Rocket League 2D

Animación 3D (Javascript)

Nicolás Tapia Sanz

3.3. - DVJU

ESNE 2017/2018

Rocket League 2D

Animación 3D (Javascript)

Introducción

A continuación se procede a explicar el diseño del juego Rocket League 2D a nivel de desarrollo, funcionamiento y planteamiento desarrollado en Javascript para la asignatura de Animación 3D.

Objetivo

El objetivo de esta práctica es la implementación de un videojuego de temática libre para navegadores web usando Box2D, Javascript y el canvas de HTML5.

El juego que se ha desarrollado en este caso es un juego similar al conocido Rocket League pero en 2D y para dos jugadores locales. El juego consiste en controlar un vehículo que se puede mover tanto a derecha como a izquierda y saltar. Mediante este vehículo, el jugador debe tratar de marcar en la portería opuesta a la suya mientras que evita que el otro jugador marque en su portería. El objetivo del usuario es marcar 10 goles el primero o, por lo menos, marcar más goles que el contrario en 4 minutos.

Fechas

Comienzo: 29/01/2018
Entrega: 18/02/2018

Gameplay

El juego tiene 2 pantallas: el juego en sí mismo y la pantalla de Game Over.

Al arrancar el juego, el partido comienza. Cada jugador puede controlar un vehículo. El jugador número 1 controla al coche rojo. Este se mueve de derecha a izquierda con las teclas A y D respectivamente. Puede utilizar el turbo con la tecla ESPACIO. Con la tecla W puede saltar. El jugador número 2 contrala su vehículo con las flechas. Su tecla de turbo es la tecla INTRO.

Cada jugador debe de tratar de meter el balón en la portería opuesta al lado en el que aparece al comienzo del juego.

Cada uno posee una barra de Power que se recarga si el jugador se mueve y/o salta. Cuanto más se mueva, antes se cargará. Al pulsar la tecla de Power se podrá mover más rápido y saltará más alto. pero esta ira disminuyendo progresivamente hasta llegar a 0.

En el centro de la pantalla el jugador podrá ver como va el marcador del partido y el tiempo restante para que acabe este.

El fondo está compuesto por dos imágenes: una estática de un estadio de fútbol y otra de un cielo que va cambiando (girando) progresivamente con el cambio del tiempo. Hay tres sonidos principales: uno de fondo del estadio, otro al golpear la pelota y otro cuando se marca gol.

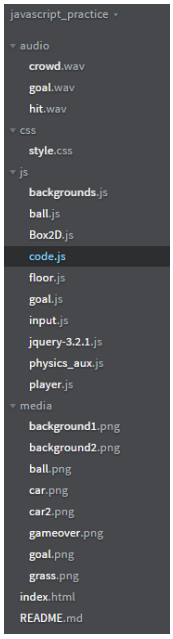
El partido puede acabar de dos maneras: si pasan 4 minutos o un jugador alcanza una puntuación de 10.

Al acabar, aparecerá una pantalla de Game Over indicando el resultado final del partido. Si se pulsa la tecla ESPACIO, se reiniciará y se inciará un nuevo partido.

Descripción Técnica

A continuación se explican los componentes del juego:

Estructura



Esta es la estructura de archivos del proyecto: En la carpeta media se encuentran todas las imágenes que utiliza el juego. En la carpeta Js se encuentran todos los archivos javascript. En la carpeta css se encuentra el archivo css que da estilo a la web. En la carpeta audio se encuentran todos los archivos .wav de sonido.

Code.js

```
var canvas;
var ctx;

var pi_2 = Math.PI * 2;

var fixedDeltaTime = 0.01666666; // 60fps: 1 frame each 16.66666ms
var deltaTime = fixedDeltaTime;

var time = 0,
    FPS  = 0,
    frames   = 0,
    acumDelta = 0;

//Referencia a las imagenes
var player1Img, player2Img, floorImg, background1Img, background2Img, goalImg, ballImg, gameOverImg;

//Referencia a los objetos
var player1, player2, floor, background1, background2, goalL, goalR, ball;

//Sonidos del juego
var sounds={ ··· }

//Valores para el Temporizador
var timer;
var compareDate;

//TIEMPO QUE DURA EL PARTIDO
var matchDurationInMinutes = 4;
var seconds;
var minutes;

//TRUE cuando el juego acaba, FALSE mientra se juega el partido
var gameOver = false;

function Init (){ ··· }

function Start(){ ··· }

function Loop (){ ··· }

function Update (){ ··· }

//Si el jugador muve su vehiculo, este se recarga de potencia
function Powerized(player){ ··· }

//MÉTODO DE DIBUJADO -------
function Draw (){ ··· }

//DIBUJADO DE LOS FONDOS
function DrawBackground (){ ··· }

//PREPARACIÓN DEL CANVAS Y EL MUNDO
function DrawWorld (world){ ··· }

//REINICIO TRAS UN GOL
function RestartGoal(){ ··· }

//CONTROLADOR DEL TEMPORIZADOR
function Clock(toDate){ ··· }
```

Esta es la clase encargada de gestionar todo el flujo de juego. Se encarga de controlar todos los objetos que hay en la escena, encontrar los sprites, los sonidos, el control de eventos, dibujado general, reiniciado, marcador y temporizador.

Background.js

```
function NewBackground (options) {
    return {

        //PROPIEDADES DEL FONDO
        type: options.type,                     //Nombre
        width: options.width,                   //Anchura
        height: options.height,                 //Altura
        position: {x: options.x, y: options.y},  //Posicion
        img: options.img,                       //Image
        angle: 0,                               //Angulo

        Update: function(deltaTime){
            if(this.type == 'background1'){
                //Actualización de la rotación del fondo 1
                this.angle = (this.angle - 0.03) % 360;
            }
        },

        //Este método tan solo se llama para el fondo que gira
        Draw: function(ctx){
            ctx.save();
            //Traslación al centro, rotación y recolocación
            ctx.translate(400, 225);
            ctx.rotate(this.angle * Math.PI /180);
            ctx.translate(-400, -225);
            //Imagen del fondo
            ctx.drawImage(this.img, 0, 0, this.width, this.height, -50,-225, this.width, this.height);
            ctx.restore();
        },

    }
}
```

Este código se encarga de las propiedades del fondo giratorio. Estaba enfocado a que ambos fondos utilizaran este código, pero finalmente solo lo utilizó en uno de ellos, el que cambia con el paso del tiempo.

Ball.js

```
function NewBall(options) {
    return {

        //Propiedades de la PELOTA
        type: options.type,
        width: 0.2,
        height: 0.2,
        position: {x: options.x, y: options.y},
        imgScale: 0.5,
        score: options.score || 1,
        img: options.img,

        isGoal: false, //Si se ha marcado gol con ella
        hitted: false, //Si ha sido golpeada con algo

        //FÍSICAS
        physicsInfo: {
            density: 1,
            fixedRotation: true,
            linearDamping: 1,
            user_data: player1,
            type: b2Body.b2_dynamicBody,
            restitution: 0.0
        },

        body: null,

        //Inicialización de las físicas de la pelota
        Start: function(){
            this.body = CreateBall(world,
                        this.position.x/scale, this.position.y / scale,
                        this.width, this.height, this.physicsInfo);

            this.body.SetUserData(this);
        },

        Update: function(deltaTime){
        },

        //DIBUJADO DE LA PELOTA
        Draw: function(ctx){

            var bodyPosition = this.body.GetPosition();
            var posX = bodyPosition.x * scale;
            var posY = Math.abs((bodyPosition.y * scale) - ctx.canvas.height);
            var bodyRotation = this.body.GetAngle();

            ctx.save();
            ctx.translate(posX, posY);
            ctx.scale(this.imgScale, this.imgScale);
            ctx.rotate(-bodyRotation);

            ctx.drawImage(this.img,
                -this.width * scale*2,
                -this.height * scale*2,
                this.width * scale * 4 , this.height * scale * 4 );
            ctx.restore();
        },

        //REINICIADO DE LA PELOTA
        Restart: function(){
            isGoal = false;
            this.body.GetWorld().DestroyBody(this.body); //Se borra su cuerpo físico para quitar cua
            this.body = CreateBall(world, this.position.x/scale, this.position.y / scale,
                this.width, this.height, this.physicsInfo); //Y se crea un nuevo cuerpo física con l
            this.body.SetUserData(this);
            ctx.restore();
        }

    }
```

Es el encargado de manejar la pelota de la escena. Posee todas sus propiedades, tanto de posición como físicas o booleanas y sus métodos de actualización, dibujado y reiniciado.

Floor.js

Es el código encargado de las propiedades, inicialización y dibujado del suelo de la escena.

```
function NewFloor (options) {
    return {
        //PROPIEDADES DEL SUELO
        type: "floor",
        width: options.width,
        height: options.height,
        position: {x: options.x, y: options.y},
        img: floorImg,

        //FÍSICAS DEL SUELO
        physicsInfo: {
            density: 10,
            fixedRotation: true,
            type: b2Body.b2_staticBody
        },

        body: null,

        //INICIALIZACIÓN DEL CUERPO FÍSICO DEL SUELO
        Start: function(){
            this.body = CreateBox(world, this.position.x/scale, this.position.y / scale,
                this.width, this.height/10, this.physicsInfo);
            this.body.SetUserData(this);
        },

        Update: function(deltaTime){
        },

        //DIBUJADO DEL SUELO
        Draw: function(ctx){
            ctx.drawImage(this.img, 0, 0,
                this.width * scale * 2, this.height * scale * 4,
                0, 414, this.width * scale* 2, this.height * scale * 2 );
            ctx.restore();
        }
    }
}
```

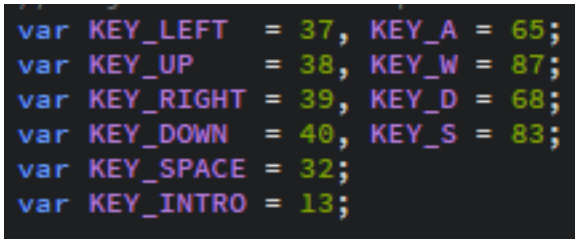
Goal.js

Se encarga de las propiedades, físicas, inicialización y dibujado de las porterías que se encuentran en la escena.

```
function NewGoal (options) {
    return {
        //PROPIEDADES DE LA PORTERIA
        type: options.type,
        width: options.width,
        height: options.height,
        position: {x: options.x, y: options.y},
        img: options.img,

        //FÍSICAS DE LA PORTERÍA
        physicsInfo: {
            density: 10,
            fixedRotation: true,
            type: b2Body.b2_staticBody
        },

        body: null,

        //INICIALIZACIÓN DEL CUERPO FÍSICO DE LA PORTERIA
        Start: function(){
            this.body = CreateBox(world, this.position.x /scale+1.25, this.position.y / scale-0.75,
            0.75, 1, this.physicsInfo);
            this.body.SetUserData(this);
        },

        Update: function(deltaTime){
        },

        //DIBUJADO DE LA PORTERÍA
        Draw: function(ctx){
            ctx.save();

            //Si la portería es la dererecha, se invierte el sprite.
            if(this.type == 'goalR'){
                ctx.scale(-1, 1);
            }

            ctx.drawImage(this.img,
                this.position.x,
                this.position.y,
                this.width, this.height);
            ctx.restore();

        }
    }
}
```

Input.js

Es el código encargado de controlar todos los eventos de teclado y de ratón que se producen. En la imagen encontramos los eventos de teclado que me interesan recoger.



```
var KEY_LEFT  = 37, KEY_A = 65;
var KEY_UP    = 38, KEY_W = 87;
var KEY_RIGHT = 39, KEY_D = 68;
var KEY_DOWN  = 40, KEY_S = 83;
var KEY_SPACE = 32;
var KEY_INTRO = 13;
```

Physics_aux.js

Este código es un auxiliar para poder trabajar con Box2D con mayor facilidad. En la imagen se muestra una parte de este código. En él se ha generado unos limites a la escena.

Así mismo se incluye el conjunto de colisiones que interesa detectar como son la pelota con cada porteria o la pelota con algun objeto para que suene.

```
function CreateWorld (ctx, gravity)
{
    var doSleep = false;
    world = new b2World(gravity, doSleep);

    // DebugDraw is used to create the drawing with physics
    var debugDraw = new b2DebugDraw();
    debugDraw.SetSprite(ctx);
    debugDraw.SetDrawScale(scale);
    debugDraw.SetFillAlpha(0.5);
    debugDraw.SetLineThickness(1.0);
    debugDraw.SetFlags(b2DebugDraw.e_shapeBit | b2DebugDraw.e_jointBit);

    world.SetDebugDraw(debugDraw);

    // BORDES Y LIMITES DEL ESPACIO
    // left wall
    CreateBox(world, 0, 1, .1, 8, {type : b2Body.b2_staticBody});
    // down wall
    CreateBox(world, 4, 0, 4, .1, {type : b2Body.b2_staticBody});
    // right wall
    CreateBox(world, 8, 1, .1, 8, {type : b2Body.b2_staticBody});


    return world;
}

//CONTROL DE COLISIONES
function OnContactDetected (contact){

    var a = contact.GetFixtureA().GetBody().GetUserData();
    var b = contact.GetFixtureB().GetBody().GetUserData();

    if(a != null && b != null &&
        typeof(a.type) !== 'undefined' &&
        typeof(b.type) !== 'undefined'){

        //EN CASO DE QUE SE HAYA MARCADO GOL EN LA PORTERÍA DERECHA...
        if((a.type == "goalR" && b.type == "ball") ||
            (b.type == "goalR" && a.type == "ball")){
            var ballTemp = (a.type == "ball") ? a : b;
            player2.score += ballTemp.score;
            ballTemp.isGoal = true;
        }

        //EN CASO DE QUE SE HAYA MARCADO GOL EN LA PORTERÍA IZQUIERDA...
        if((a.type == "goalL" && b.type == "ball") ||
            (b.type == "goalL" && a.type == "ball")){
            var ballTemp = (a.type == "ball") ? a : b;
            player1.score += ballTemp.score;
            ballTemp.isGoal = true;
        }

        //EN CASO DE QUE LA PELOTA HAYA GOLPEADO CON ALGO
        if(a.type == "ball" || b.type == "ball"){
            var ballTemp = (a.type == "ball") ? a : b;
            ballTemp.hitted = true;
        }

    }
}
```

Player.js

Es el encargado de controlar las propiedades del jugador, potencia, puntuación, booleanos, físicas, dibujado, velocidades, reiniciacido, etc. Dado que es el elemento controlable de la escena, es el más importante (sin tener en cuenta el flujo del juego), y por ello tiene más métodos extra que los demás, asi como más propiedades.

```
function NewPlayer(options) {
    return {
        //PROPIEDADES DEL JUGADOR
        type: options.type,
        position:{x: options.x, y: options.y},
        width: 0.37,
        height: 0.27,
        img: options.img,

        //Potencia y Puntuación
        power: 0,
        score: 0,

        //movement Attributes
        maxHorizontalVel: 4,
        maxVerticalVel: 6,
        jumpForce: 4,

        //Direccion
        moveLeft: false,
        moveRight: false,
        moveUp: false,

        //Posibilidad de movimientos
        canJump: false,
        isPowered: false,
        isGoingLeft: false,

        bodyInit: null,

        //ANIMACIÓN DE SPRITE Y DIBUJADO
        animation:{ ··· },

        //FÍSICAS DEL JUGADOR
        physicsInfo: { ··· },

        body: null,

        //INICIALIZACIÓN DEL CUERPO FÍSICO DEL PLAYER
        Start: function(){ ··· },

        //UPDATE DEL PERSONAJE
        Update: function(deltaTime){ ··· },

        //Dibujado de los sprites
        Draw: function(ctx){ ··· },

        //VELOCIDADES DEL PLAYER
        ApplyVelocity: function(vel){ ··· },

        //SALTO DEL PLAYER
        Jump: function(){ ··· },

        //REINICIO DEL PLAYER TRAS MARCARSE UN GOL
        Restart: function(){ ··· },

        //MOVIMIENTO EN CASO DE USAR EL POWER
        Power: function(){ ··· }
    }
}
```

Enlace a Repositorio

https://github.com/Nikoooo95/javascript_practice.git