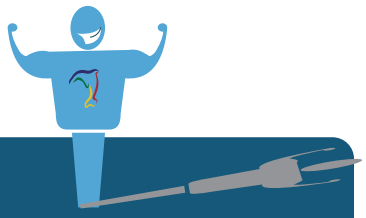
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

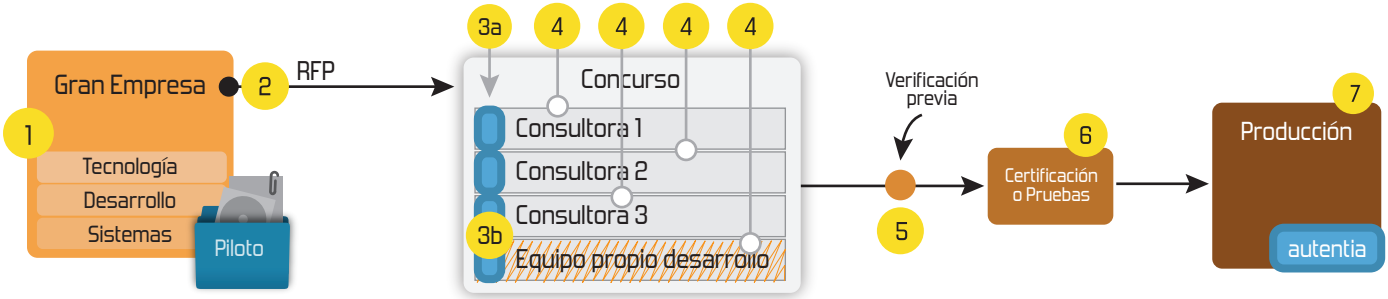
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Miguel Arlandy Rodríguez

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2012-04-10

Creando un videojuego con HTML5 y Javascript.

0. Índice de contenidos.

1. Introducción

Si decimos que HTML5 supone una verdadera revolución, en lo que a desarrollo de aplicaciones web se refiere, no estamos descubriendo nada nuevo. Sus nuevas características abren un inmenso abanico de posibilidades que nos permiten hacer verdaderas "diabluras".

En este tutorial vamos a aprovechar toda la potencia que nos ofrecen HTML5 y Javascript para crear un videojuego (con malo final incluido :).

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.2 Ghz Intel Core I7, 8GB DDR3).
- Sistema Operativo: Mac OS Snow Leopard 10.6.7
- Entorno de desarrollo: Intellij Idea 11 Ultimate.
- Mozilla Firefox 11.
- Google Chrome 18.
- Safari 5.1.
- Github

3. El escenario.

Vamos a crear un videojuego que funcione correctamente en los siguientes navegadores: Firefox, Chrome y Safari. El juego será un clásico juego de navecitas donde debemos matar a una serie de bichos para enfrentarnos con un "Jefe final". Como contenedor del juego utilizaremos el nuevo elemento de HTML5: canvas.

Para ello nuestro juego debe cumplir con los siguientes requisitos:

- Debe funcionar correctamente en: Mozilla Firefox, Google Chrome y Safari. En IE no funciona :-(
- El videojuego será un mata-marcianos
- Manejaremos una medusa (que hará de nave espacial) y que disparará (verticalmente) a los enemigos que le vayan saliendo.
- Los enemigos atacarán a nuestra medusa con disparos. Si un disparo impacta en la medusa morirá (perderá una vida). Si un enemigo colisiona con la medusa también morirá.
- Por cada "bicho" que mate el jugador incrementará su marcador de puntos.

- Si el jugador no mata a un malo, simplemente no sumará los puntos que le corresponderían por hacerlo.
- Nuestro héroe tendrá un número limitado de vidas. Cada vez que muera se le restará una vida hasta que se quede sin ninguna.
- Después de que nos ataquen todos los enemigos deberemos matar a un malvado "Jefe Final". Si conseguimos matarlo (o que no nos mate) nos habremos pasado el juego.
- El juego deberá guardar un histórico con las mejores puntuaciones del jugador haciendo uso de Local Storage.

4. Los actores.

Pues bien vamos a presentar a los actores de nuestro juego. En primer lugar "la medusa", que será el bueno del juego. Quien tiene que salvar al mundo de las malvadas hordas de bichos que quieren dominarlo. Perdonadme esta licencia tan "friki", es que a veces me emociono... :-P

El jugador manejará a la medusa con las teclas derecha e izquierda para el movimiento y con el espacio para disparar.

Como hemos dicho, nuestra medusa dispara a los enemigos, por lo que otro actor del juego será el disparo que realice.

Durante la primera parte del juego, nos atacará una horda de enemigos a los que deberemos destruir o, al menos, evitar que nos destruyan. Los enemigos descenderán verticalmente haciendo un movimiento de zig-zag.

Y, al igual que nuestra medusa, los bichos también disparan.

Cualquier matamarcianos que se precie debe tener un Malo Final. Nuestro juego no podía ser menos, así que este es nuestro último enemigo.

Pues estos son los actores que interactuarán en el juego. Ahora vamos a ver cómo se representan y se les añaden comportamientos

5. Representando a los actores.

Representaremos a nuestra medusa con un objeto "Player". El objeto contendrá el estado de diferentes propiedades como serán, la posición de la medusa en la pantalla (canvas), el número de vidas que le quedan, los puntos que lleva, si le han matado...

Además se añaden en forma de métodos los comportamientos de disparar (método privado "shoot"), de hacer algo cuando el jugador pulse una tecla (método público "doAnything") y de matar al jugador (método público "killPlayer").

```
function Player(life, score) {
    var settings = {
        marginBottom : 10,
        defaultHeight : 66
    };
    player = new Image();
    player.src = 'images/bueno.png';
    player.posX = (canvas.width / 2) - (player.width / 2);
    player.posY = canvas.height - (player.height == 0 ? settings.defaultHeight : player.he
    player.life = life;
    player.score = score;
    player.dead = false;
    player.speed = playerSpeed;

    var shoot = function () {
        if (nextPlayerShot < now || now == 0) {
            playerShot = new PlayerShot(player.posX + (player.width / 2) - 5 , player.posY
            playerShot.add();
            now += playerShotDelay;
            nextPlayerShot = now + playerShotDelay;
        } else {
            now = new Date().getTime();
        }
    };

    player.doAnything = function() {
        if (player.dead)
            return;
        if (keyPressed.left && player.posX > 5)
            player.posX -= player.speed;
        if (keyPressed.right && player.posX < (canvas.width - player.width - 5))
            player.posX += player.speed;
        if (keyPressed.fire)
            shoot();
    };

    player.killPlayer = function() {
        if (this.life > 0) {
            this.dead = true;
            evilShotsBuffer.splice(0, evilShotsBuffer.length);
            playerShotsBuffer.splice(0, playerShotsBuffer.length);
            this.src = 'images/bueno_muerto.png';
            createNewEvil();
            setTimeout(function () {
                player = new Player(player.life - 1, player.score);
            }, 500);

```

Autor

» Trabajando con Intellij IDEA 11

» JsTestDriver: testea tu código Javascript

» Cómo forzar al navegador a que descargue recursos estáticos cuando los tiene cacheados.

» Spring Integration: Ejemplo completo.

» Introducción a Spring Integration.

Categorías del Tutorial

HTML5

Javascript / JQuery

Últimas ofertas de empleo

```
        } else {
            saveFinalScore();
            youLoose = true;
        }
    };

    return player;
}
```

De la misma manera representaremos los disparos. Como hemos dicho, disparan tanto los bichos como nuestra medusa, así que parece un buen monento para usar herencia y reciclar las propiedades y comportamientos comunes de un disparo (tanto de medusa como de bicho) en un objeto (Shot), y extender las propiedades y comportamientos concretos en un objeto para el disparo de la medusa (PlayerShot) y otro para el disparo de los malos (EvilShot).

Los disparos, tanto de los bichos como de nuestra medusa, tienen determinadas características comunes como son: la posición en el canvas, un identificador, una imagen y el comportamiento de añadirse o eliminarse a un buffer que será gestionado para pintar la pantalla.

Tienen como diferencias, la imagen que los representa y el comportamiento de si han impactado contra el jugador (EvilShot) o contra un bicho malo (PlayerShot).

```
function Shot( x, y, array) {
    this.posX = x;
    this.posY = y;
    this.image = new Image();
    this.speed = shotSpeed;
    this.identifier = 0;
    this.add = function () {
        array.push(this);
    };
    this.deleteShot = function (idendificador) {
        arrayRemove(array, idendificador);
    };
}

function PlayerShot (x, y) {
    Object.getPrototypeOf(PlayerShot.prototype).constructor.call(this, x, y, playerShotsBu
    this.image.src =  'images/disparo_bueno.png';
    this.isHittingEvil = function() {
        return (!evil.dead && this.posX >= evil.posX && this.posX <= (evil.posX + evil.ima
            this.posY >= evil.posY && this.posY <= (evil.posY + evil.image.height));
    };
}

PlayerShot.prototype = Object.create(Shot.prototype);
PlayerShot.prototype.constructor = PlayerShot;

function EvilShot (x, y) {
    Object.getPrototypeOf(EvilShot.prototype).constructor.call(this, x, y, evilShotsBuffer
    this.image.src =  'images/disparo_malo.png';
    this.isHittingPlayer = function() {
        return (this.posX >= player.posX && this.posX <= (player.posX + player.width)
            && this.posY >= player.posY && this.posY <= (player.posY + player.height));
    };
}

EvilShot.prototype = Object.create(Shot.prototype);
EvilShot.prototype.constructor = EvilShot;
```

El comportamiento del bicho malo y del jefe final es una mezcla de los dos casos anteriores, tenemos un objeto Enemy del que extienden otros dos: Evil y FinalBoss. El que quiera indagar más puede verlo en el código fuente del juego.

6. El movimiento y las interacciones.

LLegados a este punto ya tenemos claro quienes son los actores que intervienen en el juego pero ¿cómo interactuan entre ellos?. ¿Cómo se mueven por el canvas?.

Pues es muy sencillo. Supongamos el caso de un computador. El computador trabaja en función a unos ciclos de reloj, según el cual, en cada ciclo se actualiza el estado de los bits que maneja. Pues en el juego es exáctamente lo mismo. Tenemos una función requestAnimFrame que será quien nos marque los pulsos. Después de cada pulso nosotros actualizamos el estado del canvas.

```
// nos marca los pulsos del juego
window.requestAnimFrame = (function () {
    return  window.requestAnimationFrame        ||
        window.webkitRequestAnimationFrame  ||
        window.mozRequestAnimationFrame     ||
        window.oRequestAnimationFrame       ||
        window.msRequestAnimationFrame      ||
        function ( /* function */ callback, /* DOMElement */ element) {
            window.setTimeout(callback, 1000 / 60);
        };
})();
```

Una vez que ya tenemos nuestro marcador de pulsos, lo que hacemos es, entre pulso y pulso, realizar las acciones propias del juego como son: actualizar la posición de los actores, comprobar las interacciones entre ellos y llevar a cabo la lógica de negocio correspondiente.

```
// el bucle del juego
function anim () {
    loop(); // hacemos cosas
    requestAnimFrame(anim); // esperamos al nuevo ciclo
}
anim();

// lo que hacemos en cada ciclo
function loop() {
    update(); // actualizamos posiciones, comprobamos interacciones
    draw(); // pintamos a los actores
}
```

Para comprobar las interacciones y actualizar a los actores usamos el método update que comprobará cosas necesarias en cada pulso como son:

- Si nos han matado
- Si nos hemos pasado el juego
- Si nos ha dado un disparo
- Si hemos dado al malo
- Si el jugador está pidiendo a la medusa que se mueva o que dispare mediante la pulsación de las teclas

```
function update() {

    drawBackground(); // pintamos el fondo de pantalla (del canvas)

    // comprobamos si hemos terminado el juego
    if (congratulations) {
        showCongratulations();
        return;
    }

    // comprobamos si nos han matado
    if (youLoose) {
        showGameOver();
        return;
    }

    // pintamos al bueno y al malo
    bufferctx.drawImage(player, player.posX, player.posY);
    bufferctx.drawImage(evil.image, evil.posX, evil.posY);

    // actualizamos al enemigo
    updateEvil();

    // actualizmaos los disparos (del bueno)
    for (var j = 0; j < playerShotsBuffer.length; j++) {
        var disparoBueno = playerShotsBuffer[j];
        updatePlayerShot(disparoBueno, j);
    }

    // comprobamos si el bueno y el malo se han chocado
    if (isEvilHittingPlayer()) {
        player.killPlayer();
    } else {
        // actualizamos los disparos del malo
        for (var i = 0; i < evilShotsBuffer.length; i++) {
            var evilShot = evilShotsBuffer[i];
            updateEvilShot(evilShot, i);
        }
    }

    // pintamos el marcador y las vidas
    showLifeAndScore();

    // comprobamos las acciones que debe realizar la medusa en función de las pulsaciones
    playerAction();
}
```

Como dije anteriormente, si alguien quiere ver qué hacen las funciones a las que llama la función "update" puede hacerlo echándole un ojo código fuente del juego, aunque lo que hacen es lo que su propio nombre indica que hacen (Clean code: Principle of Least Surprise).

7. Los controles.

Como en cualquier juego, el usuario deberá interactuar con el personaje que maneja. En nuestro caso deberá manejar a la medusa para esquivar los disparos de los bichos y para poder dispararles a ellos. Los controles serán los siguientes:

- Flecha izquierda: mueve la medusa a la izquierda.
- Flecha derecha: mueve la medusa a la derecha.
- Espacio: disparar (se puede dejar presionado).

Como vimos en los puntos anteriores, el juego se actualiza en base a unos pulsos. Por tanto debemos comprobar durante cada pulso si el usuario está presionando alguna de las teclas que manejan a la medusa para que ésta actúe en consecuencia con la orden que se le está transmitiendo. El método "doAnything" del objeto "Player" que vimos en el punto 5 será el encargado de llevar a cabo la orden del usuario y la función "playerAction", que se invoca desde la función "update" que vimos en el punto anterior será la encargada de invocarlo en cada pulso.

```
// las teclas
var keyMap = {
        left: 37,
        right: 39,
        fire: 32     // tecla espacio
    };

//registramos los eventos de pulsaciones de teclado
addListener(document, 'keydown', keyDown);
addListener(document, 'keyup', keyUp);
function addListener(element, type, expression, bubbling) {
    bubbling = bubbling || false;

    if (window.addEventListener) { // Standard
        element.addEventListener(type, expression, bubbling);
    } else if (window.attachEvent) { // IE
        element.attachEvent('on' + type, expression);
    }
}

// indicamos qué tecla está presionada en función del evento
function keyDown(e) {
    var key = (window.event ? e.keyCode : e.which);
    for (var inkey in keyMap) {
        if (key === keyMap[inkey]) {
            e.preventDefault();
            keyPressed[inkey] = true;
        }
```

```
    }
}

function keyUp(e) {
    var key = (window.event ? e.keyCode : e.which);
    for (var inkey in keyMap) {
        if (key === keyMap[inkey]) {
            e.preventDefault();
            keyPressed[inkey] = false;
        }
    }
}

// la medusa actuará según la tecla pulsada
function playerAction() {
    player.doAnything();
}
```

8. Guardando las mejores puntuaciones.

Recordemos que un requisito de nuestro juego era que, una vez finalizada la partida, debíamos grabar la puntuación del jugador haciendo uso de la nueva caracterísitica de HTML5 Local Storage (almacenamiento local).

Lo que haremos será, partiendo de un parámetro configurable que nos indicará el número de mejores puntuaciones que debemos mostrar al usuario:

- Guardar la puntuación del jugador y la fecha y hora en la que esta se produjo.
- Actualizar la lista de las X mejores puntuaciones
- Eliminar las puntuaciones que no estén entre las mejores (para no almacenar datos innecesarios en el navegador)
- Mostrar la lista de puntuaciones actualizada.

```
function saveFinalScore() {
    localStorage.setItem(getFinalScoreDate(), getTotalScore());
    showBestScores();
    removeNoBestScores();
}

function showBestScores() {
    var bestScores = getBestScoreKeys();
    var bestScoresList = document.getElementById('puntuaciones');
    if (bestScoresList) {
        clearList(bestScoresList);
        for (var i=0; i < bestScores.length; i++) {
            addListElement(bestScoresList, bestScores[i], i==0?'negrita':null);
            addListElement(bestScoresList, localStorage.getItem(bestScores[i]), i==0?'negr
        }
    }
}

function removeNoBestScores() {
    var scoresToRemove = [];
    var bestScoreKeys = getBestScoreKeys();
    for (var i=0; i < localStorage.length; i++) {
        var key = localStorage.key(i);
        if (!bestScoreKeys.containsElement(key)) {
            scoresToRemove.push(key);
        }
    }
    for (var j = 0; j < scoresToRemove.length; j++) {
        var scoreToRemoveKey = scoresToRemove[j];
        localStorage.removeItem(scoreToRemoveKey);
    }
}
```

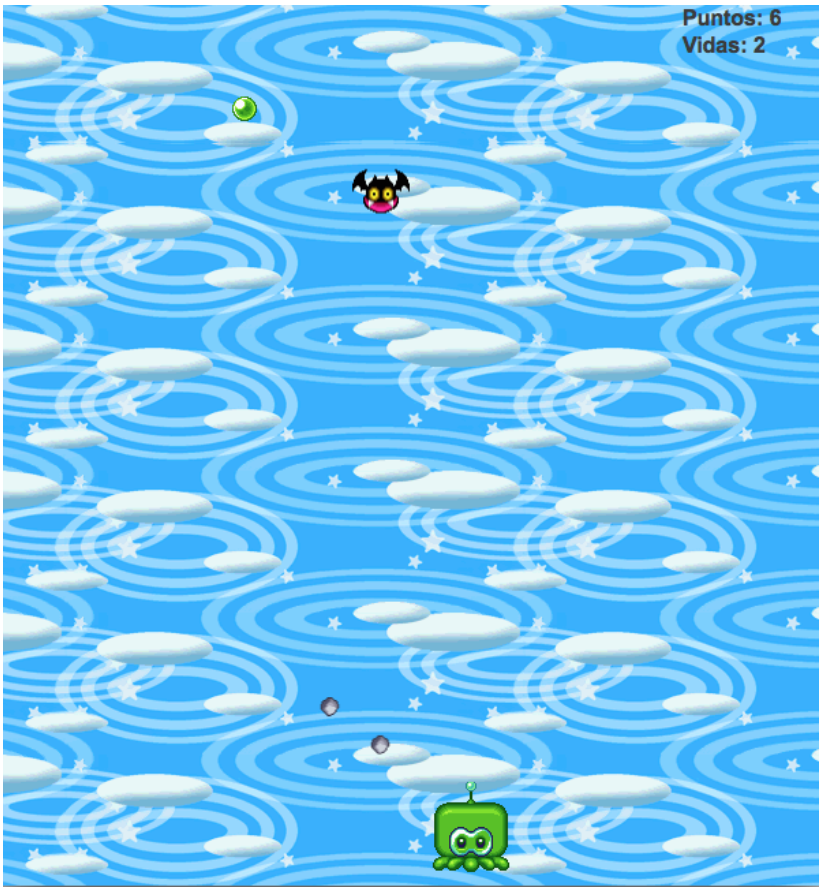
Una vez tenemos esto, mostraremos al usuario las mejores puntuaciones.

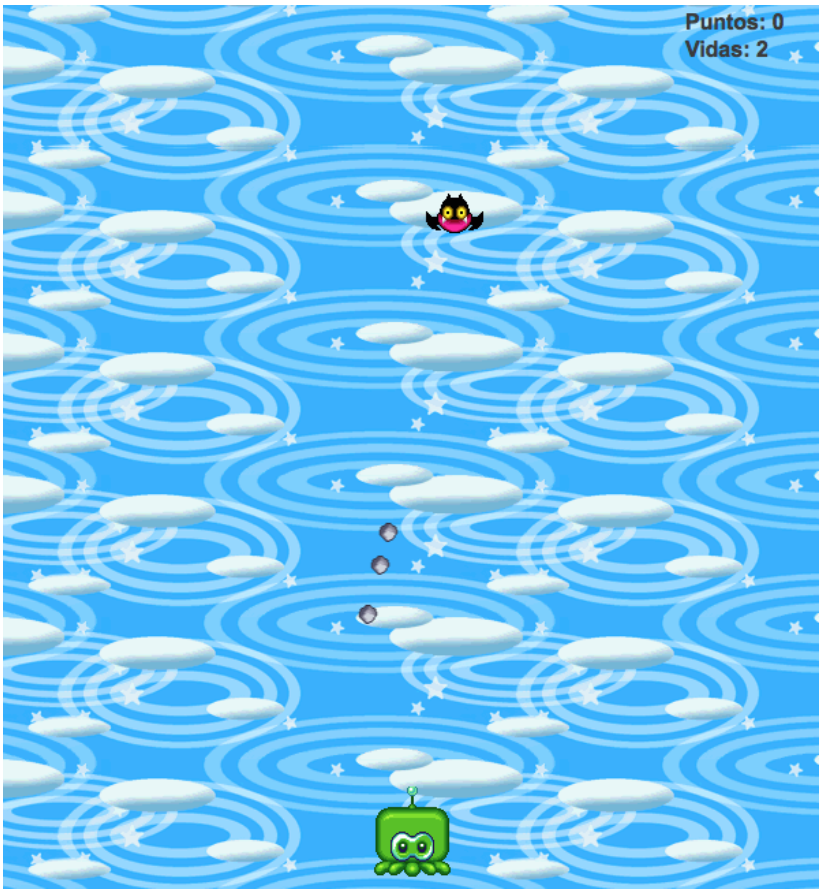
MEJORES PUNTUACIONES

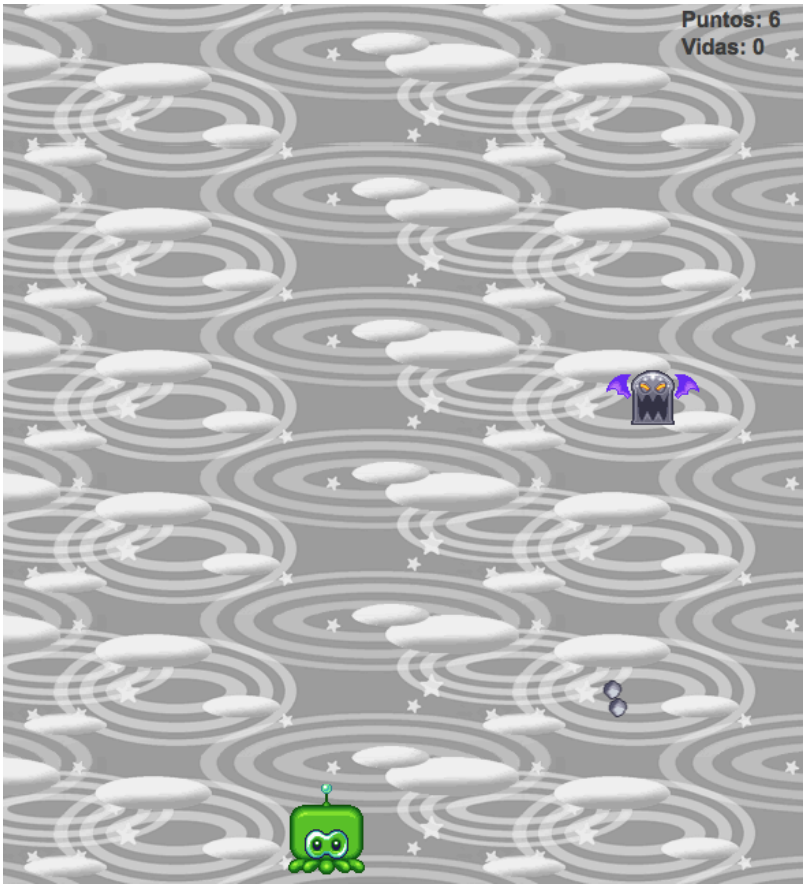
Fecha	Puntos
02/04/2012 18:58:59	**81**
01/04/2012 22:42:00	71
04/04/2012 20:58:59	66
03/04/2012 19:42:00	59
04/04/2012 21:24:20	37

9. El resultado final.

Pues básicamente con esto que hemos comentado anteriormente tendríamos nuestro videojuego HTML5 + Javascript.


Puntos: 6
Vidas: 2


Puntos: 0
Vidas: 2



PULSA AQUÍ PARA JUGAR

10. Referencias.

- El juego
- Código fuente del juego
- HTML5: Local Storage
- window.requestAnimFrame
- Creación de juegos en JavaScript (la presentación que inspiró este tutorial).
- Imágenes de juegos de toda la vida

11. Conclusiones.

Explorando las capacidades que nos ofrece HTML5 hemos visto cómo crear nuestro propio videojuego con Javascript en menos de 600 líneas de código. Os animo a que sigais investigando sobre esta tecnología y la amplia variedad de posibilidades que nos ofrece.

Al que le haya gustado esto de los juegos, puede descargarse sin ningún problema el código fuente de este videojuego y hacer con él lo que le de la "real gana". Se pueden hacer infinidad de mejoras como: añadir sonido, crear distintas fases, hacer que caigan "bolas" que al cogerlas obtengamos nuevos disparos o más vidas, hacer que funcione en Internet Explorer, usar imágenes con sprites, etc, etc, etc...

Ahora que Adobe ha anunciado que va a dejar de dar soporte a Flash, ¿será este el futuro de los conocidos como "juegos de navegador"?

Pues nada, con esto termino. Como se suele decir en estos casos... GAME OVER

Espero que este tutorial os haya sido de ayuda. Un saludo.

Miguel Arlandy

marlandy@autentia.com

Twitter: @m_arlandy

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» **Registrate** y accede a esta y otras ventajas «