

INSTITUTO TECNOLOGICO DE CHIHUAHUA II



GRAFICACIÓN 9-10 AM

PROJECT: ENDLESS RUNNER

Author: Edith Ortiz Martinez #13550419

Adelaido Cano Lozano #14550322

Teacher: Alonso Salcido

Fecha: 07/10/17

Introducción

En esta serie de tutoriales, se mostrará cómo escribir un HTML5 Endless Runner Game similar al popular juego de Canabalt desde el principio. Se realizará en 3 pasos, cada uno cubrirá una parte o aspecto del juego. Cubrirá todo, desde la configuración del juego hasta la organización del código mediante módulos.

Básicamente, el juego consiste en saltar todos los obstáculos que se presenten durante el mismo.

A continuación se muestran los pasos para la creación de nuestro juego:

Paso1

En este tutorial, aprenderemos a utilizar una técnica conocida como patrón de módulo para crear objetos de juego y organizar nuestro código.

Los controles que se utilizaran será únicamente la barra espaciadora para saltar.

IMAGEN

Como se puede observar, el juego tiene un estado de juego completo (sistema de menús, juego, capacidad de reinicio, etc.).

La música comienza de forma predeterminada y el usuario tiene que activarla si desea oírla.

- La página HTML

Comenzaremos por definir nuestra página HTML y las reglas CSS.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Kandi Runner</title>
  <link rel="stylesheet" href="kandi.css" type="text/css" media="screen">
</head>
<body>
  <div class="wrapper">
    <canvas id="canvas" width="800" height="480">
      <p>Your browser does not support the required functionality to play this game.</p>
      <p>Please update to a modern browser such as <a href="http://www.google.com/chrome/">Google Chrome</a> to play.</p>
    </canvas>
  </div>
  <script type="text/javascript" src="kandi.js"></script>
</body>
</html>
```

Se carga nuestra hoja de estilo y código JavaScript y creando el elemento canvas que le dirá a los usuarios si su navegador no soporta el elemento Canvas.

El CSS básico para nuestro juego es el siguiente:

```
1 body {
2     font-family: arial, sans-serif;
3     font-size: 16px;
4 }
5
6 .wrapper {
7     width: 600px;
8     height: 360px;
9     -webkit-box-sizing: border-box;
10    -moz-box-sizing: border-box;
11    box-sizing: border-box;
12    position: absolute;
13    top: 5px;
14    left: 5px;
15 }
16
17 canvas {
18     position: absolute;
19     top: 0;
20     left: 0;
21     border: 1px solid black;
22     z-index: 1;
23     width: 600px;
24     height: 360px;
25     -webkit-box-sizing: border-box;
26     -moz-box-sizing: border-box;
27     box-sizing: border-box;
28 }
```

Esto posicionará el juego dentro de la div del contenedor y establecerá el juego para que se sienta 5px desde la parte superior y 5px desde la izquierda de la pantalla. La propiedad CSS tamaño de caja permite que el canvas tenga un borde de 1px y no tenga el 2px adicional para el borde afectar el tamaño general del juego.

Mientras que fijamos el tamaño del canvas para ser 800×480 en el HTML, en el CSS fijamos el tamaño ser 600×360 .

En lugar de recrear todos los gráficos y editar el juego para ajustarse al límite de ancho de 600px, dejamos que CSS escalara el canvas hacia abajo. Al usar CSS para cambiar la altura o el ancho del canvas, CSS escalará proporcionalmente el juego mientras mantiene una calidad de imagen bastante decente.

Antes de hablar sobre el lado JavaScript del juego, necesitamos hablar de algunos

patrones que se usan mucho en JavaScript: expresión de función inmediatamente invocada, patrón de módulo y cierres.

- Expresión de la función Inmediatamente-invocada

La forma "correcta" para crear un singleton en JavaScript es crear lo que se llama una expresión de función inmediatamente invocada.

Existen muchos nombres para hacer esto: expresión de función inmediatamente invocada (IIFE), función de auto-invocación y función anónima auto-ejecutada.

Independientemente del nombre, el punto de un IIFE es crear un cierre (más adelante) con el que todas las variables y funciones dentro del IIFE no son visibles fuera de su alcance .

Un IIFE se declara rodeando una función con paréntesis y ejecutándola.

```
(function() {
    var hidden = "You can't see me.";
})();

console.log(hidden); // ReferenceError: hidden is not defined
```

Al encapsular una función dentro del paréntesis, JavaScript convierte la declaración de función en una expresión. Cuando ejecuta (o invoca) la expresión llamándola (los dos paréntesis al final), la expresión se evalúa inmediatamente. Así, el patrón se llama una expresión de función inmediatamente invocada.

Cuando un IIFE devuelve un objeto, se crea un singleton.

```
var singleton = (function() {
    var visible = "Now you can see me.";
    return {
        visible : visible
    }
})();

console.log(singleton.visible); // "Now you can see me."
```

Probablemente ya esté familiarizado con este patrón si ha utilizado la animación de solicitud de Paul Irish polyfill (que debería usar):

```
var requestAnimationFrame = (function(){
    return window.requestAnimationFrame ||
           window.webkitRequestAnimationFrame ||
           window.mozRequestAnimationFrame ||
           window.oRequestAnimationFrame ||
           window.msRequestAnimationFrame ||
           function(callback, element){
               window.setTimeout(callback, 1000 / 60);
           };
})();
```

- El patrón del módulo

La técnica de devolver propiedades de un IIFE se conoce como patrón de módulo. El patrón que estamos usando se llama exportación de módulo ya que estamos exportando (regresando) propiedades desde dentro del IIFE para que puedan ser usadas fuera del IIFE.

No sólo puede exportar propiedades en un IIFE, pero también puede importar propiedades también. Esto funciona igual que pasar una variable a una función.

```
(function($) {
    // the IIFE now has access to jQuery through "$"
})(jQuery);
```

La mejor parte acerca de la importación de módulos es que el estado de la variable se guarda cuando se pasa al IIFE. Esto será importante más adelante cuando discutamos el módulo cargador de activos.

- Cierres

Los IIFEs están directamente relacionados con cierres. Un cierre es simplemente una función que está dentro de otra función. La función externa no puede acceder a ninguna de las variables de funciones internas, pero la función interna puede acceder a cualquiera de las variables de funciones externas. En otras palabras, JavaScript utiliza cierres para crear patrones orientados a objetos.

```

function Greeting() {
  var saying = "hello";
  this.sayHello = function() { // JavaScript closure
    var name = "Bob";
    console.log(saying + " " + name);
  }
  this.sayGoodbye = function() {
    // this function cannot access the variable 'name' from the previous closure
    console.log("goodbye " + name + "'");
  }
}

greet = new Greeting();
greet.sayHello();      // "hello Bob"
greet.sayGoodbye();   // "goodbye "

```

Debido a que el nombre de la variable se declaró dentro del cierre, la función sayGoodbye no puede acceder a ella. Esto puede parecer obvio y lo es, pero hay una cosa muy importante a recordar cuando se trata de cierres: los cierres conservan esta referencia al alcance en el que fueron creados.

Esto se explica mejor mediante un ejemplo. Supongamos que tiene una función constructora que tiene una variable pública. Digamos también que tiene una función de ayuda privada que hará un cálculo en la variable y que la función de ayuda es utilizada por una función pública:

```

function Example() {
  this.base = 10; // public variable
  function isEven() { // private helper function
    return this.base % 2 === 0;
  }
  this.calculate = function() { // public function
    if (isEven()) {
      console.log("even");
    }
    else {
      console.log("odd");
    }
  }
}

e = new Example();
e.calculate(); // "odd"

```

En este ejemplo simple, esperamos que la salida de this.calculate () sea 'even', pero en su lugar obtendremos 'odd'. ¿Pero por qué? Obviamente 10 es un número par, así que ¿por qué piensa que es extraño?

La captura aquí es que la función isEven () es un cierre, y cómo fue declarada como

una entidad de la función (y no como una propiedad de la función, es decir `this.isEven ()`), esta referencia ya no apunta a la función de ejemplo. Dado que esta referencia no es la función de ejemplo, `this.base` no existe dentro del cierre. Esta es la razón por la que `isEven ()` devuelve `false` ya que `undefined% 2 === false`.

Hay tres métodos utilizados para solucionar este problema. El primero es crear una referencia privada a este puntero y utilizar que dentro del cierre en lugar de esto:

```
function Example() {
    var _this = this;
    this.base = 10;
    function isEven() {
        return _this.base % 2 === 0; // this will return true
    }

    // ...
}
```

Otro método es utilizar el método de llamada para pasar la correcta esta referencia:

```
function Example() {
    // ...

    this.calculate = function() {
        if (isEven.call(this)) // this will return true
        // ...
    }
}
```

El último método es vincular la función al ámbito correcto:

```
function Example() {
    this.base = 10;
    function isEven() {
        return this.base % 2 === 0;
    }
    var isEven = isEven.bind(this); // this will return true

    // ...
}
```

- El JavaScript

Ahora que hemos hablado de IIFEs y cierres, tenemos la base para crear nuestro juego.

```

1  (function () {
2      // define variables
3      var canvas = document.getElementById('canvas');
4      var ctx = canvas.getContext('2d');
5      var player = {};
6      var ground = [];
7      var platformWidth = 32;
8      var platformHeight = canvas.height - platformWidth * 4;
9

```

```

    /* Request Animation Polyfill
     */
    var requestAnimationFrame = (function(){
        return window.requestAnimationFrame ||
            window.webkitRequestAnimationFrame ||
            window.mozRequestAnimationFrame ||
            window.oRequestAnimationFrame ||
            window.msRequestAnimationFrame ||
            function(callback, element){
                window.setTimeout(callback, 1000 / 60);
            };
    })();

```

Lo primero que hacemos es encerrar nuestro código dentro de un IIFE. Este es un patrón muy popular, ya que impide que todas nuestras variables y funciones de ser puesto en el espacio de nombres global, potencialmente causando problemas en el futuro. Una nota sobre esta técnica: si está depurando su programa, ayuda a eliminar el IIFE ya que no podrá acceder a nada desde la consola si su programa está incluido en un IIFE.

A continuación, obtenemos nuestro contexto de lienzo y lienzo para que podamos usarlos y también establecemos algunas variables predeterminadas para dónde se colocarán las plataformas (4 filas desde la parte inferior). También creamos la animación de solicitud polyfill para usar cuando animamos. Por último, creamos los inicios del módulo cargador de activos.

- Módulo del cargador de activos

El propósito principal del módulo de cargador de activos es crear un objeto que contenga todos nuestros objetos de imagen y audio para que podamos hacer referencia a ellos cuando los necesitamos. El propósito secundario del cargador de activos es saber cuándo todos los activos han terminado de descargar así que sabemos cuándo comenzar el juego. Si, por ejemplo, tratamos de dibujar una imagen en el lienzo antes de terminar de descargar, no se dibujaría nada.

Mi primer intento de crear un cargador de activos no fue muy bueno. Una gran cantidad de código se repitió que realmente hizo desordenado para trabajar. Desde entonces he mejorado mucho en la idea que a su vez ha hecho más fácil trabajar con.

```
/**...
var assetLoader = (function() {
    // images dictionary
    this.imgs      = {
        'bg'          : 'imgs/bg.png',
        'sky'         : 'imgs/sky.png',
        'backdrop'    : 'imgs/backdrop.png',
        'backdrop2'   : 'imgs/backdrop_ground.png',
        'grass'       : 'imgs/grass.png',
        'avatar_normal': 'imgs/normal_walk.png'
    };

    var assetsLoaded = 0;                                // how many assets have been loaded
    var numImgss     = Object.keys(this.imgs).length;    // total number of image assets
    this.totalAssest = numImgss;                         // total number of assets

    /**
     * Ensure all assets are loaded before using them
     * @param {number} dic - Dictionary name ('imgs', 'sounds', 'fonts')
     * @param {number} name - Asset name in the dictionary
     */
    function assetLoaded(dic, name) {
        // don't count assets that have already loaded
        if (this[dic][name].status !== 'loading') {
            return;
        }

        this[dic][name].status = 'loaded';
        assetsLoaded++;

        // finished callback
        if (assetsLoaded === this.totalAssest && typeof this.finished === 'function') {
            this.finished();
        }
    }
})
```

```

        if (assetsLoaded === this.totalAssets && typeof this.finished === 'function') {
            this.finished();
        }
    }

    /**
     * Create assets, set callback for asset loading, set asset source
     */
    this.downloadAll = function() {
        var _this = this;
        var src;

        // load images
        for (var img in this.imgs) {
            if (this.imgs.hasOwnProperty(img)) {
                src = this.imgs[img];

                // create a closure for event binding
                (function(_this, img) {
                    _this.imgs[img] = new Image();
                    _this.imgs[img].status = 'loading';
                    _this.imgs[img].name = img;
                    _this.imgs[img].onload = function() { assetLoaded.call(_this, 'imgs', img) };
                    _this.imgs[img].src = src;
                })(_this, img);
            }
        }
    }

    return {
        imgs: this.imgs,
        totalAssets: this.totalAssets,
        downloadAll: this.downloadAll
    };
}();

assetLoader.finished = function() {
}

```

En primer lugar, creamos un objeto que enumera todos nuestros activos de imagen, así como sus caminos. Luego contamos cuántas imágenes hay en el objeto usando `Object.keys()`. Esto nos permite saber cuántos activos deben cargarse antes de comenzar el juego.

A continuación, creamos una función de devolución de llamada `assetLoaded()` que lleva el nombre de un diccionario (ya que sólo cargamos imágenes hay sólo un diccionario) y el nombre del elemento de ese diccionario que se cargó. La función marca el activo como cargado e incrementa nuestros activos de contador de activos. Finalmente, si todos los activos han terminado de cargar llamará a la función `this.finished()` si se ha definido. Al definir la función al final, sabemos cuándo se han cargado todos nuestros activos.

A continuación, creamos una función que inicia el proceso de descarga. Esta

función hace un bucle sobre cada uno de los elementos de imagen y lo convierte en un objeto de imagen, lo marca como carga, le da una función de devolución de llamada una vez que se ha cargado y establece su origen.

Observe que se envuelve el proceso dentro de un IIFE. Esto es importante porque estamos asignando una devolución de llamada que utiliza la variable img. Si no lo envolveríamos en un IIFE, la devolución de llamada de la función onload () siempre pasaría el nombre del último recurso en el objeto imgs.

```
for (var i = 0; i < 10; i++) {
    setTimeout(function() { console.log(i); }, 1000); // prints "10" ten times
}
```

Esto sucede porque la variable img ya ha progresado a través del bucle completo antes de llamar a la función onload (). Por lo tanto, cuando se trata de llamadas asíncronas es importante que guarde el estado de las variables que estamos utilizando para que el estado de la variable no cambie antes de que termine la llamada asíncrona. Aquí es donde los IIFEs son útiles ya que salvarán el estado de la variable cuando fue pasado al IIFE.

```
for (var i = 0; i < 10; i++) {
    (function(i) {
        setTimeout(function() { console.log(i); }, 1000); // prints 0-9
    })(i);
}
```

Lo último que hace el cargador de activos es devolver un objeto con las propiedades que queremos que sean accesibles.

Una vez que el cargador de activos haya terminado de cargar todas las imágenes, ahora se puede acceder a cualquier imagen usando assetLoader.imgs [imageName].

- Hoja de sprites

Con el cargador de activos listo para funcionar, ahora podemos crear una animación de spritesheet para nuestro personaje principal. Puedes leer más sobre esto desde el artículo que escribí.

```

    */
function Spritesheet(path, frameWidth, frameHeight) {
    this.image = new Image();
    this.frameWidth = frameWidth;
    this.frameHeight = frameHeight;

    // calculate the number of frames in a row after the image loads
    var self = this;
    this.image.onload = function() {
        self.framesPerRow = Math.floor(self.image.width / self.frameWidth);
    };

    this.image.src = path;
}

/**
 * Creates an animation from a spritesheet.
 * @param {SpriteSheet} - The spritesheet used to create the animation.
 * @param {number}      - Number of frames to wait for before transitioning the animation.
 * @param {array}       - Range or sequence of frame numbers for the animation.
 * @param {boolean}     - Repeat the animation once completed.
 */
function Animation(spritesheet, frameSpeed, startFrame, endFrame) {

    var animationSequence = []; // array holding the order of the animation
    var currentFrame = 0; // the current frame to draw
    var counter = 0; // keep track of frame rate

    // start and end range for frames
    for (var frameNumber = startFrame; frameNumber <= endFrame; frameNumber++)
        animationSequence.push(frameNumber);

    /**
     * Update the animation
     */
    this.update = function() {

        // update to the next frame if it is time
        var counter = 0; // keep track of frame rate

        // start and end range for frames
        for (var frameNumber = startFrame; frameNumber <= endFrame; frameNumber++)
            animationSequence.push(frameNumber);

        /**
         * Update the animation
         */
        this.update = function() {

            // update to the next frame if it is time
            if (counter == (frameSpeed - 1))
                currentFrame = (currentFrame + 1) % animationSequence.length;

            // update the counter
            counter = (counter + 1) % frameSpeed;
        };

        /**
         * Draw the current frame
         * @param {integer} x - X position to draw
         * @param {integer} y - Y position to draw
         */
        this.draw = function(x, y) {
            // get the row and col of the frame
            var row = Math.floor(animationSequence[currentFrame] / spritesheet.framesPerRow);
            var col = Math.floor(animationSequence[currentFrame] % spritesheet.framesPerRow);

            ctx.drawImage(
                spritesheet.image,
                col * spritesheet.frameWidth, row * spritesheet.frameHeight,
                spritesheet.frameWidth, spritesheet.frameHeight,
                x, y,
                spritesheet.frameWidth, spritesheet.frameHeight);
        };
    };
}

```

- Parallax

Para hacer el juego más interesante, vamos a crear un fondo de paralelo. Esta técnica utiliza el mismo efecto que el fondo panorámico de mi primer tutorial, solo agregamos más imágenes que se desplazan a diferentes velocidades.

```
/*
var background = (function() {
  var sky    = {};
  var backdrop = {};
  var backdrop2 = {};

  /**
   * Draw the backgrounds to the screen at different speeds
   */
  this.draw = function() {
    ctx.drawImage(assetLoader.imgs.bg, 0, 0);

    // Pan background
    sky.x -= sky.speed;
    backdrop.x -= backdrop.speed;
    backdrop2.x -= backdrop2.speed;

    // draw images side by side to loop
    ctx.drawImage(assetLoader.imgs.sky, sky.x, sky.y);
    ctx.drawImage(assetLoader.imgs.sky, sky.x + canvas.width, sky.y);

    ctx.drawImage(assetLoader.imgs.backdrop, backdrop.x, backdrop.y);
    ctx.drawImage(assetLoader.imgs.backdrop, backdrop.x + canvas.width, backdrop.y);

    ctx.drawImage(assetLoader.imgs.backdrop2, backdrop2.x, backdrop2.y);
    ctx.drawImage(assetLoader.imgs.backdrop2, backdrop2.x + canvas.width, backdrop2.y);

    // If the image scrolled off the screen, reset
    if (sky.x + assetLoader.imgs.sky.width <= 0)
      sky.x = 0;
    if (backdrop.x + assetLoader.imgs.backdrop.width <= 0)
      backdrop.x = 0;
    if (backdrop2.x + assetLoader.imgs.backdrop2.width <= 0)
      backdrop2.x = 0;
  };
}

RESET BACKGROUND TO ZERO
*/
this.reset = function() {
  sky.x = 0;
  sky.y = 0;
  sky.speed = 0.2;

  backdrop.x = 0;
  backdrop.y = 0;
  backdrop.speed = 0.4;

  backdrop2.x = 0;
  backdrop2.y = 0;
  backdrop2.speed = 0.6;
}

return {
  draw: this.draw,
  reset: this.reset
};
})();
```

De nuevo, utilizamos un IIFE para crear un singleton que expone solamente las funciones draw () y reset ().

- Toques finales

Lo último que tenemos que hacer es crear el bucle de animación e iniciar el juego.

```
function startGame() {
    // setup the player
    player.width = 60;
    player.height = 96;
    player.speed = 6;
    player.sheet = new Spritesheet('imgs/normal_walk.png', player.width, player.height);
    player.anim = new Animation(player.sheet, 4, 0, 15);

    // create the ground tiles
    for (i = 0, length = Math.floor(canvas.width / platformWidth) + 2; i < length; i++) {
        ground[i] = {'x': i * platformWidth, 'y': platformHeight};
    }

    background.reset();
    animate();
}

assetLoader.downloadAll();
})();
```

Cuando se inicia el juego, establece las propiedades predeterminadas del reproductor. La propiedad de velocidad se utiliza para mover todos los objetos de la pantalla a una velocidad constante. Este movimiento hace que el jugador se sienta como si estuvieran moviéndose aunque estén parados.

El juego también crea todas las fichas de terreno en las que el jugador se encuentra. Creamos 2 más azulejos entonces cabría en la pantalla de modo que tengamos un amortiguador de los azulejos y no parecen parpadear en existencia cuando los creamos. Lo último que hace es restablecer el objeto de fondo e iniciar el bucle de animación.

El bucle de animación primero dibuja el fondo de paralejo al lienzo. A continuación, loops sobre todos los azulejos de tierra y los mueve por la velocidad de los jugadores y luego los atrae. Si una de las fichas desaparece de la pantalla, la elimina de la matriz y crea una nueva al final de la matriz. Esto crea el efecto de una

plataforma sin fin.

El bucle de animación luego actualiza la animación de spritesheet del jugador y lo dibuja a la pantalla.

Lo último que hacemos es llamar a la función downloadAll () de assetLoader para comenzar el proceso de descarga e iniciar el juego cuando haya terminado.

Parte 2

Ahora tenemos un endless runner agradable completo con los malos, las trampas, y las plataformas y las alturas que varían.

El HTML y el CSS para nuestro juego no han cambiado, así que nos moveremos derecho en hablar de cómo planeamos organizar nuestro código usando la técnica del Javascript de la herencia del prototipo.

- Herencia Prototípica

Prototypal Inheritance es la forma en que JavaScript implementa el principio orientado a objetos de la herencia. Sin embargo, como JavaScript no tiene clases, implementa la herencia de manera diferente que otros lenguajes basados en clases (como Java y C ++). Debido a esto, la herencia de prototipo puede ser confusa para los nuevos desarrolladores de JavaScript.

Hemos hablado de cómo funciona la herencia prototípica de JavaScript antes, por lo que haré una recapitulación rápida aquí.

Cada objeto en JavaScript tiene una referencia a un objeto principal llamado prototipo del objeto. Esta referencia se utiliza para encontrar propiedades que pueden existir en el prototipo del objeto que no existen en el objeto actual. De esta manera, JavaScript es capaz de vincular objetos juntos para que puedan compartir un conjunto común de propiedades.

- El JavaScript

Con la comprensión de cómo funciona la herencia prototípica, podemos comenzar a discutir cómo la usaremos en nuestro juego. Cada objeto de nuestro juego (el jugador, las plataformas, los malos y las plantas) comparten un conjunto común de funcionalidad: todos ellos necesitan saber su posición actual y también cómo

actualizar su posición en cada fotograma.

Para implementar esta funcionalidad, crearemos un objeto Vector básico que contendrá la información necesaria. Cada objeto del juego heredará de Vector para que no tengamos que implementar la funcionalidad en los objetos mismos.

```
function Vector(x, y, dx, dy) {
    // position
    this.x = x || 0;
    this.y = y || 0;
    // direction
    this.dx = dx || 0;
    this.dy = dy || 0;
}

/**
 * Advance the vectors position by dx,dy
 */
Vector.prototype.advance = function() {
    this.x += this.dx;
    this.y += this.dy;
};

/**
 * Get the minimum distance between two vectors
 * @param {Vector}
 * @return minDist
 */
Vector.prototype.minDist = function(vec) {
    var minDist = Infinity;
    var max     = Math.max( Math.abs(this.dx), Math.abs(this.dy),
                           Math.abs(vec.dx ), Math.abs(vec.dy ) );
    var slice   = 1 / max;

    var x, y, distSquared;

    // get the middle of each vector
    var vec1 = {}, vec2 = {};
    vec1.x = this.x + this.width/2;
    vec1.y = this.y + this.height/2;
    vec2.x = vec.x + vec.width/2;
    vec2.y = vec.y + vec.height/2;
    for (var percent = 0; percent < 1; percent += slice) {
        x = (vec1.x + this.dx * percent) - (vec2.x + vec.dx * percent);
        y = (vec1.y + this.dy * percent) - (vec2.y + vec.dy * percent);
        distSquared = (x * x) + (y * y);
        if (distSquared < minDist) {
            minDist = distSquared;
        }
    }
    return Math.sqrt(minDist);
}
```

- El módulo del reproductor

En el tutorial anterior el jugador era sólo un objeto independiente. Vamos a convertir ese objeto en un singleton adecuado, mientras que dejar que hereda de Vector. Para ello, utilizaremos la importación modular para pasar el objeto Vector prototípico al IIFE y luego agregar nuestras propiedades de reproductor directamente al objeto importado.

```

var player = (function(player) {
    // add properties directly to the player imported object
    player.width      = 60;
    player.height     = 96;
    player.speed      = 6;

    // jumping
    player.gravity    = 1;
    player.dy         = 0;
    player.jumpDy     = -10;
    player.isFalling  = false;
    player.isJumping  = false;

    // spritesheets
    player.sheet       = new SpriteSheet('imgs/normal_walk.png', player.width, player.height);
    player.walkAnim   = new Animation(player.sheet, 4, 0, 15);
    player.jumpAnim   = new Animation(player.sheet, 4, 15, 15);
    player.fallAnim   = new Animation(player.sheet, 4, 11, 11);
    player.anim        = player.walkAnim;

    Vector.call(player, 0, 0, 0, player.dy);

    var jumpCounter = 0; // how long the jump button can be pressed down

    /**
     * Update the player's position and animation
     */
    player.update = function() {

        // jump if not currently jumping or falling
        if (KEY_STATUS.space && player.dy === 0 && !player.isJumping) {
            player.isJumping = true;
            player.dy = player.jumpDy;
            jumpCounter = 12;
        }

        // jump higher if the space bar is continually pressed
        if (KEY_STATUS.space && jumpCounter) {
    
```

Como puedes ver, es un objeto muy simple. El único truco para el objeto Sprite es que debe pasarle un tipo que coincida con uno de los nombres de imagen del cargador de recursos. Esto hace que sea muy fácil de dibujar la imagen, ya que podemos agarrar la imagen directamente desde el cargador de activos por su nombre.

Otra cosa importante a tener en cuenta es que la velocidad de todos los objetos siempre será la velocidad del jugador, pero en la dirección opuesta. Esto nos permite incrementar la velocidad del jugador durante todo el juego, aumentando la dificultad para el jugador, y todavía tenemos todos los objetos en la pantalla que reflejan ese cambio sin trabajo adicional.

- Plataformas

Para ayudar a nuestro juego a ser más desafiante, vamos a agregar plataformas de diferentes longitudes y alturas, así como las diferencias entre los grupos de la plataforma. Utilizando una técnica de construcción de un corredor infinito Canabalt-Style, determinaremos aleatoriamente cuántas brechas crear usando gapLength y cuántas plataformas crear usando platformLength. También crearemos una función que generará las brechas y plataformas que se pueden utilizar en la función animate ()�.

```
function rand(low, high) {
    return Math.floor( Math.random() * (high - low + 1) + low );
}

/**
 * Bound a number between range
 * @param {integer} num - Number to bound
 * @param {integer}
 * @param {integer}
 */
function bound(num, low, high) {
    return Math.max( Math.min(num, high), low );
}

/**
 * Asset pre-loader object. Loads all images
 */
var assetLoader = (function() {
    // images dictionary
    this.imgs      = {
        'bg'          : 'imgs/bg.png',
        'sky'         : 'imgs/sky.png',
        'backdrop'    : 'imgs/backdrop.png',
        'backdrop2'   : 'imgs/backdrop_ground.png',
        'grass'       : 'imgs/grass.png',
        'avatar_normal' : 'imgs/normal_walk.png',
        'water'       : 'imgs/water.png',
        'grass1'      : 'imgs/grassMid1.png',
        'grass2'      : 'imgs/grassMid2.png',
        'bridge'      : 'imgs/bridge.png',
        'plant'       : 'imgs/plant.png',
        'bush1'       : 'imgs/bush1.png',
        'bush2'       : 'imgs/bush2.png',
        'cliff'       : 'imgs/grassCliffRight.png',
        'spikes'      : 'imgs/spikes.png',
        'box'          : 'imgs/boxCoin.png',
        'slime'        : 'imgs/slime.png'
    };
});
```

La función primero incrementa la puntuación del jugador que controla cuántas plataformas ha cruzado el jugador. Esta puntuación se traducirá en hasta qué punto el jugador ha corrido. A continuación, la función crea brechas al no crear nuevos sprites si gapLength es mayor que 0. Si no quedan espacios, se creará una nueva plataforma basada en la altura de la plataforma actual. La función también aleatoriamente desovar plantas y enemigos en las plataformas.

Como no queremos que estos nuevos sprites aparezcan en la pantalla, los pondremos justo detrás de la vista del jugador. Esto significa ponerlos un poco fuera del lienzo para comenzar de modo que luego se mueven en el lienzo. Al tomar el mod de la anchura de una plataforma por la velocidad del jugador, eliminamos la brecha entre las plataformas que se crearán al generar un sprite en el mismo momento en que aumentamos la velocidad del jugador.

Si no quedan huecos o plataformas para crear, la función generará aleatoriamente una nueva longitud de hueco basada en la velocidad del jugador (velocidades más altas significan intervalos más largos), determine la altura de la próxima plataforma asegurándose de que nunca vaya 2 alturas por encima de la corriente altura (de lo contrario el jugador no podría hacer el salto), y determina la longitud de la próxima plataforma.

- Continuación de las plataformas

Para determinar la posición de cada plataforma, así como el tipo de imagen que se utilizará para la plataforma, usaremos platformHeight para tomar las decisiones por nosotros. Dado que tenemos cinco alturas de plataforma diferentes, platformHeight será un número entre 0 y 4.

Para determinar qué plataforma usar, crearemos una función auxiliar que devolverá el nombre de la imagen a utilizar para la plataforma en función del valor de platformHeight.

```
function getType() {
  var type;
  switch (platformHeight) {
    case 0:
    case 1:
      type = Math.random() > 0.5 ? 'grass1' : 'grass2';
      break;
    case 2:
      type = 'grass';
      break;
    case 3:
      type = 'bridge';
      break;
    case 4:
      type = 'box';
      break;
  }
  if (platformLength === 1 && platformHeight < 3 && rand(0, 3) === 0) {
    type = 'cliff';
  }

  return type;
}

/**
 * Update all ground position and draw. Also check for collision against the player.
 */
function updateGround() {
  // animate ground
  player.isFalling = true;
  for (var i = 0; i < ground.length; i++) {
    ground[i].update();
    ground[i].draw();

    // stop the player from falling when landing on a platform
    var angle;
    if (player.minDist(ground[i]) <= player.height/2 + platformWidth/2 &&
        (angle = Math.atan2(player.y - ground[i].y, player.x - ground[i].x) * 180/Math.PI) > -130 &&
```

La función determina primero que las plantas y los arbustos no pueden aparecer antes de que el jugador haya acumulado 40 puntos. Después de eso, las plantas tienen un 5% de probabilidad de desovar y solo aparecerán en plataformas de hierba. El último requisito asegura que las plantas y los arbustos no engendran en puentes y ladrillos, lo que parecería una tontería.

Las plantas y los arbustos pueden engendrar alrededor del mismo, pero los arbustos pueden desovar solamente en la plataforma del tercer nivel (apenas porque ése es cómo lo quise). Los arbustos son un pedazo de dos imágenes, así que requiere que dos sprites se generen para él.

A continuación vamos a generar enemigos en nuestro juego.

```
    /**
     * Spawns enemies
     */
    function spawnEnemies() {
        if (score > 100 && Math.random() > 0.96 && enemies.length < 3 && platformLength > 5 &&
            (enemies.length ? canvas.width - enemies[enemies.length-1].x >= platformWidth * 3 || 
            canvas.width - enemies[enemies.length-1].x < platformWidth : true)) {
            enemies.push(new Sprite(
                canvas.width + platformWidth % player.speed,
                platformBase - platformHeight * platformSpacer - platformWidth,
                Math.random() > 0.5 ? 'spikes' : 'slime'
            ));
        }
    }

    /**
     * Game loop
     */
    function animate() {
        if (!stop) {
            requestAnimationFrame( animate );
            ctx.clearRect(0, 0, canvas.width, canvas.height);

            background.draw();

            // update entities
            updateWater();
            updateEnvironment();
            updatePlayer();
            updateGround();
            updateEnemies();

            // draw the score
            ctx.fillText('Score: ' + score + 'm', canvas.width - 140, 30);

            // spawn a new Sprite
            if (ticker % Math.floor(platformWidth / player.speed) === 0) {
                spawnSprites();
            }
        }
    }
```

Los enemigos tienen un requisito muy interesante para desovar. En primer lugar, no generar un enemigo a menos que el jugador ha acumulado 100 puntos. De esta manera no se encuentran demasiado temprano en el juego y tener una idea de cómo saltar antes de que tienen que poner realmente su habilidad a la prueba.

A continuación, sólo queremos que los enemigos generen el 4% del tiempo. Esto se debe a que al jugar pruebas, sentí que el 5% creó demasiados enemigos, mientras que el 4% creó una buena cantidad. A continuación, solo permitiremos que 3

enemigos estén en la pantalla a la vez para que el jugador no sea bombardeado por enemigos y no pueda evitarlos.

También sólo generamos un enemigo en grupos de plataforma de más de 5 plataformas. Esto asegura que el jugador puede saltar sobre el enemigo y todavía aterrizar en una plataforma si se cronometra correctamente. El último requisito mira a la colocación de otros enemigos en la pantalla y evita que otro enemigo desove dentro de 3 plataformas de la misma. Esto asegura que un jugador tiene una cierta cantidad de espacio entre los enemigos para que haya un lugar seguro para aterrizar.

Una vez que se permite que un enemigo engendre, hay una posibilidad de 50/50 de desovar o un "pico" o un "limo".

- Actualización de Sprites

Ahora que podemos crear todos los sprites a nuestro juego, vamos a averiguar cómo vamos a animar a todos.

Usted puede haber notado que no creamos una función para frezar el "agua" sprites. En su lugar, ya que el agua no juega ningún papel en el juego, excepto como decoración, vamos a reutilizar cualquier sprites que salgan de la pantalla como nuevos sprites de agua que vienen en la pantalla. De esta manera los sprites del agua actúan como una piscina del objeto que ayuda a ahorrar memoria.

```

    }

    function updatewater() {
        // animate water
        for (var i = 0; i < water.length; i++) {
            water[i].update();
            water[i].draw();
        }

        // remove water that has gone off screen
        if (water[0] && water[0].x < -platformWidth) {
            var w = water.splice(0, 1)[0];
            w.x = water[water.length-1].x + platformWidth;
            water.push(w);
        }
    }

    /**
     * Update all environment position and draw.
     */
    function updateEnvironment() {
        // animate environment
        for (var i = 0; i < environment.length; i++) {
            environment[i].update();
            environment[i].draw();
        }

        // remove environment that have gone off screen
        if (environment[0] && environment[0].x < -platformWidth) {
            environment.splice(0, 1);
        }
    }

    /**
     * Update all enemies position and draw. Also check for collision against the player.
     */
    function updateEnemies() {
        // animate enemies
        for (var i = 0; i < enemies.length; i++) {

```

Actualizar los enemigos es casi exactamente lo mismo que actualizar los entornos excepto que necesitamos comprobar que el reproductor no chocó con uno de ellos usando la función minDist ()�

- Animando

Ya casi terminamos con el juego. Ahora que podemos generar y actualizar sprites, todo lo que queda por hacer es animarlos cada fotograma y comenzar el juego.

Parte 3:

- La interfaz de usuario del juego

Antes de que podamos agregar la interfaz de usuario al juego, debemos discutir los diferentes enfoques para crear una interfaz de usuario de juegos.

Existen dos métodos principales para dibujar UI a la pantalla: usar el lienzo para dibujar la interfaz de usuario o utilizar elementos DOM nativos y CSS.

- El Menú Principal

Ahora que hemos discutido los pros y los contras de cada enfoque de interfaz de usuario, es hora de hacer nuestra interfaz de usuario de juegos.

El primer menú que vamos a crear es el menú principal que los jugadores verán cuando quieran jugar el juego.



El menú es relativamente simple, sólo el título y dos botones más la imagen de fondo. Dado que los botones son también sencillos, puedo crearlos utilizando el lienzo o utilizando elementos DOM. No estoy pensando en hacer que el juego responda, así que he elegido para hacerlos en el DOM para ahorrar un poco de tiempo de codificación.

El menú principal se encuentra dentro de su propia div para que podamos restringir su tamaño para que coincida con el lienzo. Me gusta hacer menús usando listas desordenadas, ya que es un hábito mío de crear sitios web. No son necesarios aquí

y fácilmente puede utilizar las etiquetas de anclaje sin ellos.

Las etiquetas de anclaje tienen javascript: void (0); como el href. Navegadores le permiten crear href que utilizan código javascript prefaciando el código con la palabra javascript :. Utilizando void (0); estamos diciendo javascript para no hacer nada, por lo que el enlace no hacer nada.

También podríamos usar un botón aquí en lugar de una etiqueta de anclaje, pero considero que reemplazar el estilo por defecto de los botones es un poco más tedioso que las etiquetas de anclaje.

El siguiente es el CSS para el menú.

```
body {  
    font-family: arial, sans-serif;  
    font-size: 16px;  
    -webkit-user-select: none;  
    -khtml-user-select: none;  
    -moz-user-select: none;  
    -ms-user-select: none;  
    user-select: none;  
    cursor: default;  
}  
  
canvas {  
    position: absolute;  
    top: 0;  
    left: 0;  
    border: 1px solid black;  
    z-index: 1;  
    width: 600px;  
    height: 360px;  
    -webkit-box-sizing: border-box;  
    -moz-box-sizing: border-box;  
    box-sizing: border-box;  
}  
  
ul {  
    list-style: none;  
    padding: 0;  
    margin: 0  
}  
li {  
    padding: 10px 0;  
}  
  
a {  
    text-decoration: none;  
    color: inherit;  
    text-shadow: -1px -1px 0 #000, 1px -1px 0 #000, -1px 1px 0 #000, 1px 1px 0 #000;  
}
```

En primer lugar, damos cada elemento en la página de tamaño de caja: borde-cuadro. Esto asegura que los tamaños que declaramos en CSS permanecen ese tamaño, lo cual es muy útil.

Después de establecer la imagen de fondo para el menú, establecemos el tamaño del menú div a aproximadamente la mitad del tamaño del lienzo y use position: absolute; izquierda: 0; derecha: 0; top: 0; fondo: 0; para centrar verticalmente el menú al juego.

Por último, creamos el estilo para el título del juego y los botones. La propiedad text-shadow se utiliza para dar al texto blanco un borde negro para que se sienta mejor contra el degradado púrpura.

Con el menú en su lugar, ahora necesitamos escribir el JavaScript para iniciar el juego cuando el jugador hace clic en el botón Reproducir.

```
/*
 * Load the main menu
 */
assetLoader.finished = function() {
    mainMenu();
}
/**
 * Show the main menu after loading all assets
 */
function mainMenu() {
    $('#main').show();
}

/**
 * Click handlers for the different menu screens
 */
$('.play').click(function() {
    $('#menu').hide();
    startGame();
});
```

Debido a que ya no estamos iniciando el juego una vez que todos los activos se han cargado, necesitamos cambiar la función que se llama en assetLoader.finished (). La nueva función mainMenu muestra el menú cuando todos los activos están cargados de modo que no podemos pulsar el botón de reproducción antes de que el juego esté listo.

A continuación, crear un controlador de clic en el botón de reproducción que oculta el menú y, a continuación, inicia el juego.

- La pantalla del juego

A continuación, crearemos el juego sobre la pantalla que permitirá al jugador reiniciar el juego si muere. Lo pondremos justo después de la etiqueta de canvas.

```
</div>
<canvas id="canvas" width="800" height="480"></canvas>
<div id="game-over">
  <h2>Forrest! te dije que corrieras :C <span id="score"></span> metros!</h2>
  <a href="#" class="restart">Jugar de nuevo?</a>
</div>
<script type="text/javascript" src="kandi.js"></script>
</body>
</html>
```

```
#game-over {
  display: none;
  text-align: center;
  padding-top: 92px;
  z-index: 7;
  width: 600px;
  height: 360px;
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
  overflow: auto;
  margin: auto;
  position: absolute;
}
```

```
/**
 * End the game and restart
 */
function gameOver() {
  stop = true;
  $('#score').html(score);
  $('#game-over').show();
  assetLoader.sounds.bg.pause();
  assetLoader.sounds.gameOver.currentTime = 0;
  assetLoader.sounds.gameOver.play();
}

/**
 * Click handlers for the different menu screens
 */
```

En el juego a través de la pantalla, vamos a dejar que el jugador sabe hasta dónde llegaron y luego dejar que el reiniciar el juego. Mediante el uso de una etiqueta span, podemos poner la puntuación del juego en el texto cuando el jugador muere.

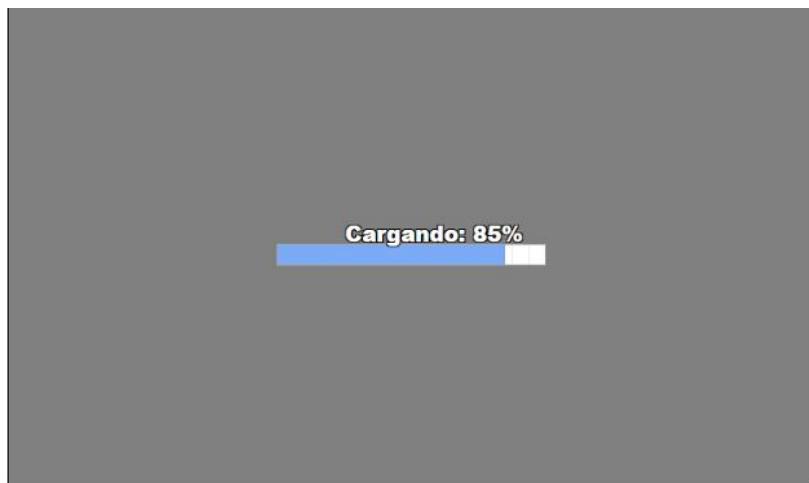
A diferencia de la pantalla de menú, no queremos alinear verticalmente el juego a través de la pantalla al juego. Esto es porque el texto es más fácil de leer si está delante del cielo azul claro y no delante de la de las colinas verdes del fondo. Así

que vamos a añadir el relleno a la parte superior hasta que esté en un nivel que se ve bien.

- La Pantalla de Carga

Es bueno que el jugador sepa que el juego está cargando los activos para que no crean que el juego está roto al iniciarlos. Es aún mejor mostrar al jugador el progreso de esta carga para que puedan juzgar cuánto tiempo va a tomar.

Para ello, crearemos una simple pantalla de carga con una barra de progreso para indicar cuántos activos se han cargado actualmente. Lo agregaremos justo antes del menú principal div.



```
</head>
<body>
  <div class="wrapper">
    <div class="sound sound-off"></div>
    <div id="menu">
      <div id="progress">
        <div id="percent">Cargando: <span id="p"></span></div>
        <progress id='progress-bar' value = '0'></progress>
      </div>
    </div>
  </div>
</body>
```

```
}

#progress-bar {
    position: absolute;
    top: 50%;
    left: 50%;
    margin-left: -100px;
    width: 200px;
    z-index: 3;
}
```

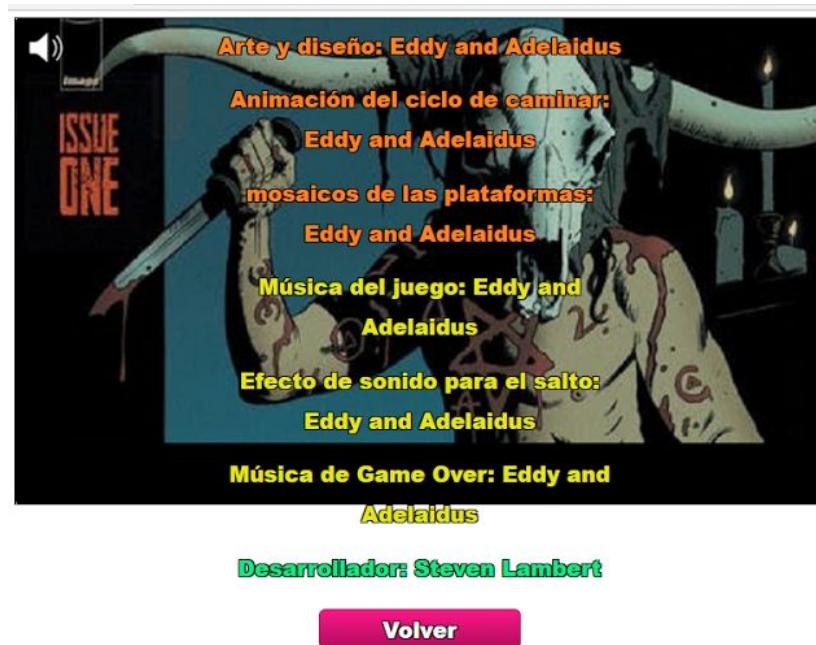
Estamos usando un elemento de progreso HTML5 para obtener una barra de progreso simple. Esto nos permite obtener un simple cargador de progreso sin demasiado trabajo.

Dado que queremos saber cuándo se ha cargado un activo, modificaremos la función assetLoaded () dentro del cargador de recursos para llamar a la función progress () si existe. Dentro de esta función, determinaremos el porcentaje de activos cargados en el total de activos y estableceremos el valor de la barra de progreso y el texto en este valor.

También tenemos que ocultar la pantalla de carga una vez que todos los activos se han cargado, por lo que vamos a modificar la función mainMenu () para hacer eso antes de mostrar el menú. Por último, agregamos la clase principal al menú div que muestra la imagen de fondo del menú principal en lugar de la imagen de la pantalla de carga gris.

- La pantalla de crédito

La última pantalla que vamos a agregar es la pantalla de crédito.



Cada vez que utilices activos que no hiciste (arte, música, etc.), deberías darle crédito a la persona que los creó en tu juego. La persona que hizo el activo determina cómo desea ser atribuido, por lo que tendrá que cumplir con lo que piden para poder utilizarlo en su juego.

Por ejemplo, me gusta usar opengameart.org para mis necesidades de arte y música. Si nos fijamos en los azulejos de plataformas utilizados para el juego, verá que el autor desea un enlace a su sitio web, aunque dicen que no es obligatorio para este activo. Me gusta atribuir siempre a los autores de los activos que utilizo porque creo que es buena forma.

También es una buena idea de credito a ti mismo, así en algún lugar de su juego para que otros puedan encontrar si les gusta su juego.

Vamos a agregar el menú de crédito justo después del menú principal y justo antes del lienzo.

Hemos dividido la pantalla de crédito en tres grupos: los que proporcionaron obras de arte, los que hicieron la música y yo como el desarrollador. Cada grupo tiene su

propio color para hacer obvio que hay tres grupos.

También necesitamos proporcionar un botón de retroceso para que los jugadores puedan volver al menú principal desde la pantalla de créditos.

El JavaScript sólo muestra u oculta el menú principal y la pantalla de crédito en función de qué botones se han hecho clic.

- Añadir música

Ahora que tenemos todos los menús para nuestro juego, lo último que tenemos que hacer es agregar música y efectos de sonido.

Normalmente, cuando estoy buscando música para ir con un juego, abro el juego en su propia ventana para que pueda escuchar la música mientras el juego está jugando. Esto me ayuda a determinar si la música se ajusta al estado de ánimo y el tema del juego.

Pude encontrar una divertida pista de fondo de 8 bits que pensé que funcionó muy bien para el juego en opengameart.org, así que la usaremos para nuestro juego.

Lo primero que tenemos que hacer es agregar toda la música al cargador de activos para que podamos usarlos en el juego.

IMAGEN

Cuando llamamos a `downloadAll()`, el cargador de activos cargará ahora todas las imágenes, así como todos los archivos de audio. Loading Audio es muy similar a cargar imágenes, con una gran diferencia: saber cuándo se ha cargado el audio.

En mi anterior intento de cargar música, tuve que recurrir a una comprobación `setInterval` de cada fuente de audio para ver si había cargado. Más tarde descubrí una manera de hacer esto con una llamada de evento similar a `onload` para imágenes. Al escuchar el evento de `canplay` de un objeto de audio, podría comprobar el `readyState` para asegurarse de que estaba realmente cargado.

Dado que el evento de `canplay` puede disparar varias veces para un único objeto de Audio, es importante comprobar que el audio no se ha cargado al comprobar su propiedad de estado. Si acaba de comprobar el `readState` del audio, podría cargarse dos veces y podría tener problemas más tarde si el audio que pensaba que estaba listo estaba, de hecho, todavía cargando.

A continuación, tocaremos los sonidos en el momento apropiado durante el juego.

La música de fondo cuando el jugador comienza el juego, el juego sobre la música cuando el jugador muere, y el sonido saltando cuando el jugador salta.

Lo último que queremos hacer es añadir una forma de silenciar todos los sonidos del juego. Si hubo una cosa que aprendí de mi último juego, fue que la reproducción automática de música sin ninguna manera de apagarla es lo peor! Así que esta vez, arreglaremos ese problema.

Otra buena característica que vamos a añadir es la capacidad del juego para saber si el usuario tenía la música en o fuera de la última vez que jugaron. Para ello, usaremos el almacenamiento local para almacenar un booleano que representa si la música está activada o desactivada.

Vamos a añadir un ícono en la esquina superior izquierda del juego que muestra el estado de la música, y vamos a permitir que el jugador haga clic en este ícono para alternar la música. Vamos a añadir el html justo antes de la div menú.

Lo primero que hacemos en JavaScript es determinar si el navegador del reproductor soporta almacenamiento local. Si lo hace, seguimos adelante y leemos la propiedad kandi.playSound para determinar si la música está activada (true) o desactivada (false). Utilizamos la variable playSound para guardar esta configuración.

Dado que el almacenamiento local es una lista de pares clave-valor, es posible que tenga que anotar cualquier nombre de clave que agregue al almacenamiento local para que no anule ninguna clave de otros juegos. Sólo tienes que hacer esto si planeas configurar todos tus juegos en el mismo subdominio, como yo, ya que el navegador ya tiene namespaces cada origen automáticamente.

Otra cosa importante a tener en cuenta sobre el almacenamiento local es que almacena todos los valores como cadenas. Esto significa que si almacena un booleano en el almacenamiento local, se vuelve a salir como una cadena. Por eso, cuando miramos el valor kandi.playSound, lo comparamos con la cadena "true" y no con el valor booleano true.

Una vez que se hayan cargado todos los activos, estableceremos la propiedad silenciada de todos los objetos Audio para que coincida con la preferencia del usuario. Puesto que estamos utilizando playSound === true para significar que el usuario quiere sonido, necesitamos configurar la propiedad silenciada en el contrario de eso.

Lo último que hacemos es añadir un controlador de clics al ícono que alterna las clases de sonido y de sonido (para mostrar los diferentes estados de los iconos), establece la variable de almacenamiento local para que coincida con el estado del

ícono y, a continuación, silencia o silencia todos el juego suena basado en la nueva preferencia.

Con el audio añadido al juego, nuestro juego está terminado y listo para jugar.

