Aprenda CreateJS construyendo un juego de Pong HTML5

by Daniel Albu 26 Jun 2012

Difficulty: Beginner Length: Long Languages: Español

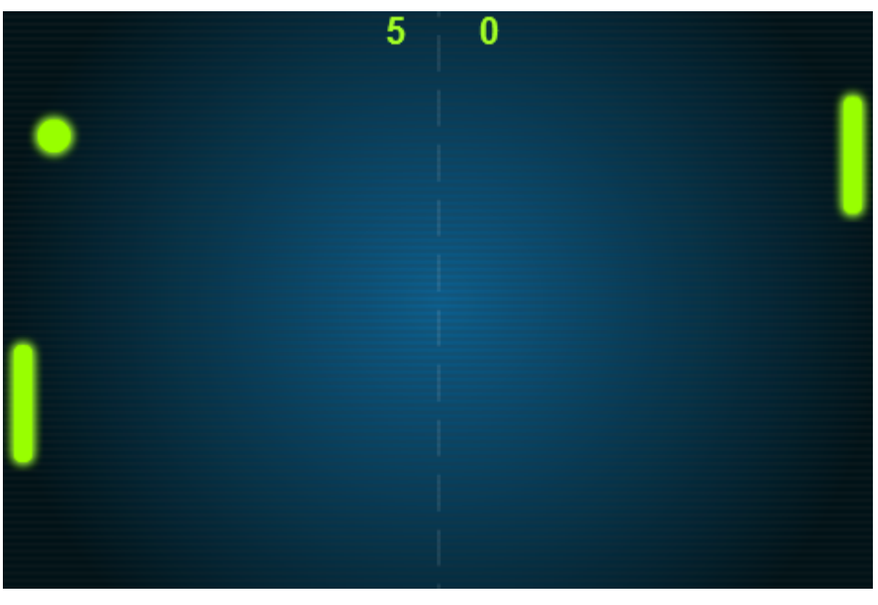
Games HTML5 Flash

Spanish (Español) translation by Juan Pablo Diaz Cuartas (you can also view the original English article)

La web se mueve rápido, ¡tan rápido que nuestro tutorial original de EaselJS ya está desactualizado! En este tutorial, aprenderá a usar el nuevo conjunto de aplicaciones CreateJS creando un clon de Pong simple.

Vista previa del resultado final

Echemos un vistazo al resultado final para el que trabajaremos:



Dele "click" para jugar

Este tutorial se basa en el juego Create a Pong de Carlos Yanez en HTML5 con EaselJS, que a su vez se basa en su guía Getting Started With EaselJS. Los gráficos y efectos de sonido están tomados del tutorial anterior.

Paso 1: Crea `index.html`

Este será nuestro archivo `index.html` principal:

```
<!DOCTYPE html>
<html>
  <head>
        <title>Pong</title>

        <style>/* Removes Mobile Highlight */ *{-webkit-tap-highlight-color: rgba(0, 0,

        <script src="http://code.createjs.com/easeljs-0.4.2.min.js"></script>
        <script src="http://code.createjs.com/tweenjs-0.2.0.min.js"></script>
        <script src="http://code.createjs.com/soundjs-0.2.0.min.js"></script>
        <script src="http://code.createjs.com/preloadjs-0.1.0.min.js"></script>
        <script src="http://code.createjs.com/movieclip-0.4.1.min.js"></script>
        <script src="assets/soundjs.flashplugin-0.2.0.min.js"></script>
        <script src="Main.js"></script>

    </head>
    <body onload="Main();">
        <canvas id="PongStage" width="480" height="320"></canvas>
    </body>
</html>
```

Como puede ver, es bastante corto y consiste principalmente en cargar las bibliotecas CreateJS.

Desde el lanzamiento de CreateJS (que básicamente agrupa todas las bibliotecas EaselJS separadas) ya no tenemos que descargar los archivos JS y alojarlos en nuestro sitio web; los archivos ahora se colocan en un CDN (Content Delivery Network) que nos permite cargar estos archivos de manera remota lo más rápido posible.

Repasemos el código:

```
<style>/* Removes Mobile Highlight */ *{-webkit-tap-highlight-color: rgba(0, 0, 0, 0);}<
```

Esta línea elimina el resaltado móvil que puede aparecer cuando intenta jugar el juego en un dispositivo móvil. (El resaltado móvil hace que el objeto lienzo quede resaltado y, por lo tanto, ignora los movimientos de los dedos).

A continuación, tenemos la carga de las bibliotecas CreateJS:>

```
<script src="http://code.createjs.com/easeljs-0.4.2.min.js"></script>
        <script src="http://code.createjs.com/tweenjs-0.2.0.min.js"></script>
```

```
        <script src="http://code.createjs.com/soundjs-0.2.0.min.js"></script>
        <script src="http://code.createjs.com/preloadjs-0.1.0.min.js"></script>
        <script src="http://code.createjs.com/movieclip-0.4.1.min.js"></script>
```

Este código carga los archivos JS desde CreateJS CDN y básicamente nos permite usar cualquiera de las funciones CreateJS en nuestro código

A continuación, cargaremos el complemento SoundJS Flash, que proporciona soporte de sonido para los navegadores que no son compatibles con el audio HTML5. Esto se hace usando un SWF (un objeto Flash) para cargar los sonidos.

```
<script src="assets/soundjs.flashplugin-0.2.0.min.js"></script>
```

En este caso, no usaremos el CDN; en su lugar, descargaremos la biblioteca SoundJS desde http://createjs.com/#!/SoundJS/download y colocaremos los archivos `soundjs.flashplugin-0.2.0.min.js` y `FlashAudioPlugin.swf` en una carpeta local llamada `assets`.

Último entre los archivos JS, cargaremos el archivo `Main.js` que contendrá todo el código de nuestro juego:

```
<script src="Main.js"></script>
```

Finalmente, pongamos un objeto Canvas en nuestro escenario.

```
<body onload="Main();">
        <canvas id="PongStage" width="480" height="320"></canvas>
    </body>
```

Ahora podemos comenzar a trabajar en el código del juego.

Paso 2: las variables

Nuestro código de juego estará dentro de un archivo llamado `Main.js`, así que cree y guarde esto ahora.

Antes que nada, definamos variables para todos los objetos gráficos en el juego:

```
var canvas; //Will be linked to the canvas in our index.html page
var stage; //Is the equivalent of stage in AS3; we'll add "children" to it

// Graphics
//[Background]

var bg; //The background graphic

//[Title View]
```

```
10
11
12  var main; //The Main Background
13  var startB; //The Start button in the main menu
14  var creditsB; //The credits button in the main menu
15
16  //[Credits]
17
18
19  var credits; //The Credits screen
20
21  //[Game View]
22
23
24  var player; //The player paddle graphic
25  var ball; //The ball graphic
26  var cpu; //The CPU paddle
27  var win; //The winning popup
28  var lose; //The losing popup
```

He agregado un comentario para cada variable para que sepas lo que vamos a cargar en esa variable

A continuación, los puntajes:

```
1  //[Score]
2
3  var playerScore; //The main player score
4  var cpuScore; //The CPU score
5  var cpuSpeed=6; //The speed of the CPU paddle; the faster it is the harder the game is
```

Vamos a necesitar variables para la velocidad de la pelota:

```
1  // Variables
2
3  var xSpeed = 5;
4  var ySpeed = 5;
```

Puede cambiar estos valores a lo que desee, si desea que el juego sea más fácil o más difícil.

Si eres desarrollador de Flash, sabes que Flash `onEnterFrame` es muy útil cuando creas juegos, ya que hay cosas que deben suceder en cada cuadro. (Si no está familiarizado con esta idea, consulte este artículo en Game Loop).

Tenemos un equivalente para `onEnterFrame` en CreateJS, y ese es el objeto `ticker`, que puede ejecutar código cada fracción de segundo. Vamos a crear la variable que se vinculará a ella:

```
1  var tkr = new Object;
```

A continuación tenemos el preloader, que usará los nuevos métodos PreloadJS.

```
1
2  //preloader
3  var preloader;
4  var manifest;
   var totalLoaded = 0;
```

- `preloader` - contendrá el objeto PreloadJS.
- `manifest` -contendrá la lista de archivos que necesitamos cargar.
- `totalLoaded` - esta variable contendrá la cantidad de archivos ya cargados.

Por último, pero no por ello menos importante, en nuestra lista de variables, tenemos `TitleView`, que guardará varios gráficos para mostrarlos juntos (como Flash `DisplayObjectContainer`).

```
var TitleView = new Container();
```

Pasemos a la función Principal ...

Paso 3: la función principal ()

Esta función es la primera función que se ejecuta después de cargar todos los archivos JS de `index.html`. Pero, ¿cómo se llama a esta función?

Bueno, ¿recuerdas esta línea del archivo `index.html`?

```
<body onload="Main();">
```

Este fragmento de código indica que una vez que se carguen las bibliotecas HTML (y JS), se debe ejecutar la función `Principal`.

Repasemoslo:

```
function Main()
{
    /* Link Canvas */

    canvas = document.getElementById('PongStage');
    stage = new Stage(canvas);

    stage.mouseEventsEnabled = true;


    /* Set The Flash Plugin for browsers that don't support SoundJS */
    SoundJS.FlashPlugin.BASE_PATH = "assets/";
    if (!SoundJS.checkPlugin(true)) {
      alert("Error!");
      return;
    }

    manifest = [
                {src:"bg.png", id:"bg"},
                {src:"main.png", id:"main"},
                {src:"startB.png", id:"startB"},
                {src:"creditsB.png", id:"creditsB"},
                {src:"credits.png", id:"credits"},
                {src:"paddle.png", id:"cpu"},
                {src:"paddle.png", id:"player"},
                {src:"ball.png", id:"ball"},
                {src:"win.png", id:"win"},
                {src:"lose.png", id:"lose"},
```

```
                {src:"playerScore.mp3|playerScore.ogg", id:"playerScore"},
                {src:"enemyScore.mp3|enemyScore.ogg", id:"enemyScore"},
                {src:"hit.mp3|hit.ogg", id:"hit"},
                {src:"wall.mp3|wall.ogg", id:"wall"}
            ];



    preloader = new PreloadJS();
    preloader.installPlugin(SoundJS);
    preloader.onProgress = handleProgress;
    preloader.onComplete = handleComplete;
    preloader.onFileLoad = handleFileLoad;
    preloader.loadManifest(manifest);

    /* Ticker */

    Ticker.setFPS(30);
    Ticker.addListener(stage);
}
```

Analicemos cada parte:

```
canvas = document.getElementById('PongStage');
stage = new Stage(canvas);

stage.mouseEventsEnabled = true;
```

Aquí vinculamos el objeto `PongStage` Canvas del archivo `index.html` a la variable canvas y luego creamos un objeto Stage desde ese lienzo. (El escenario nos permitirá colocar objetos en él).

`mouseEventsEnabled` nos permite usar eventos de mouse, por lo que podemos detectar movimientos y clics del mouse.

```
/* Set The Flash Plugin for browsers that don't support SoundJS */
    SoundJS.FlashPlugin.BASE_PATH = "assets/";
    if (!SoundJS.checkPlugin(true)) {
      alert("Error!");
      return;
    }
```

Aquí configuramos dónde reside el complemento de sonido de Flash para aquellos navegadores en los que no se admite el audio HTML5

```
manifest = [
            {src:"bg.png", id:"bg"},
            {src:"main.png", id:"main"},
            {src:"startB.png", id:"startB"},
            {src:"creditsB.png", id:"creditsB"},
            {src:"credits.png", id:"credits"},
            {src:"paddle.png", id:"cpu"},
            {src:"paddle.png", id:"player"},
            {src:"ball.png", id:"ball"},
            {src:"win.png", id:"win"},
            {src:"lose.png", id:"lose"},
            {src:"playerScore.mp3|playerScore.ogg", id:"playerScore"},
            {src:"enemyScore.mp3|enemyScore.ogg", id:"enemyScore"},
            {src:"hit.mp3|hit.ogg", id:"hit"},
            {src:"wall.mp3|wall.ogg", id:"wall"}
        ];
```

En la variable de manifiesto, colocamos una matriz de archivos que queremos cargar (y proporcionamos una identificación única para cada uno). Cada sonido tiene dos formatos: MP3 y OGG, porque diferentes navegadores son (en) compatibles con diferentes formatos.

```
preloader = new PreloadJS();
preloader.installPlugin(SoundJS);
preloader.onProgress = handleProgress;
preloader.onComplete = handleComplete;
preloader.onFileLoad = handleFileLoad;
preloader.loadManifest(manifest);
```

Aquí configuramos el objeto preloader usando PreloadJS. PreloadJS es una nueva adición a las bibliotecas CreateJS y bastante útil.

Creamos un nuevo objeto PreloadJS y lo colocamos en la variable `preloader`, luego asignamos un método a cada evento (`onProgress`, `onComplete`, `onFileLoad`). Finalmente usamos el `preloader` para cargar el manifiesto que creamos anteriormente.

```
Ticker.setFPS(30);
    Ticker.addListener(stage);
```

Aquí agregamos el objeto Ticker a la etapa y establecemos la frecuencia de cuadro a 30 FPS; lo usaremos más adelante en el juego para la funcionalidad `enterFrame`.

Paso 4: Creando las Funciones del Preloader

```
function handleProgress(event)
{
    //use event.loaded to get the percentage of the loading
}

function handleComplete(event) {
         //triggered when all loading is complete
}

function handleFileLoad(event) {
         //triggered when an individual file completes loading

         switch(event.type)
         {
            case PreloadJS.IMAGE:
            //image loaded
             var img = new Image();
              img.src = event.src;
              img.onload = handleLoadComplete;
              window[event.id] = new Bitmap(img);
            break;

            case PreloadJS.SOUND:
            //sound loaded
            handleLoadComplete();
            break;
         }
}
```

Repasemos las funciones:

- `handleProgress` - en esta función, podrá seguir el porcentaje del progreso de carga utilizando este parámetro: `event.loaded`. Puede usar esto para crear, por ejemplo, una barra de progreso.
- `handleComplete` - se llama a esta función una vez que se han cargado todos los archivos (en caso de que desee colocar algo allí).
- `handleFileLoad` - como cargamos dos tipos de archivos (imágenes y sonidos), tenemos esta función que manejará cada uno por separado. Si se trata de una imagen, creamos una imagen de mapa de bits y la colocamos en una variable (cuyo nombre es el mismo que el ID de la imagen cargada) y luego llamamos a la función `handleLoadComplete` (que escribiremos a continuación); si es un sonido, simplemente llamamos inmediatamente a `handleLoadComplete`.

Ahora veamos la función `handleLoadComplete` que acabo de mencionar:

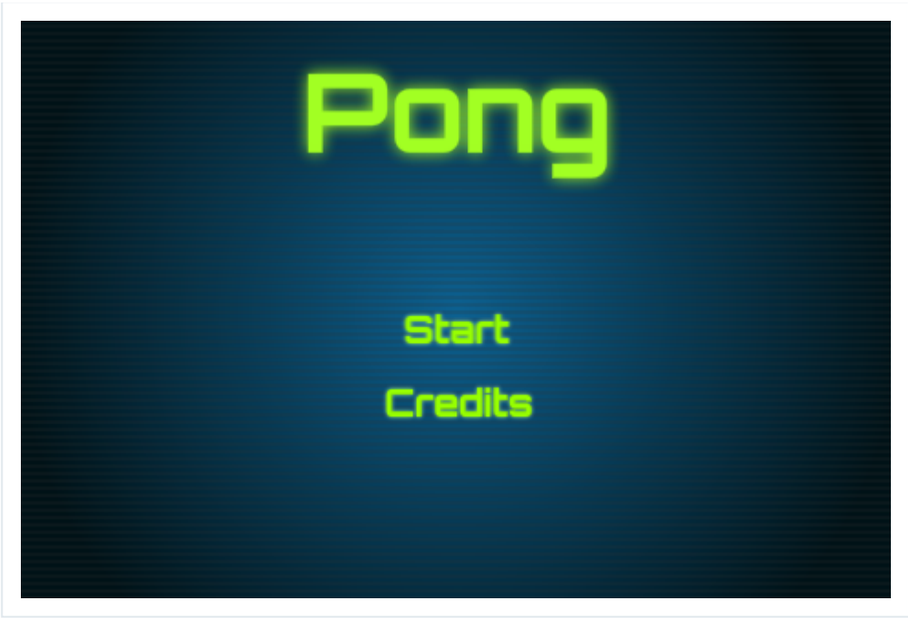
```
function handleLoadComplete(event)
{

   totalLoaded++;

   if(manifest.length==totalLoaded)
   {
       addTitleView();
   }
}
```

Es una función bastante sencilla; aumentamos la variable `total cargada` (que contiene el número de activos cargados hasta el momento) y luego verificamos si el número de elementos en nuestro manifiesto es el mismo que el número de activos cargados, y si es así, vaya a la pantalla del menú principal.

Paso 5: Creando el menú principal




```
function addTitleView()
{
    //console.log("Add Title View");
    startB.x = 240 - 31.5;
    startB.y = 160;
    startB.name = 'startB';

    creditsB.x = 241 - 42;
    creditsB.y = 200;

    TitleView.addChild(main, startB, creditsB);
    stage.addChild(bg, TitleView);
    stage.update();

    // Button Listeners

    startB.onPress = tweenTitleView;
    creditsB.onPress = showCredits;
```

Nada especial aquí. Colocamos las imágenes del botón Fondo, botón de inicio y créditos en el escenario y vinculamos los manejadores de eventos `onPress` a los botones de Inicio y Créditos.

Aquí están las funciones que muestran y eliminan la pantalla de créditos y el `tweenTitleView` que inicia el juego:

```
function showCredits()
{
    // Show Credits

    credits.x = 480;

    stage.addChild(credits);
    stage.update();
    Tween.get(credits).to({x:0}, 300);
    credits.onPress = hideCredits;
}

// Hide Credits

function hideCredits(e)
{
    Tween.get(credits).to({x:480}, 300).call(rmvCredits);
}

// Remove Credits

```

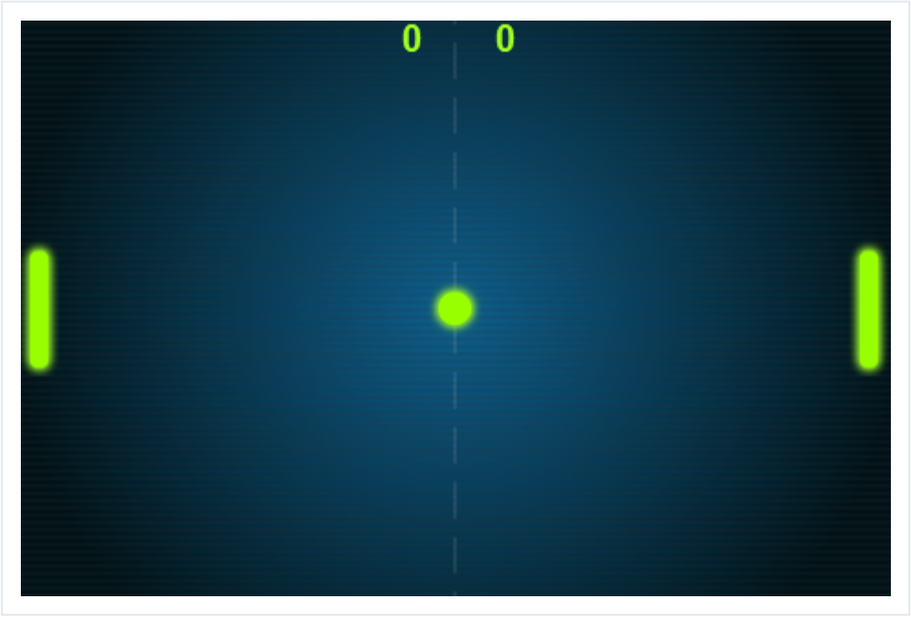
```
function rmvCredits()
{
    stage.removeChild(credits);
}

// Tween Title View

function tweenTitleView()
{
    // Start Game

    Tween.get(TitleView).to({y:-320}, 300).call(addGameView);
}
```

Paso 6: El código del juego



Hemos llegado a la parte principal de este tutorial, que es el código del juego en sí.

En primer lugar, debemos agregar todos los activos requeridos a la etapa, por lo que hacemos eso en la función `addGameView`:

```
function addGameView()
{
    // Destroy Menu & Credits screen

    stage.removeChild(TitleView);
    TitleView = null;
    credits = null;

    // Add Game View

    player.x = 2;
    player.y = 160 - 37.5;
    cpu.x = 480 - 25;
    cpu.y = 160 - 37.5;
    ball.x = 240 - 15;
    ball.y = 160 - 15;

    // Score

    playerScore = new Text('0', 'bold 20px Arial', '#A3FF24');
    playerScore.x = 211;
    playerScore.y = 20;

    cpuScore = new Text('0', 'bold 20px Arial', '#A3FF24');
    cpuScore.x = 262;
    cpuScore.y = 20;
```

```
27
28      stage.addChild(playerScore, cpuScore, player, cpu, ball);
29      stage.update();
30
31      // Start Listener
32
33      bg.onPress = startGame;
34  }
```

De nuevo, una función bastante sencilla que coloca los objetos en la pantalla y agrega un mouseEvent a la imagen de fondo, de modo que cuando el usuario haga clic en el juego se iniciará (llamaremos a la función `startGame`).

Repasemos la función `startGame`:

```
1  function startGame(e)
2  {
3      bg.onPress = null;
4      stage.onMouseMove = movePaddle;
5
6      Ticker.addListener(tkr, false);
7      tkr.tick = update;
8  }
```

Aquí, como puede ver, además de agregar un evento `onMouseMove` que moverá nuestra paleta. Agregamos el evento `tick`, que llamará a la función de `actualización` en cada cuadro.

Repasemos las funciones `movePaddle` y `reset`:

```
01  function movePaddle(e)
02  {
03      // Mouse Movement
04      player.y = e.stageY;
05  }
06
07  /* Reset */
08
09  function reset()
10  {
11      ball.x = 240 - 15;
12      ball.y = 160 - 15;
13      player.y = 160 - 37.5;
14      cpu.y = 160 - 37.5;
15
16      stage.onMouseMove = null;
17      Ticker.removeListener(tkr);
18      bg.onPress = startGame;
19  }
```

En `movePaddle`, básicamente colocamos la paleta del usuario en la coordenada y del mouse.

En el `reinicio`, hacemos algo similar a `agregarGameView`, excepto que aquí no agregamos ningún elemento gráfico porque ya están en la pantalla.

Usando la función de `alerta`, mostraremos la ventana emergente ganadora y perdedora:

```
function alert(e)
{
    Ticker.removeListener(tkr);
    stage.onMouseMove = null;
    bg.onPress = null

    if(e == 'win')
    {
        win.x = 140;
        win.y = -90;

        stage.addChild(win);
        Tween.get(win).to({y: 115}, 300);
    }
    else
    {
        lose.x = 140;
        lose.y = -90;

        stage.addChild(lose);
        Tween.get(lose).to({y: 115}, 300);
    }
}
```

Paso 7: The Game Loop

Ahora, para la última parte de nuestro tutorial, trabajaremos en la función de `actualización` (que ocurre en cada fotograma del juego, similar a la función `onEnterFrame` de Flash):

```
function update()
{
    // Ball Movement

    ball.x = ball.x + xSpeed;
    ball.y = ball.y + ySpeed;

    // Cpu Movement

    if(cpu.y < ball.y) {
        cpu.y = cpu.y + 4;
    }
    else if(cpu.y > ball.y) {
        cpu.y = cpu.y - 4;
    }

    // Wall Collision

    if((ball.y) < 0) { ySpeed = -ySpeed; SoundJS.play('wall'); };//Up
    if((ball.y + (30)) > 320) { ySpeed = -ySpeed; SoundJS.play('wall');};//down

    /* CPU Score */

    if((ball.x) < 0)
    {
        xSpeed = -xSpeed;
        cpuScore.text = parseInt(cpuScore.text + 1);
        reset();
        SoundJS.play('enemyScore');
    }

    /* Player Score */

    if((ball.x + (30)) > 480)
    {
        xSpeed = -xSpeed;
        playerScore.text = parseInt(playerScore.text + 1);
```

```
        reset();
        SoundJS.play('playerScore');
    }

    /* Cpu collision */

    if(ball.x + 30 > cpu.x && ball.x + 30 < cpu.x + 22 && ball.y >= cpu.y && ball.y < c
    {
        xSpeed *= -1;
        SoundJS.play('hit');
    }

    /* Player collision */

    if(ball.x <= player.x + 22 && ball.x > player.x && ball.y >= player.y && ball.y < p
    {
        xSpeed *= -1;
        SoundJS.play('hit');
    }

    /* Stop Paddle from going out of canvas */

    if(player.y >= 249)
    {
        player.y = 249;
    }

    /* Check for Win */

    if(playerScore.text == '10')
    {
        alert('win');
    }

    /* Check for Game Over */

    if(cpuScore.text == '10')
    {
        alert('lose');
    }
}
```

Parece aterrador, ¿no? No se preocupe, revisaremos cada parte y lo discutiremos.

```
// Ball Movement

ball.x = ball.x + xSpeed;
ball.y = ball.y + ySpeed;
```

En cada cuadro, la bola se moverá de acuerdo con sus valores de velocidad x , y

```
// Cpu Movement

    if((cpu.y+32) < (ball.y-14)) {
        cpu.y = cpu.y + cpuSpeed;
    }
    else if((cpu.y+32) > (ball.y+14)) {
        cpu.y = cpu.y - cpuSpeed;
    }
```

Aquí tenemos la IA básica de la computadora, en la que la paleta de la computadora simplemente sigue la pelota sin ninguna lógica especial. Simplemente comparamos la ubicación del centro de la paleta (por lo que agregamos 32 píxeles al valor de la Y de la

CPU) con la ubicación de la bola, con un pequeño desplazamiento, y movemos la paleta hacia arriba o hacia abajo según sea necesario.

```
if((ball.y) < 0) { //top
    ySpeed = -ySpeed;
    SoundJS.play('wall');
};
if((ball.y + (30)) > 320) { //bottom
    ySpeed = -ySpeed;
    SoundJS.play('wall');
};
```

Si la pelota golpea el borde superior o el borde inferior de la pantalla, la bola cambia de dirección y tocamos el sonido de golpe de pared.

```
/* CPU Score */
if((ball.x) < 0)
{
    xSpeed = -xSpeed;
    cpuScore.text = parseInt(cpuScore.text + 1);
    reset();
    SoundJS.play('enemyScore');
}
/* Player Score */
if((ball.x + (30)) > 480)
{
    xSpeed = -xSpeed;
    playerScore.text = parseInt(playerScore.text + 1);
    reset();
    SoundJS.play('playerScore');
}
```

El acceso al puntaje es simple: si la pelota pasa los bordes izquierdo o derecho, aumenta la puntuación del jugador o la CPU respectivamente, reproduce un sonido y restablece la ubicación de los objetos usando la función de `restablecimiento` que hemos discutido anteriormente.

```
/* CPU collision */
if(ball.x + 30 > cpu.x && ball.x + 30 < cpu.x + 22 && ball.y >= cpu.y && ball.y < cpu.y
{
    xSpeed *= -1;
    SoundJS.play('hit');
}
/* Player collision */
if(ball.x <= player.x + 22 && ball.x > player.x && ball.y >= player.y && ball.y < playe
{
    xSpeed *= -1;
    SoundJS.play('hit');
 }
```

Aquí tratamos con colisiones de la pelota con las paletas; cada vez que la pelota golpea una de las paletas, la pelota cambia de dirección y se reproduce un sonido

```
if(player.y >= 249)
{
    player.y = 249;
}
```

Si la paleta del jugador sale de los límites, la colocamos dentro de los límites.

```
/* Check for Win */
 if(playerScore.text == '10')
 {
    alert('win');
}
/* Check for Game Over */
if(cpuScore.text == '10')
{
    alert('lose');
}
```

En este fragmento, verificamos si el puntaje de alguno de los jugadores ha alcanzado los 10 puntos, y si es así mostramos la ventana emergente ganadora o perdedora al jugador (según su estado ganador).

Conclusión

Eso es todo, has creado un juego de pong completo usando CreateJS. Gracias por tomarse el tiempo para leer este tutorial.

Daniel Albu

Daniel Albu is a freelance Flash platform developer with more than ten years of experience in the production and deployment of rich media websites, games and applications. You can follow Daniel on Twitter: @danielalbu

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

Email Address

Update me weekly

Download Attachment

View Online Demo

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by

0 Comments Activetuts+ Mauricio A. B.

Recommend Tweet Share Sort by Best

Activetuts+ requires you to verify your email address before posting. Send verification email to afasia.band@gmail.com

Start the discussion…

Be the first to comment.

QUICK LINKS - Explore popular categories

ENVATO TUTS+ +

JOIN OUR COMMUNITY +

HELP +

tuts+

28,317 Tutorials 1,268 Courses 39,892 Translations

Follow Envato Tuts+ Facebook Twitter Pinterest