concepto de recolección de basura es un tema complejo que no cubren en este curso, pero es importante que usted es consciente de que para futuros desarrollos. La operación de recolección de basura de JavaScript está dirigido, por ejemplo, en esta URL [8].

Es una buena práctica para eliminar todas las referencias a objetos no utilizados o propiedades en los programas que puede desarrollar, ya que podría hacerse, por ejemplo, en el Final de partida ()
en función de la *En la plaza fi nidad* juego. Si quisiéramos para deshacerse de todas las referencias a objetos contenidos en una matriz de modo que todos ellos estaban marcados como desechables y, de esta manera, su memoria podría ser liberado por el recolector de basura, que simplemente podría asignar 0 a la longitud atributo de la matriz.

[8] https://developer.mozilla.org/es/docs/Web/JavaScript/Gestion_de_Memoria

Por ejemplo, podríamos hacer esto para la matriz que contiene los obstáculos en el juego: obstacles.length = 0 ;. La referencia a la propia matriz, sin embargo, se mantendría. Para perder esta referencia siempre podríamos asignar la nulo valor a la misma: obstáculos = nulo .

Tratando de resolver las pérdidas de memoria

Este artículo muestra cómo se pueden resolver los problemas de memoria en sus proyectos HTML + CSS + JS.

3 poner todas las piezas juntas:

El infinito Square!

En esta sección, vamos a describir el contenido de la *En la plaza fi nidad* juego, nuestro primer juego completo de HTML + CSS + JS. Una captura de pantalla del juego se muestra en la figura 1 . En primer lugar, nos centraremos en el documento HTML, que "los ejércitos" del juego: index.html.

Los dos elementos HTML que son relevantes para el juego son < pag carné de identidad = "Crono" > y < div camé de identidad = "Como se Juega" >.
El primero es utilizado para visualizar el tiempo transcurrido desde que el juego fue comenzado, mientras que el segundo tiene el lienzo donde se desarrolla el juego. Como mostramos en la Sección 2 , El lienzo se crea mediante programación y se inserta como un nodo secundario en el árbol DOM de esta < div >.

El aspecto de los elementos HTML ff Erent di se logra mediante el establecimiento de sus atributos en el iftysquare.css CSS fi l. Ahora, leer detenidamente el contenido de este archivo.

En este momento, el contenido de este CSS fi l no deberían sorprender. Lo interesante más que se dé cuenta es el hecho de que las propiedades establecidas para cada < lona > elemento se aplica no sólo a los lienzos ya existentes -presente en el documento- HTML, sino también a los lienzos que podrían ser creados mediante programación después. Por lo tanto, las propiedades establecidas para los lienzos se aplicarán a la lona creado por medio del código JS.

Por último, al final de index.html nos encontramos la etiqueta utilizan para cargar el código JS almacenada en el expediente iftysquare.js:

```
< guión src = "JS / iftysquare.js" > </ guión >
```

Recuerde que una vez que se ha cargado el documento, el empezar juego() la función se llevará a cabo:

```
ventana .onload = empezar juego;

función empezar juego() {
    gameArea.initialise ();
    gameArea.render ();
    ventana . documento .addEventListener ( "Keydown" ,
                                handlerOne);
    ventana . documento .addEventListener ( "tecla Arriba" ,
                                handlerTwo);
    segundos = 0 ; se acabó el tiempo = ventana .setTimeout (updateChrono, 1000 );
    theChrono = documento .getElementByld ( "Crono" );

}
```

Como se explicó en la sección anterior, a partir de este mo- mento en adelante, sólo las funciones handlerOne (evento), handlerTwo (evento), -tanto a mango eventos-teclado updateGame (), y updateChrono () -tanto ser de gestión cuando sus respectivos eventos del temporizador son triggered- se ejecutará en el programa.

3.1 Organizar el código

El código JavaScript en el expediente iftysquare.js está organizado de manera que la definición de las constantes es en la parte superior de la fi l (se acostumbra a escribir en mayúsculas):

```
const GAME_AREA_WIDTH = 800 ;
const GAME_AREA_HEIGHT = 500 ;
const SQUARE_SIZE = 40 ; [...]
```

Constantes en JavaScript

Desde ECMAScript 6, también conocido como ECMAScript 2015 (o ES2015 en corto), que es la última versión ampliamente implementado en JavaScript, por una nueva palabra clave y la sintaxis para la definición de las constantes se ha introducido:

```
const my_str = "Some_value" ;
const MY_NUM = 25 ;
```

Esta versión está hoy en día totalmente compatible con todos los navegadores modernos (***Chrome, Safari, Firefox,*** y ***Borde).***
Por el contrario, el código anterior no se ejecutará en ***Explorador de Internet.***

El valor de una constante no se puede cambiar (un error aumentará si se intenta cambiar el valor). Si se declara un objeto como si constante, es la referencia almacenada a este objeto que no se puede cambiar, ***pero el objeto en sí mismo puede ser cambiado.*** En otras palabras: se puede cambiar los valores de los atributos de este objeto, o incluso añadir nuevos atributos o eliminar existentes.

Tome un momento para comprobar el código y los valores de estas constantes para inferir qué significan sus nombres para. Tenga en cuenta que algunos de estos valores se pueden modificar para aumentar, por ejemplo, *la cultad di fi del juego.*

Evitar valores mágicos!

Has notado, para ejemplo, el uso de OBSTACLE_MIN_HEIGHT, OBSTACLE_MAX_HEIGHT, OBSTACLE_WIDTH, SQUARE_SIZE, SQUARE_COLOR o CHRONO_MSG? Hemos afirmado en nuestro programa que son constantes los números o cadenas que representan di ff Erent "conceptos" del juego. Por ejemplo, SQUARE_COLOR representa el código de color mal hexadeci- que el cuadrado tendrá en el juego. Podríamos haber utilizado la cadena literal cuando llenando la plaza, pero en general es una prác- tica mejor programación para evitar el uso de la llamada **los valores mágicos** y utilizar constantes. Proporcionan auto-documentación y facilitar el mantenimiento del software. Las razones detrás de estos beneficios son evidentes, ¿verdad?

Tras la declaración de constantes, hemos escrito el código para el SquaredForm clase, que inicializa sus propiedades en el constructor. La plaza y los obstáculos en el juego son instancias de esta clase, creados por medio de la nuevo operador.

A continuación, tenemos la declaración de la GameArea clase, y la declaración e inicialización de variables "globales", que incluye la creación de la gameArea objeto. Entonces, después de la ventana .onload frase, que de fi ne las funciones para manejar los eventos del teclado. Por último, se definen las funciones de ING requerida restante para que el juego pueda trabajar. Prestar especial atención a las funciones UpdateGame (), el cual pe- riódicamente se ejecuta la actualización a renderizar bucle necesaria para cada animación para el trabajo, y updateChrono (), que pasa cada segundo para actualizar el cronómetro del juego.

Divide y conquistaras

El nuevo trazado de la tela comprende dos tareas principales: despejar el lienzo y hacer que todos sus elementos en él. Estos dos di ff Erent tareas se han separado en dos di ff métodos Erent de la GameArea clase:

claro() y hacer(). Del mismo modo, necesitamos una función,

Final de partida (), para ejecutar código que limpia fi acabados del juego, tales como la eliminación de los detectores de eventos, restablecer las variables del juego, mostrando una pantalla de finalización, etc.

Esto hace que el programa sea más modular y fácil de mantener. La modularidad ayuda a la reutilización de código y aumento de la productividad. Por ejemplo, imaginemos que nos gustaría borrar el lienzo cuando el juego se termina. O simplemente realizar una transición sin problemas a una pantalla de "juego final". Entonces, fácilmente podríamos llamar gameArea.clear ()

o simplemente cambiar el código de Final de partida () Para lograr estos objetivos. ¿No es esto una ventaja? ¡Sí lo es! Pero aún podemos ir más allá: ahora imaginamos que decidimos borrar sólo una parte de la tela (por ejemplo, queremos salir de la esquina superior derecha del lienzo sin tocar). Entonces, sólo se debe modificar el claro() método de la

GameArea clase. Sin ella, tendríamos el mismo fragmento de código repite y sería una pesadilla para localizar todo el código repetido en varias ocasiones y la edición de los mismos cambios.

3.2 algunos cambios para mejorar este juego

Tu turno 2

1.- Reemplazar la plaza con la imagen de una nave espacial. El funcionamiento del juego seguirá siendo el mismo.
Insinuación: dibujar y animar la imagen en el lienzo de una manera similar a la que se empleó en el proyecto
ScreenCarTravel dibujar y mover la imagen de un coche (ver sesión de laboratorio anterior).

2.- Añadir nuevos obstáculos para el juego. Serán pequeños cuadrados negros en movimiento. Cuando los colisiona la nave con uno de ellos, el pequeño cuadrado negro se eliminarán de la tela y el jugador será penalizado restando 15 segundos del crono. Ajuste la velocidad de estos pequeños cuadrados para ser un valor aleatorio en un intervalo dado de valores. Tenga en cuenta que el cronómetro no puede mostrar valores negativos.

3.- Hacer el jugador tiene tres naves ("vida"). El juego termina cuando el jugador le ha estrellado tres barcos. Tiene que haber un elemento HTML en la página web que muestra el número de las naves restantes todo el tiempo. ¡Ser creativo!

4.- En la actualidad, el juego termina abruptamente. Diseñar una pantalla de fi nal que se mostrará cuando el juego termina. Esta pantalla fi nal debe incluir, al menos, transcurrido una imagen y un mensaje que muestra el tiempo total en el juego.

5.- Diseñar una pantalla inicial que permite al jugador elegir el cultad di fi del juego: fácil, normal y difícil. Usted tendrá que cambiar más adelante algunos valores de las constantes declaradas en el juego de acuerdo a la elección del jugador (por ejemplo, el número y la velocidad de los obstáculos). **Insinuación:** Crear botones de radio personalizadas un para conseguir la elección del jugador.

un https://www.w3schools.com/howto/howto_css_custom_checkbox.asp

4 Futbolín: un juego hecho sin un lienzo

Nuestro segundo ejemplo de proyecto es un juego similar a un juego de futbolín en el que el usuario juega contra la computadora a través de la entrada del ratón. Figura 2 muestra una captura de pantalla del juego.

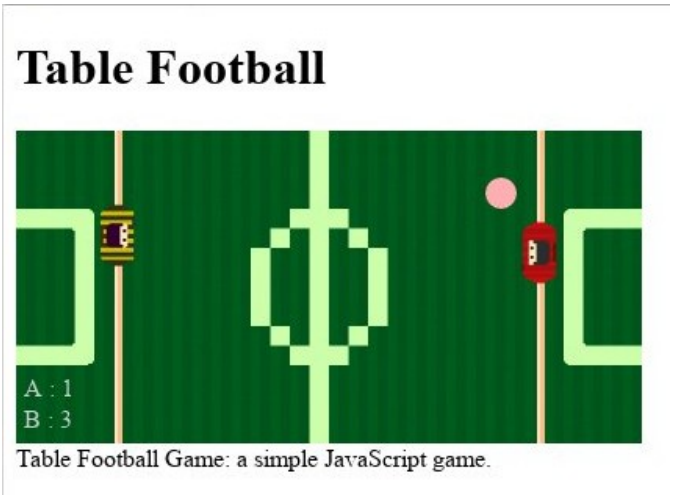

Figura 2: *Una captura de pantalla del juego de l Tabla Footbal.*

En el index.html fi l -el documento HTML en el que el juego es desempeñado, se quiere hallar < div > etiquetas para cada elemento en el juego: la paleta-manos, las paletas, la bola, y las puntuaciones -por favor, compruebe con la figura 2 . Observe que el < div > etiqueta de juego se divide en dos más < div > elementos: el " patio de recreo" y el " marcador". Se agrupan los elementos correspondientes: la pelota, las paletas, y sus manos, por un lado, y las puntuaciones en el otro. Tenga en cuenta que *No* lienzo se ha empleado en este juego. Esto podría ser considerado como una alternativa: usted no está obligado a utilizar siempre un lienzo, aunque es muy probable que usted tendrá que emplear uno, especialmente si usted tiene que dibujar imágenes y borrado de forma dinámica. Sin embargo,

en este juego todas las imágenes se cargan y se estiran antes de la animación -los real comienza Game-.

```
< div camé de identidad = "juego" >
    <! - elementos de juego para estar aquí ->
    < div camé de identidad = "patio de recreo" > < div clase = "Derecho de paletas a
        mano" > </ div >
        < div clase = "Paleta de mano izquierda" > </ div >
        < div camé de identidad = "PaddleA" clase = "paleta" > </ div >
        < div camé de identidad = "PaddleB" clase = "paleta" > </ div >
        < div camé de identidad = "pelota" > </ div >
    </ div >
    < div camé de identidad = "marcador" > < div clase = "Puntuación" > R: < lapso camé de identidad = "Puntuación
        -un" > 0
                                                </ lapso > </ div >
        < div clase = "Puntuación" > B: < lapso camé de identidad = "Puntuación-b" > 0
                                                </ lapso > </ div >
    </ div >
</ div >
```

4.1 Estructura del proyecto

Como es habitual, se carga el documento HTML dos externos archivos: la tableFootball.css CSS fi l y la tablefootball.js JavaScript fi l:

```
< cabeza >
    [...] < título > Tabla partido de fútbol </ título >

    < enlace rel = "Hoja de estilo" href = "Css / tableFootball.css" >
</ cabeza >
< cuerpo >
    [...] < guión src = "JS / tablefootball.js" > </ guión >

</ cuerpo >
```

Las imágenes que se utilizan para dibujar el patio de recreo, la paleta a mano y tanto el jugador de la izquierda y la derecha el jugador se almacenan en la IMG carpeta. Tener esto en cuenta cuando se echa las rutas utilizadas en el CSS fi l para cargar las imágenes.

Los estilos de fi nido en el comienzo de la CSS fi l nos permiten establecer la anchura y la altura del juego y su posición, las dimensiones de la zona de juegos -set a fi cio de la anchura y la altura de la de su imagen de fondo juego-, su posición, y la forma del cursor del ratón sobre él. Además, tenemos de fi ne los atributos de la pelota: la ubicación inicial, color de fondo, y el radio. Ver los estilos de fi nido para el # juego , # patio de recreo , y # pelota identi fi ERS.

Es particularmente interesante observar la forma en que los tributos At-de las paletas -y sus manos, como bien se han establecido. Las propiedades que son comunes a ambas paletas

- anchura y la altura, la distancia desde la parte superior, y tamaño- de fondo se han establecido empleando sólo el *etiquetar de la clase* de fi nido:. paleta . A continuación, las propiedades fi espec cas del cojín de las DLE -su del fondo de la imagen y la posición: izquierda o derecha se ajusta con el fi cación ERS de su correspondiente < div > elementos: # paddleA y # paddleB .

Un enfoque similar se ha seguido para establecer los atributos de las levas de las manos (véase la clase. paddle a mano) excepto que hemos utilizado dos "subclases", por lo que somos capaces de discriminar entre la mano izquierda y la mano derecha:

. izquierda . paddle a mano y. Derecha . paddle a mano . Por último, hemos fijado la posición y el color del marcador.

Al comienzo de la JS fi l hemos utilizado un objeto constante, tableFootballData, que es en realidad una colección de varios objetos que representan los elementos ff on diferente de paletas del juego -la, el parque infantil, el balón, y los scores- y grupales sus propiedades.

> **tableFootballData es un objeto**
>
> Es un objeto ya que tiene sus propias propiedades -y también podría tener su métodos-, pero no declarar una clase ni de fi ne un constructor. Es el único objeto de su tipo desde que no hay nuevos objetos se pueden crear instancias de ella.

Como es habitual, la ejecución del código se inicia cuando el documento HTML se ha cargado completamente en la ventana del navegador (ventana .onload) llamando al en eso() función. De hecho, este programa no termina nunca: se ejecutará hasta que el navegador carga otra página, o en la pestaña / ventana está cerrada.

```
ventana .onload = en eso;

función en eso() {
    // intervalo establecido a la lógica llamada gameloop en 30 FPS
    tableFootballData.timer = setInterval (gameloop,
                                                1000-1030 );
    // vista representación
    tableFootballData.request =
        ventana .requestAnimationFrame (render);
    tableFootballData.isRendering = cierto ;
    tableFootballData.isPaused = falso ;
    // entradas
    handleMouseInputs (); }
```

los handleMouseInputs () función permite a manejar los eventos del ratón para mover la pala sobre la base de la posición del ratón. Hemos rastreado entrar en el ratón y eventos de ratón de licencia por lo que somos capaces de iniciar y detener el juego. Recuerde que la gameloop () función se ejecutará aproximadamente 30 veces por segundo. Esto fue explicado en las Secciones 1.2 y 1.3 . También, observar que la referencia a la < div > elemento etiquetado con el # patio de recreo identi fi cador se almacena en una variable global: pGround.

También hemos utilizado el desplazamiento del ratón para obtener la posición del ratón y actualizar la posición de paletas del jugador en base a la posición del ratón en la sección de juegos. Tenemos que conseguir la *y* posición del cursor sobre la base de la parte superior del parque infantil izquierda borde. Sin embargo, el valor de *y* en el evento de ratón se refiere a la posición del cursor del ratón desde el borde superior izquierdo de la página. Por lo tanto, restamos la posición del campo de juego.

```
const tableFootballData = { [...] patio de recreo : {offsetTop : documento .getElementById
( "patio de recreo" ).
                                getBoundingClientRect (). top,
altura : parseInt ( documento .getElementById ( "patio de recreo" ).
                                clientHeight),
anchura : parseInt ( documento .getElementById ( "patio de recreo" ).
                                clientWidth)
}, [...]

función handleMouseInputs () {
    [...]
    // Calcular la posición de paletas mediante el uso de la
```

```
// posición del ratón.
pGround.addEventListener ( "movimiento del ratón" ,
            función ( evento) {tableFootballData.paddleB.y = event.pageY -
        tableFootballData.playground.offsetTop; }); }
```

El método getBoundingClientRect () devuelve las coordenadas del cuadro delimitador de su elemento, siendo parte superior la distancia entre la parte superior de la página y la parte superior de la caja de delimitación. Por lo tanto, en offsetTop almacenamos el valor de conjunto o FF del patio de arriba. Actualizamos la década de dle PAD *y* valor mediante el uso de la *y* posición del cursor del ratón. Este valor con el tiempo se reflejar en la pantalla cuando la vista de paleta se actualiza en la función de hacer que durante el redibujado navegador. Recuerde que, en este proyecto, hemos dividido la actualización de los movimientos de las entidades dentro del juego -la "lógica" - y sus dibujos en dos funciones de tasa SEPA: gameloop () y hacer() como se explica en la Sección 1.3 .

Comprender el modelo de caja CSS

Lea esta explicación sobre el modelo de caja CSS para que pueda entender las pro- piedades empleadas en los cálculos anteriores. Esta artículo corto lo explica muy bien. este otro artículo explica aún mejor.

La primera uno - gameloop () - anima tanto la pelota y el ordenador es de paddle. Para actualizar la posición de la pelota simplemente considera su dirección y velocidad. Tiene que comprobar las colisiones con los límites de la zona de juegos y con las paletas para tener debidamente en cuenta los cambios de dirección. Además, si la bola se mueve más allá de los límites de izquierda o derecha de la zona de juegos que van a llamar a las funciones que actualizan la puntuación correspondiente. remo de la computadora se mueve siguiendo la dirección de la bola en el *Y* eje, pero con una velocidad ligeramente menor [9] .

Por último, queremos llamar su atención sobre el código de las funciones que hacen que la bola, las paletas, y actualizar las puntuaciones, respectivamente. Estamos utilizando la API de DOM en las funciones que hacen que las paletas y la bola para cambiar dinámicamente las propiedades de fi nida en el CSS fi l. Esto es posible debido a que la estilo propiedad nos permite acceder a las propiedades CSS y cambiar sus valores. Por lo tanto, estas frases cambian el valor de la CSS izquierda y parte superior

propiedades de la # pelota selector.

```
dejar drawnBall = documento .querySelector ( "#pelota" ); drawnBall.style.left = (ball.x
+ ball.speed *
        ball.directionX) + "Px" ; drawnBall.style.top = (excesivamente + ball.speed
*
        ball.directionY) + "Px" ;
```

Y estas frases cambian el valor de la CSS parte superior propiedad tanto de la # paddleA y el # paddleB selectores.

```
dejar drawnPaddleA = documento .querySelector ( "#PaddleA" );
dejar drawnPaddleB = documento .querySelector ( "#PaddleB" ); drawnPaddleB.style.top = tableFootballData.paddleB.y
        + "Px" ; drawnPaddleA.style.top = tableFootballData.paddleA.y
        + "Px" ;
```

Para establecer las nuevas cadenas que se mostrarán como los valores de las puntuaciones que han utilizado el innerHtml propiedad.

```
función playerAWin () {
    dejar scorePlayerA = documento .
            getElemenById ( "Puntuación -un" );
    // el jugador B pierde.
    tableFootballData.scoreA + = 1 ; scorePlayerA.innerHTML = tableFootballData.scoreA.
        Encadenar(); [...]}

función playerBWin () {
    dejar scorePlayerB = documento .
            getElemenById ( "Puntuación-b" );
    // Un jugador pierde.
    tableFootballData.scoreB + = 1 ; scorePlayerB.innerHTML = tableFootballData.scoreB.
        Encadenar(); [...]}
```

4.2 ¿Qué es CSS para?

En esta sección vamos a tratar de resumir el papel que CSS3 archivos de juego en los juegos.

- En primer lugar, los archivos CSS vamos definen el aspecto del juego y la posición de las propiedades básicas -por ejemplo, el tamaño. . .- de los elementos en el juego. En este sentido, esto es como declarar propiedades "estáticos" que se aplicarán a todos los elementos en el juego.
- En segundo lugar, hemos aprendido que estas propiedades también se aplicarán a los elementos creados dinámicamente. Es decir, los valores de las propiedades CSS no sólo se aplican a los elementos ya existentes en los documentos HTML que conforman el juego, sino también a los elementos que se podrían crear mediante programación en el futuro.
- Por último, podemos mediante programación acceso y cambiar los valores de las propiedades CSS a través de la API de DOM. Tenga en cuenta que estos cambios se aplican inmediatamente a los elementos del juego.

4.3 algunos cambios para mejorar este juego

Tu turno 3

1.- El código de este juego está plagado de los llamados números mágicos. De fi ne las constantes que usted piensa que son necesarios para resolver este problema. darles nombres adecuados.

2.- Hacer el juego termina cuando se ha anotado un total de nueve goles. Tiene que haber un elemento HTML en la página web que mostrará las bolas res- tantes que se jugará todo el tiempo. **Insinuación:** sugerimos que se agrega un < div > elemento index.html para colocar las bolas y escribir un bucle para

[9] Sería imposible que el jugador a batir la computadora otro- sabio.

programáticamente añadir a la misma las nueve bolas iniciales como sus hijos por medio de la API DOM. Cada bola será, a su vez, a < div > elemento (similar a la de la bola usada en el juego). Todos estos < div > elementos pertenecerán a la misma clase (digamos ballRemaining). Crear un estilo en el CSS fi l para esta clase de modo que todas las bolas restantes tienen el mismo aspecto. A continuación, puede utilizar estas instrucciones en el bucle:

```
[...]
dejar elemento = documento .createElement ( "Div" );
dejar att = documento .createAttribute ( "clase" ); att.value = "BallRemaining"
; element.setAttributeNode (att); [...]
```

Por último, retire el último hijo de la < div > elemento usado para colocar las bolas restantes cada vez que se marca un gol (no importa quién las puntuaciones).

3.- Hacer el patio grande -dos veces el tamaño original, lo menos- y dibujar tres levas de manos con, respectivamente, dos, dos, y un jugador de fútbol. Escribir el código para mover todas las levas de manos en consecuencia. **Insinuación:** le sugerimos que, en primer lugar, comente la < guión > línea en el código HTML fi l. De esta manera, usted será capaz de centrarse sólo en la apariencia deseada de la página. Una vez logrado, se puede escribir el código y elimine el comentario < guión > la línea para probar el programa.

4.- Coloque el marcador en la parte superior derecha de la ventana del navegador. Vuelva a colocar las etiquetas “A” y “B” con “PC” y “jugador”, respectivamente. Además, cuando el usuario hace clic en el marcador, un dow-ganar preguntando por el nombre del equipo del jugador aparecerá. La etiqueta de la puntuación del jugador debe ser sustituido por la cadena escrita por el usuario (no se permite que la cadena vacía y dejará la etiqueta del reproductor sin cambios). **Insinuación:** utilizar el rápido

función para pedir el nombre de equipo del jugador.

5.- Diseñar una pantalla de fi nal que se mostrará cuando el juego termina. Esta pantalla fi nal debe incluir una imagen y un mensaje que indica el marcador fi nal, por lo menos. ¡Ser creativo! No se olvide de memoria libre y retirar los oyentes ya no son necesarios.

Un tutorial 4.4 HTML5 lienzo para principiantes

Tu turno 4

Hay un interesante tutorial en YouTube. Este tu- toría muestra cómo manejar el lienzo (dibujar objetos, mover y animar a ellos, etc.) a través de tres di ff Erent episodios:

- Episodio uno .
- episodio Dos .
- episodio Tres .

Trata de seguir en su tiempo libre!

referencias

Para preparar este documento hemos encontrado inspiración principalmente en 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
- Makzan, *HTML5 desarrollo del juego con el ejemplo. Guía para principiantes.* Segunda edición, editorial Packt, 2015.