Chocoboworld

Cocos2d Framework

Software de entretenimiento y videojuegos

Universidad de Oviedo

2016/2017

Adrián Bueno

Introducción

He realizado un juego con perspectiva superior en el que se puede manejar un personaje en 4 **direcciones** diferentes (arriba, abajo, izquierda y derecha), además de otras acciones como **lanzar proyectiles** y **buscar tesoros**. Este juego pretende emular al minijuego del Final Fantasy IX ***hot and cold***.
Enlace al mini juego original: https://www.youtube.com/watch?v=UtD-CENP0Ks

Pantalla de inicio

Pantalla de juego (recortada)

NOTA PARA EMPEZAR A JUGAR: usar FIREFOX.
En un principio se tienen 0 puntos, no puedes lanzar proyectiles para alejar a los cuervos, puedes pulsar X para obtener 20 puntos, pero lo normal sería intentar encontrar tesoros.
He realizado un video dónde muestro varias de las acciones que se pueden realizar en el juego.

https://www.youtube.com/watch?v=PlTtDsk8h_s

Como Jugar

Deben utilizarse las **flechas para desplazarse**, el **control con el ratón también está implementado**, pero como es más difícil manejar el juego con el ratón he decidido desactivarlo comentando ese código, de este modo los botones no estorban en la pantalla.

La **barra espaciadora** sirve para dar un picotazo y **buscar tesoros**. Al pulsarla se recibe un *feedback* (*es un label con una sequence*).

- *Kue…* *No hay nada cerca*
- *Kue?!* *Hay algo no demasiado lejos*
  Kue??!! *Hay algo muy cerca.*
- *K-kueee!!* *Ha encontrado algo.*

Al encontrar el tesoro se recibirá otro tipo de *feedback* y **un efecto de sonido**. También aparecerá un sprite mostrado el cofre y se sumará una cantidad aleatoria de puntos.

Con la tecla **Z** lanzamos un **proyectil** en la dirección en la que miramos, dependiendo de los puntos que tengamos el efecto puede ser variable al alcanzar a un enemigo. **Hay varios tipos** de proyectiles, con los **botones 1 y 2** pueden cambiarse, además estos mejoran al **alcanzar los 200** puntos.

Elementos del juego

El elemento principal es el jugador, representado por las animaciones de un **chocobo**. Este interactúa con los **enemigos**, **proyectiles** y **muros** y busca **tesoros.**

Los **enemigos** son cuervos que usan una animación al desplazarse, estos persiguen al jugador si este está cerca, quitándole puntos y vidas si le tocan.

El jugador puede lanzar **proyectiles**, hay 4 tipos, 2 principales que son mejorados cuando se llega a 200 puntos. Hay proyectiles que avanzan en línea recta y otros que giran alrededor del jugador protegiéndolo.

Los **tesoros** son el elemento principal que suma puntos, al encontrar un tesoro este aparece representado por un sprite que se queda visible de forma permanente.

Funcionalidades

Menús

Solo existe el menú principal para empezar a jugar y un menú de pausa y ayuda que se activa al pulsar la tecla **ESC**. La implementación no tiene mucho misterio más allá de la creación de su sprite, posicionarlo y añadirlo a la capa.

Controles

La implementación de los controles es un poco más compleja, sobretodo porque cuándo la implementé desconocía que había que usar **“.clone()”** para las secuencias y animaciones, por este motivo tuve bastante cuidado en que no se solaparan peticiones de control, pues al solaparse la misma secuencia o animación en ocasiones está se quedaba a la mitad congelada.
La imagen de abajo se corresponde al código que se encarga de llamar al método de búsqueda de tesoros en el jugador.

```
}else if(instancia.teclaPulsada != keyCode && !instancia.enTransaccion){
    instancia.enTransaccion = true;
    var velBase = gameLayer.jugador.velBase;
    /*  Aquí se llama al método que hace la búsqueda de tesoros
        En función del retorno mostramos un label
        OJO: Este label se añade a la capa de controles no a la del jugador,
        el jugador está en medio de la pantalla asique damos esta posición ABSOLUTA
        La posición debe darse aquí siempre, no al instanciar, pues la secuencia text_popup la modifica.
        Solo se podrá entrar si no hay una secuencia text_popup activa en este momento
    */
    if(keyCode == 32 && anims.secuencia_text_fin){ // KUEEE Usando clone() podría quitar esto si hago de etiquetaKue instancias distintas.
        gameLayer.jugador.sprite.stopAllActions();
        gameLayer.jugador.body.setVel(cp.v(0, 0));
        var kue =  gameLayer.jugador.kue();
        instancia.etiquetaKue.setString(kue);
        instancia.etiquetaKue.setPosition(cc.p(cc.winSize.width/2, cc.winSize.height/2));
        instancia.addChild(instancia.etiquetaKue);
        instancia.etiquetaKue.runAction(anims.secuencia_text_popup.clone());
        instancia.etiquetaTesoros.setString("Puntuación: "+gameLayer.jugador.puntos);
```

Los controles de movimiento son todos similares a este

```
if(keyCode == 37){
    gameLayer.jugador.sprite.stopAllActions();
    gameLayer.jugador.body.setVel(cp.v(0, 0));
    gameLayer.jugador.caminarIzquierda();
    gameLayer.jugador.body.applyImpulse(cp.v(-velBase,0), cp.v(0,0));
```

Lista de controles

ESC	Pausa el juego.	ARRIBA	Movimiento
ESC	Muestra el panel de ayuda	ABAJO	Movimiento
Z	Lanza un proyectil si se tienen puntos.	DERECHA	Movimiento
X	Para **DEBUG** suma 20 puntos.	IZQUIERDA	Movimiento
V	Aumenta la velocidad del jugador a cambio de 200 puntos.		
ESPACIO	Busca tesoros en el área circundante.		

La capa de controles también se encarga de la gestión de toda la interfaz, los puntos son actualizados y visualizados en esta capa junto al número de vidas mostradas.

Animaciones y secuencias.

He sacado todas las animaciones (con sus cargas) y secuencias a un fichero a parte solo para esto. A continuación, muestro el código de carga de la secuencia que muestra los mensajes encima de la cabeza del **chocobo**. Realmente las acciones inicio y fin no harían falta una vez sabes que se debe usar **.clone()**. Las he dejado como ejemplo del uso de **cc.CallFunc**, además también está el **cc.RemoveSelf** que es necesario para que el mensaje desaparezca y no se quede en la capa de controles mostrándose siempre.

```
},cargar_secuancia_text_popup(){
    var mov1 = cc.MoveBy.create(0.3, cc.p(0, 20));
    var mov2 = cc.MoveBy.create(0.05, cc.p(3, 0));
    var mov3 = cc.MoveBy.create(0.05, cc.p(-3, 0));
    var rem = cc.RemoveSelf.create();
    var inicio = cc.CallFunc.create(this.inicio_secuencia_text, this);
    var fin = cc.CallFunc.create(this.fin_secuencia_text, this);
    this.secuencia_text_popup = cc.Sequence.create(inicio,mov1,mov2,mov3,mov2,mov3, rem, fin);
```

Esta es la secuencia para representa la pérdida de una vida.

```
},cargar_secuencia_damage(){
    var fadeIn = new cc.FadeIn(0.3);
    var fadeOut = new cc.FadeOut(0.3);
    //var desTint = tintTo.reverse();
    this.secuencia_damage = cc.Sequence.create(fadeOut,fadeIn,
```

A continuación, describo lo que nos encontramos en el fichero fuente *"Animaciones.js".*

Animaciones para caminar	secuencia para animar textos temporalmente
Animaciones para picar	secuencia para realizar efectos visuales (daño)
Animaciones proyectiles	secuencia para animar los tesoros.
Animaciones enemigos	
Animacion de explosion	

Cabe destacar la utilidad de **cc.Spawn()** aunque no he tenido la necesidad de usarlo si he experimentado con ello para realizar acciones paralelas.

Enemigos y movimiento.

Los enemigos persiguen al jugador tras detectarle para ello simplemente he calculado la distancia al jugador y cuándo es la adecuada calculo la velocidad en cada eje para aproximarse al jugador manteniendo la velocidad base del enemigo.

Cálculo de vx y vy

```
if(distancia < 150 && !this.impactado){
    this.persiguiendo = true;
    var velocidadX;
    var velocidadY;
    var velocidadBase = Math.abs(this.velBase);
    var factor = 0;
    if(distanciaX >= 0)
        this.orientacionX = 1;
    else
        this.orientacionX = -1;
    if(distanciaY >= 0)
        this.orientacionY = 1;
    else
        this.orientacionY = -1;
    distanciaX = absx;
    distanciaY = absy;
    if(distanciaX > distanciaY){
        factor = distanciaX/distanciaY;
        var vB = velocidadBase * velocidadBase;
        factor = 1 + (factor * factor);
        velocidadY = vB / factor;
        velocidadY = Math.sqrt(velocidadY);
        velocidadX = velocidadBase - velocidadY;
    }else{
        factor = distanciaY/distanciaX;
        var vB = velocidadBase * velocidadBase;
        factor = 1+ (factor * factor);
        velocidadX = vB / factor;
        velocidadX = Math.sqrt(velocidadX);
        velocidadY = velocidadBase - velocidadX;
    }
    velocidadX = velocidadX * this.orientacionX;
    velocidadY = velocidadY * this.orientacionY;
    this.body.setVel(cp.v(velocidadX,velocidadY));
```

Cuándo los enemigos no detectan al jugador simplemente se mueven en dirección vertical u horizontal según les dé.

Proyectiles y movimiento.

Los proyectiles de desplazamiento lineal son sencillos de hacer, pero los que se mueven de forma circular son un poco más complicados, su código se encuentra en el update() de Proyectiles.js.
Se controlan las colisiones con muros y enemigos, y según el tipo de proyectil este se destruye o no.

Un proyectil solo elimina completamente un enemigo si se ha mejorado el proyectil teniendo más de 200 puntos.

Música y sonido

Se utilizan efectos del sonido al disparar proyectiles o encontrar tesoros, también hay música de fondo. Hay archivos de tipo **.wav** y **.ogg** lo más prudente es usar **FireFox** para ejecutar el juego.

Ampliaciones y Conclusiones

Hay elementos que pueden ser fácilmente mejorados o ampliados.

- Enemigos que lanzan proyectiles.
- Más tipos de proyectiles, por ejemplo, proyectiles que persiguen a enemigos.

Pues hasta hace muy poco le tenía bastante asco a JavaScript y a este entorno, pero justo cuando conseguí manejar a mi gusto las secuencias empecé a cambiar de opinión, estoy bastante encantado con el framework este.
El mayor problema que le veo es que mucha de la documentación que hay esta en C++ o en LUA, y aunque uno tenga un manual o página web de referencia que te diga la API, realmente de ahí a usarla hay un abismo.