

Cocos2d-html5



Grupo:

Danya Barros

Gabriel Rilo

Fernando Velazquez

Taller de Videojuegos 2D 2013
Tecnologo en Informática

Introducción a Framework Cocos2d-html5

Cocos2d-html5 es un framework web, de código abierto para juegos en 2D , publicado bajo licencia MIT. Es una versión HTML5 de proyecto Cocos2d-x. El enfoque para el desarrollo Cocos2d-html5 gira entorno a hacer plataformas cruzadas cocos2d entre navegadores y aplicaciones nativas. Como característica principal el framework proporciona que los juegos pueden ser escritos en JavaScript, usando la API que es totalmente compatible con el de Cocos2d-x javascript. Un proyecto Cocos2d-html5 se puede ejecutar fácilmente en los navegadores que soportan HTML5. Para la realización del juego utilizamos la version 2.2 del framework.

Sitio: <http://www.cocos2d-x.org/wiki/Cocos2d-html5>

Descargas versión para html5: <http://www.cocos2d-x.org/download>

Box2D

Box2D es un motor físico en dos dimensiones, es código abierto y esta desarrollado en C++ . Box2D es desarrollado por Erin Catto y se distribuye bajo licencia zlib .

Principales características - (Extraído de: <http://box2d.org/about/>)

COLLISION

- Continuous collision detection
- Contact callbacks: begin, end, pre-solve, post-solve
- Convex polyons and circles.
- Multiple shapes per body
- One-shot contact manifolds
- Dynamic tree broadphase
- Efficient pair management
- Fast broadphase AABB queries
- Collision groups and categories

PHYSICS

- Continuous physics with time of impact solver
- Persistent body-joint-contact graph
- Island solution and sleep management
- Contact, friction, and restitution
- Stable stacking with a linear-time solver
- Revolute, prismatic, distance, pulley, gear, mouse joint, and other joint types
- Joint limits, motors, and friction
- Momentum decoupled position correction
- Fairly accurate reaction forces/impulses

Cocos2d-html5 incluye box2dweb, una versión de la librería portada a javascript.

Sitio: <http://box2d.org/>

Sitio Box2dweb - <http://box2d-js.sourceforge.net/#about>

Instalación Cocos2d-html5

Lo primero que se debe hacer es descargar la ultima version disponible del framework, en nuestro caso utilizamos la version 2.2. Se debe descargar de la siguiente dirección:

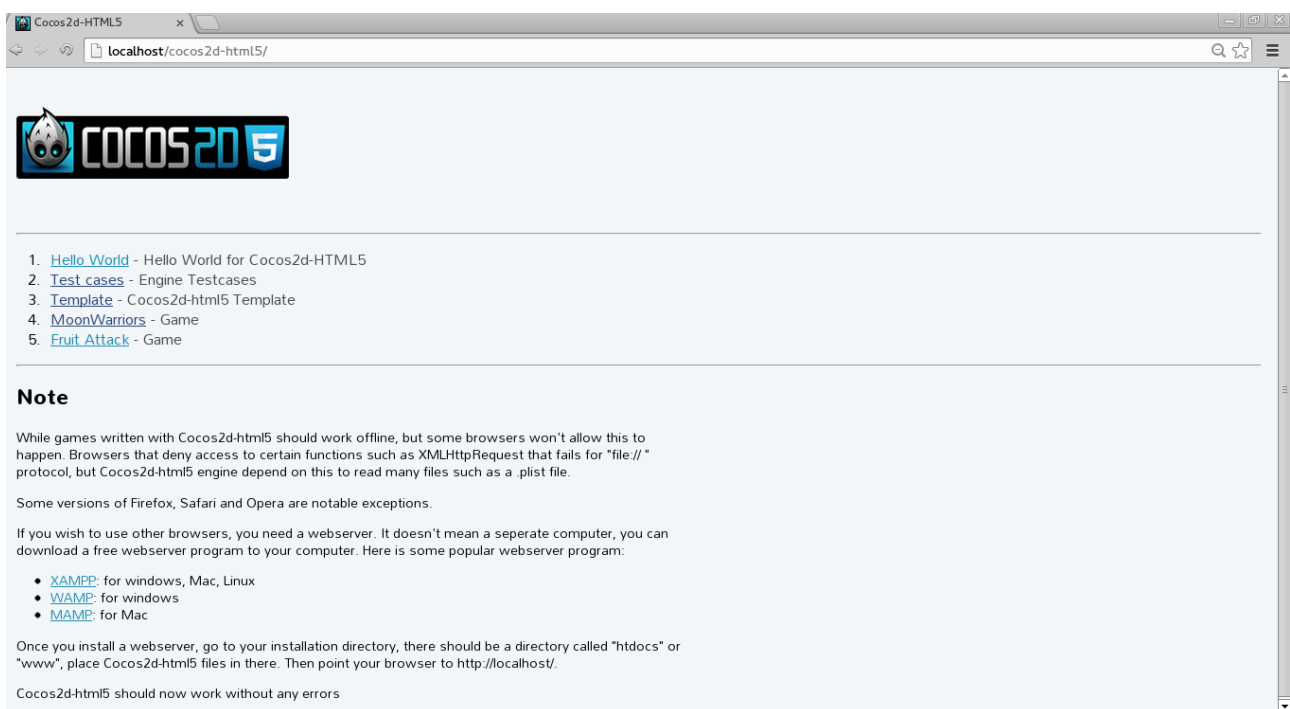
<http://www.cocos2d-x.org/download>

Dependiendo del sistema operativo utilizado se debe instalar un servidor web, en nuestro caso como utilizamos Linux, instalamos apache, en el siguiente enlace se pueden seguir los pasos para instalarlo en Ubuntu : http://www.guia-ubuntu.com/index.php/Servidor_web

Una vez que se corrobora el correcto funcionamiento del servidor web ingresando en cualquier browser a la dirección localhost, se procede con la instalación del framework descargado.

Se debe descomprimir el zip del framework y copiar todo el contenido a la carpeta del apache (por lo general /var/www).

Luego en cualquier browser ingresar a localhost/index.html y si todo va bien ser debería ver lo siguiente:



Comenzando el proyecto

Lo ideal para comenzar a trabajar en una aplicación con este framework es utilizar como base la carpeta **template** que viene dentro de la carpeta de dicho framework, y a partir se puede comenzar a desarrollar la aplicación.

Se recomienda seguir los pasos detallados a continuación:

- Crear una carpeta con el nombre del juego, en nuestro caso **killthewabbit**
- Copiar dentro de la carpeta creada la carpeta **template** que viene en el zip del framework
- Cambien copiar las carpetas **CocosDenshion**, **box2d**, **cocos2d**, **extensions**.

El contenido de la carpeta **killthewabbit** debe quedar de la siguiente forma:

```
killthewabbit
    CocosDenshion
    box2d
    cocos2d
    extensions
    template
```

Es necesario incluir los directorios **cocos2d**, **box2d** y **extensions** para poder hacer uso de todas las funcionalidades , sobretodo si se va a tener física en el juego, el directorio **box2d** es muy importante.

Luego el directorio **CocosDenshion** es incluido para manejar el audio en la aplicación. CocosDenshion es una biblioteca de audio. El objetivo principal es satisfacer las necesidades de audio de los juegos. CocosDenshion incluye un motor de baja latencia de sonido para la reproducción de efectos de sonido del juego.

Estructura de archivos dentro de la carpeta template

Se describen unicamente los archivos mas relevantes y en los cuales se debe trabajar

template
 res
 src
 cocos2d.js
 index.html
 main.js

index.html

En el archivo index.html es en donde esta definido el canvas del juego:

```
<canvas id="gameCanvas" width="320" height="480"></canvas>
```

también en donde se incluye la llamada al cocos2d por medio del tag:

```
<script src="cocos2d.js"></script>
```

cocos2d.js

El archivo cocos2d.js contiene el punto de inicio del juego. Es una gran función anónima que se define y se ejecuta en el momento.

Este archivo contiene una configuración básica del juego: si se va a utilizar alguna de las dos librerías físicas que cocos2d trae (box2d o chipmunk), si va a mostrar el número de frames por segundo , y el listado de archivos que conforma el juego.

La funcionalidad de este archivo en general no cambia entre proyectos: su objetivo es introducir al documento todos los scripts Javascript que definen al juego, incluyendo los propios de cocos2d. Al introducirlos al documento se cargan y al finalizar este proceso se ejecuta el script main.js.

En nuestro caso habilitamos el box2d para poder hacer uso de la física y deshabilitamos "showFPS", para que no muestre el contador de frames por segundo.

Inicio del archivo:

```
box2d:true,  
chipmunk:false,  
showFPS:false,  
loadExtension:true,  
frameRate:60,  
renderMode:0,
```

```
tag:'gameCanvas',  
engineDir: '../cocos2d/',  
//SingleEngineFile:",
```

También se define el nombre del elemento canvas que va a contener el juego, el mismo que se estableció en el index.html.

En la sección **appfiles** se deben agregar todos los archivos que se tengan dentro del directorio src.

En nuestro caso:

```
appFiles:[  
  'src/resource.js',  
  'src/MenuPrincipal.js',  
  'src/MenuSecundario.js',  
  'src/Win.js',  
  'src/Game.js',  
  'src/Creditos.js',  
  'src/Canon.js',  
  'src/Conejo.js',  
  'src/Level1.js',  
  'src/Sky.js',  
  'src/Plane.js',  
  'src/Explosion.js'  
]
```

main.js

El script main.js crea la Aplicación de cocos2d. La aplicación inicializa cosas que se accederán posteriormente por ejemplo el Director, es un objeto que se puede acceder desde cualquier parte del código para dar instrucciones del juego (por ejemplo, ir de una escena a otra).

Este script también hace la pre-carga del listado de recursos del juego para que la funcionalidad del juego no se vea afectada por recursos que aun no se han cargado, como pueden ser imágenes y las pistas de audio.

Al final de este script se define cual va a ser la escena con la que inicia el juego, en nuestro caso es el archivo MenuPrincipal.js que se encuentra en el directorio src, por lo tanto el código queda de a siguiente forma.

```
var myApp = new cocos2dApp(MenuPrincipal);
```

Directorio res

Dentro de este directorio en el subdirectorio **Normal** es en donde se almacenan las imágenes que luego oficiaran de sprites.

En el subdirectorio **music** se almacenan las pistas de audio que utiliza el juego y en el subdirectorio **fonts** las fuentes usadas.

En **res** es en donde se almacenan todo tipo de recursos multimedia que se deseen utilizar.

Directorio src

Dentro de este directorio se encuentran todos los archivos javascript que constituyen el juego. Lo ideal para una mejor performance es crear uno por cada sprite que se va a crear.

Las escenas de cocos2d también están cada una en un archivo diferente.

Un archivo que ya viene por defecto es el **resource.js**, en el cual se asocian los recursos que están en el directorio **res** a una variable que luego puede ser llamada desde cualquier lugar de la aplicación.

En nuestro caso:

```
var s_fondo = "menu.jpg";  
var s_bala = "bala.png";  
var s_canon = "canon.png";  
var s_conejo = "conejo.png";  
var s_pantalla = "pantalla.jpg";  
var s_sky = "sky.png";  
var s_plane = "plane.png";
```


Clases principales

Escena o Scene

Una escena representa una ventana para el jugador.

Sprite

El sprite es un recuadro relleno con alguna imagen, y tiene propiedades como posición, escalamiento y rotación.

Capa o Layer

La escena ya incluye por defecto una capa, pero los sprites se superponen unos a otros dependiendo del orden en el que son añadidos. Las capas permiten agrupar a los sprites con mayor facilidad. Cocos2d incluye algunos tipos especializados de capas que tienen funcionalidades particulares.

Menú

El menú es un tipo de capa que solo puede contener un tipo de objeto llamado MenuItem. Los MenuItem encapsulan funcionalidad básica para interfaces de usuario, botones.

Box2d World

World significa un mundo de la física con una colección de bodies, fixtures y constraints que interactúan entre sí. Box2D soporta la creación de múltiples worlds, pero esto no suele ser necesario o deseable.

//Funciones mas comunes box2D

```
var b2Vec2 = Box2D.Common.Math.b2Vec2
    , b2BodyDef = Box2D.Dynamics.b2BodyDef
    , b2Body = Box2D.Dynamics.b2Body
    , b2FixtureDef = Box2D.Dynamics.b2FixtureDef
    , b2World = Box2D.Dynamics.b2World
    , b2PolygonShape = Box2D.Collision.Shapes.b2PolygonShape
    , b2CircleShape = Box2D.Collision.Shapes.b2CircleShape;
```

//Creación del mundo con gravedad 0,-10 y el booleano en true significa que los cuerpos pueden quedar en reposo si nada pasa

```
this.world = new b2World(new b2Vec2(0, -10), true);
```

En el juego utilizamos cuerpos box2d para establecer el piso del nivel, las cajas de los conejos y los márgenes del canvas.

Escenas

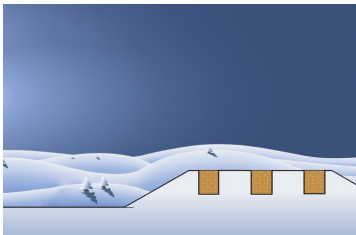
Las escenas en cocos2d ayudan a separar las funcionalidades que puede tener un juego. Por ejemplo, en nuestro caso tenemos un menú principal, otra para los créditos, otra escena para el juego en si, y otra que advierte el final del juego.

Una de las ventajas que ofrece cocos2d es el manejo automático de memoria, por lo que las escenas de cocos2d no solamente ayudan a organizar el código, también facilitan la utilización de recursos.

El manejo de la memoria funciona de forma que cuando se crea una escena es necesario tener que crear todos sus componentes: sprites, nodos, textos, etc. y agregarlos a la misma. Cuando se cambia de escena, no es necesario tener que destruir esos componentes explícitamente porque las referencias que hace esa escena saliente también son eliminadas.

Sprites

- Crear sprite para el fondo de pantalla del juego



1. Primero se debe copiar la imagen a res/Normal
2. Luego en resources.js agregarla como variable.
`var s_pantalla = "pantalla.jpg";`
3. Luego crear en src el archivo Level1.js con el siguiente contenido

```
var Level1Sprite = cc.Sprite.extend({  
    ctor:function(){  
        this._super();  
        this.initWithFile(s_pantalla);  
    }  
});
```

4. Por ultimo agregar la ruta al archivo en la sección appFiles de cocos2d.js

- Crear sprite para el cañon



1. Primero se debe copiar la imagen a res/Normal
2. Luego en resources.js agregarla como variable.
`var s_canon = "canon.png";`
3. Luego crear en src el archivo Canon.js con el siguiente contenido, ademas de cargarle la imagen al sprite se programa que siga el movimiento del mouse para poder apuntar hacia donde este posicionado el mismo en todo momento.

```
var CanonSprite = cc.Sprite.extend({
    _currentRotation: 0,
    ctor:function(){
        this._super();
        this.initWithFile(s_canon);
    },
    update:function(dt){
        this.setRotation(this._currentRotation);
    },
    handleMouseMove:function(touchLocation){

        angle = angle * (180/Math.PI);
        if (angle >= -30 && angle <= 50) {
            this._currentRotation = angle;
            canonAngle = angle;
        }
    }
});
var canonAngle = 0;
```

4. Por ultimo agregar la ruta al archivo en la sección appFiles de cocos2d.js

Ahora se puede crear la escena principal en donde se va a desarrollar el juego, en el ejemplo se muestra como definir el sprite pantalla como fondo y posicionar el sprite cañon dentro del juego.

Game.js

```
var GameLayer = cc.Layer.extend({
    _level1Sprite:null,
    init:function () {
        this._super();
        //Asi se obtiene el tamaño de la pantalla para luego posicionar los
        objetos
        var size = cc.Director.getInstance().getWinSize();

        // Aqui se llama al sprite previamente definido en Level1.js
        //Level1
        this._level1Sprite = new Level1Sprite();
        this.addChild(this._level1Sprite);
        this._level1Sprite.setPosition(cc.p(size.width,size.height));
        this._level1Sprite.setAnchorPoint(cc.p(1,1));

        //Aqui se llama al sprite previamente definido en Canon.js
        this._canonSprite = new CanonSprite();
        this.addChild(this._canonSprite);
        this._canonSprite.setPosition(cc.p(size.width/4-100,size.height/4-45));
        this._canonSprite.setAnchorPoint(cc.p(0.55,0.3));
        this.reorderChild(this._canonSprite,4);
        this._canonSprite.scheduleUpdate();

    }
});
```

```
var Game = cc.Scene.extend({
    onEnter:function () {
        this._super();
        var layer = new GameLayer();
        this.addChild(layer);
        layer.init();
    }
});
```

Ademas de estos sprites, se crearon los sprites de conejos, plane y sky.

Sprite Bala

Luego de la creación de un sprite con la imagen de la bala como se explico anteriormente, se le configura la física al cuerpo, para esto utilizamos Box2d.

Codigo:

```
//Funciones box2d
var b2Vec2 = Box2D.Common.Math.b2Vec2
, b2BodyDef = Box2D.Dynamics.b2BodyDef
, b2Body = Box2D.Dynamics.b2Body
, b2FixtureDef = Box2D.Dynamics.b2FixtureDef
, b2World = Box2D.Dynamics.b2World
, b2CircleShape = Box2D.Collision.Shapes.b2CircleShape;

//Creacion sprite bala
var sprite = cc.Sprite.create(s_bala, 0);
this.addChild(sprite,2,TAG_SPRITE_MANAGER);

//Posicion inicial desde donde es lanzada la bala
var posinicialx = 1.7;
var posinicialy = 2;
```

```
//Creacion de cuerpo Box2D con forma de Circulo , radio 0.15 y tipo dinamico
```

```
var bala = new b2CircleShape(0.15);

var bodyDef = new b2BodyDef();
bodyDef.type = b2Body.b2_dynamicBody;
bodyDef.position.Set(posinicialx, posinicialy);
bodyDef.userData = sprite;
var body = this.world.CreateBody(bodyDef);

var fixtureDef = new b2FixtureDef();
fixtureDef.shape = bala;
fixtureDef.density = 1.0;
fixtureDef.friction = 0.3;
fixtureDef.restitution = 0;
body.CreateFixture(fixtureDef);
```

La bala se crea en el lugar donde se haga click con el mouse gracias a la función OnMouseUp. Resta aplicarle a la bala una fuerza, impulso o velocidad para que se dispare.

Para ello utilizamos el siguiente código:

```
var velocidad = new Box2D.Common.Math.b2Vec2(0,0);
velocidad.x = p.x / PTM_RATIO - posinicialx + 2;
velocidad.y = p.y / PTM_RATIO - posinicialy + 2;
body.SetLinearVelocity(velocidad);
```

Este código nos permite calcular con que velocidad saldrá disparada la bala respecto a la ubicación del mouse.

Al momento de disparar la bala se calculan 6 segundos para que la misma explote. La explosión se genera como un evento en el sprite bala. También se agrega el efecto de sonido de explosión, seguido de la destrucción del sprite y eliminación del cuerpo físico box2d.

Código de la función `explotion`, aquí es donde se detecta la posición de la bala para ver si esta en la zona de algún conejo. También se lanza el evento de audio y la animación correspondiente a la explosión :

```
explotion:function(s, b) {  
  
    var p = b.GetPosition();  
    var x = p.x;  
    if ((x > 10.8) && (x < 12.35)){  
        this.eliminarConejo(this._conejoSprite1);  
    }  
    if ((x > 13.8) && (x < 15.3)){  
        this.eliminarConejo(this._conejoSprite2);  
    }  
    if ((x > 17) && (x < 18.2)){  
        this.eliminarConejo(this._conejoSprite3);  
    }  
    cc.AudioEngine.getInstance().playEffect("res/music/Explo.ogg", false);  
    var explosion = cc.BuilderReader.load("Explosion.ccbi");  
    s.addChild(explosion);  
    return explosion;  
}
```


Scrolling Background

Se utilizaron los sprites Plane.js y Sky.js para simular el pasaje de nubes y un avión cada cierto tiempo.

En un comienzo se intento realizar el efecto con una acción de cocos2d (MoveBy) pero no fue así finalmente ya que no era posible que una vez que las nubes terminaran de pasar lo volvieran a hacer de forma continua.

Lo que se hizo fue actualizar las posiciones de los sprites cada cierto tiempo para simular la sensación de movimiento.

Código de la función **onEnter** que se ejecuta al iniciar la escena, contiene el background scrolling.

```
onEnter:function() {
    this._super();

    var size = cc.Director.getInstance().getWinSize();
    //Clouds
    this._skySprite1 = new SkySprite();
    this.addChild(this._skySprite1);
    this._skySprite1.setPosition(size.width-200, size.height/2+125);

    this._skySprite2 = new SkySprite();
    this.addChild(this._skySprite2);
    this._skySprite2.setPosition(size.width-200, size.height/2+125);

    this._planeSprite = new PlaneSprite();
    this.addChild(this._planeSprite);
    this._planeSprite.setPosition(0, size.height/2+100);

    var self = this;
    this._skySprite1.setPositionX(-100);
    this._skySprite2.setPositionX(-1200);
```

```
setInterval(function() {  
  
    self._skySprite1.setPositionX(self._skySprite1.getPositionX() + 1);  
    self._skySprite2.setPositionX(self._skySprite2.getPositionX() + 1);  
  
    if (self._skySprite1.getPositionX() > 1200) {  
        self._skySprite1.setPositionX(-1200);  
    }  
    if (self._skySprite2.getPositionX() > 1200) {  
        self._skySprite2.setPositionX(-1200);  
    }  
}, 70);
```

```
setInterval(function() {
```

```
    self._planeSprite.setPositionX(self._planeSprite.getPositionX() +  
1);  
    if (self._planeSprite.getPositionX() > 1200) {  
        self._planeSprite.setPositionX(-500);  
    }  
}, 20);  
},
```

AudioEngine

Para la reproducción de música, efectos de sonido, etc. Cocos2d tiene una librería llamada SimpleAudioEngine.

Es necesario obtener una instancia de la misma:

```
audio = AudioEngine.getInstance();
```

Esto permite luego precargar sonidos de efectos y música a utilizar.

```
audio.preloadEffect("Explosion.ogg");  
audio.preloadMusic("LaTrampa.ogg");
```

De esta manera podemos reproducir los archivos cuando se considere necesario:

```
audio.playEffect("Explosion.ogg");  
audio.playMusic("LaTrampa.ogg");
```

Esta librería es bastante potente y tiene muchas opciones, entre ellas:

- Control si el archivo de audio es válido.
- Cargar música de Background.

Problemas Encontrados

El principal problema a la hora de implementar el juego fue la casi inexistente documentación que existe acerca de la versión de cocos2d para html5.

Con respecto a las colisiones, uno de los problemas que encontramos es al detectar las colisiones. Al tener objetos con estructura física no podemos utilizar las propiedades que tienen los sprites para detectar las mismas entre ellos. Entonces nos encontramos que teníamos que detectar las colisiones físicamente.

El problema está en que por la forma que estructuráramos el programa se hacía difícil la detección de las mismas, entonces resolvimos hacer un control de la ubicación de la bala en el momento de explotar, y si se encuentra en donde están posicionados los conejos ejecutamos la acción, que consta en colorear de negro los conejos a medida que una bala los toca y los “mata”.

El desconocimiento del framework sumado a el poco tiempo de aprendizaje que teníamos provoco que no aprovecháramos el potencial que tiene Cocos2d.