MEMORIA

ANIMACIÓN 3D

PRÁCTICA FINAL

Pablo Trascasa Alcalde
VDJ 3.3

Pablo Trascasa Alcalde
VDJ 3.3

DESCRIPCIÓN DE LA PRÁCTICA

El juego desarrollado consiste en una versión/imitación del videojuego Timberman, disponible para varias plataformas.

El juego es un "Endless Game" donde la mecánica principal es que el jugador debe ir cortando el árbol esquivando las ramas, para ello puede cambiar de lado moviéndose a izquierda o derecha, el juego finaliza cuando el jugador choca con una rama.

En el siguiente enlace se ve un gameplay de ejemplo del juego original.
https://www.youtube.com/watch?v=rX38KHu72_0

Para esta práctica he podido acceder a los assets originales de la primera versión del juego, los cuales he utilizado por ahorrar tiempo en hacer los míos propios o encontrar algunos similares.

Esta práctica contiene los siguientes apartados requeridos.

- Apartados de funcionalidad obligatorios
- Apartados de funcionalidad optativos
    - APl de Facebook
    - Animación de un spritesheet
    - Efectos de sonido
    - Adaptación a Cordova

Esta práctica **no** contiene los siguientes apartados requeridos.

- Apartados de funcionalidad optativos
    - Utilización de box2d – No implementado por falta de tiempo
    - Sistemas de partículas – Sólo se implementaba si sobraba tiempo
    - Utilización de ficheros json
    - Menú con estilos CSS* - No implementado por falta de tiempo

*Pero adjunto un **zip** con mi web personal para demostrar mi desempeño con CSS

GAMEPLAY DE LA PRÁCTICA

Al iniciar la página nos aparece un canvas de 800x400 con un fondo y un botón de play.
Al pulsar el botón se inicia el juego, se dibuja al jugador siempre al lado izquierdo al principio y se construye y dibuja el árbol.
Los controles son A o $\Leftarrow$, para mover y cortar por el lado izquierdo o, D o $\Rightarrow$, para mover y cortar por el lado derecho.
Cada corte satisfactorio incrementará la puntuación en un punto.
Cuando el jugador colisione con una rama se acaba el juego, aparecerá una pantalla de "game-over" con la puntuación conseguida y a los dos segundos se recargará la página.

Pablo Trascasa Alcalde
VDJ 3.3

DESCRIPCIÓN DEL CÓDIGO

index.html

Es la página principal donde se va a cargar el juego.
Al cargar el body lanza la función **Init()** de game.js, para no cargar la partida nada más empezar, he creado un botón que lanza una función que, desactiva el propio botón, cambia el valor de una variable de control para que se ejecute el código en el **Loop()** de game.js

```
<body onload="Init();">
    <div id="container">

        <button id="btn_play" style="background: url(img/btn-play.png)" onclick="startFunction()"></button>
        <canvas id="my_canvas" width="450" height="800" allow="autoplay"></canvas>
    </div>

    <!-- Al pulsar el botón lo eliminamos para que no moleste -->
    <!-- Cambia la variable controladora selected a true para que inicie el juego en game -->
    <script>function startFunction() {
            selected = true;
            var elem = document.getElementById('btn_play');
            elem.parentNode.removeChild(elem);
          }
        </script>
</body>
```

game.js

Es el código principal donde se va a cargar el grueso del juego y funcionar una parte de la lógica. Hay dos variables que controlan si ha empezado la partida o ha acabado (selected, gameOver).
La función **Update()** comprueba si las teclas que han sido pulsadas corresponden con (A,D, ⇐, ⇒), también actualiza el **Update()** de player.js y la función **CheckCollision()** para comprobar si el jugador ha chocado con alguna rama.

```
function Update() {
  if (input.isKeyDown(KEY_D) || input.isKeyDown(KEY_RIGHT)){ //Si se ha pulsado D o tecla derecha
    player.moveRight = true;
  }
  if (input.isKeyDown(KEY_A) || input.isKeyDown(KEY_LEFT)){ //Si se ha pulsado A o tecla izquierda
    player.moveLeft = true;
  }

 player.Update(deltaTime);                                  //Lógica del player
 CheckCollision();                                          //Comprueba colisiones
}
```

La función **Draw()** se encarga del dibujado de los componentes de la escena, se actualiza en la función **Loop()**, dibuja el fondo, el jugador y el árbol, también dibuja los FPS y la puntuación.

```
function Draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);       //Limpia el canvas
  background.Draw(ctx);                                   //Dibuja el fondo
  ctx.globalCompositeOperation = "source-over";
  player.Draw(ctx);                                       //Dibuja el jugador
  tree.Draw(ctx);                                         //Dibuja el árbol
```

La función **CheckCollision()** controla si el jugador ha chocado con una rama.
Como los objetos no tienen cuerpo, se calcula comprobando el lado en el que se encuentra el jugador con la rama que hay en la posición [0] del array del árbol.

```
function CheckCollision(){
  if(player.lado==1 && tree.tree[0] == 4){          //Si el jugador está en el lado derecho y el tronco en la posición [0] es la rama hacia la derecha
    this.gameOver=true;                             //Se acaba el juego
    this.deathAudio.play();                         //Inicia el sonido de muerte del jugador
    //console.log("CheckCollision");
  }else if(player.lado==0 && tree.tree[0]==3){      //Si el jugador está en el lado izquierdo y el tronco en la posición [0] es la rama hacia la izquierda
    this.gameOver=true;                             //Se acaba el juego
    this.deathAudio.play();                         //Inicia el sonido de muerte del jugador
    //console.log("CheckCollision");
  }
}
```

background.js

Este código se encarga del dibujado del fondo en el canvas, nada más.

player.js

Este código se encarga de la lógica del jugador, junto a su dibujado y las animaciones.
En el **Update()** comprobaremos si el jugador ha presionado una tecla y se ha movido hacia un lado u otro o no se ha movido, para así cambiar la posición del jugador, realizar la animación de cortar e incrementar la puntuación.

```
if (this.moveRight)                                 //Si el jugador ha pulsado la tecla D o la Derecha
{
  this.chop=true;                                   //Ponemos chop a true, esto activará la animación de cortar en el Draw
  tree.RemoveTrunk();                               //Llamamos a la función eliminar de tree
  console.log('Derecha');
  this.lado=1;                                      //Ajustamos el Lado a 1 para controlar las colisiones en game
  this.desX = 700;                                  //Cambiamos la posición del jugador
  this.moveRight = false;                           //Restauramos el estado del controlador de mover a la derecha
  this.score++;                                     //Incrementamos la puntuación
}
```

En la función **Draw()** dibujaremos la acción determinada por una variable de control en el **Update()**. En caso de que el jugador no haya presionado ninguna tecla se dibujará la animación de “idle”, si el jugador ha presionado alguna tecla, se dibujará la animación de cortar y al acabar volverá a la animación de “idle”.

```
Draw: function (ctx)
{
  if(this.chop == true){                            //si el controlador chop está en true
    ctx.save();
    ctx.scale(this.imgScale, this.imgScale);
    ctx.drawImage(playerImg, this.srcX, this.srcY, this.spriteWidth, this.spriteHeight, this.desX, this.desY, this.spriteWidth, this.spriteHeight); //Dibujamos al jugador
    this.srcX += 527;                               //mueve 527px en el spritesheet para cambiar de sprite
      if(this.srcX >= 1054){                        //si llega a este punto del spritesheet,
          this.srcX = 0;                            //vuelve al punto 0 para reiniciar la animación
          this.chopAudio.play();                    //inicia el sonido de cortar tronco
          this.chop = false;                        //devuelve el estado del jugador a false para que vuelva a idle
      }
    ctx.restore();
```

tree.js

Este código se encarga de todo lo relacionado con el árbol del juego. Consiste en un array de 7 elementos, donde cada elemento corresponde a una parte del tronco del árbol.
Se crea el árbol desde el principio con la función **ConstructTree()**, que construye el árbol desde cero siguiendo unas pequeñas reglas como:
Los dos primeros troncos no tienen ramas para dar margen de maniobra al jugador, los siguientes troncos se generan de manera aleatoria excepto si el tronco anterior es una rama, lo cual automáticamente crea un tronco sin rama para dar margen de maniobra al jugador, queda mejor visualmente y se juega mucho mejor.

```
ConstructTree: function(){
    var trunks = [this.varTrunk1, this.varTrunk2];                //Array de variables para asignar aleatoriamente uno de cada tipo
    var branchs = [this.varBranchL, this.varBranchR];


    this.tree[0] = this.varTrunk1;                                //Al principio, los dos primeros troncos no tienen rama para dar margen de maniobra al jugador
    this.tree[1] = this.varTrunk2;

    for(var i=2; i<=6; i++) {                                       //Los siguientes 5 troncos se generan al azar pero
        if(this.tree.length-1 == this.varBranchL  || this.tree[5] == this.varBranchR){ //Si el tronco anterior es una rama directamente pone un tronco liso
            this.tree[i] = trunks[Math.floor(Math.random() * 2)];       //El tipo de tronco 1 o 2, asigna aleatoriamente al 50%
        }else{
            if(Math.random() * 4 <= 1){                             //Hay una probabilidad entre 4 de que salga un tronco liso
                this.tree[i] = trunks[Math.floor(Math.random() * 2)];   //El tipo de tronco 1 o 2, asigna aleatoriamente al 50%
            }else{                                                  //Hay 3 probabilidades entre 4 de que salga una rama
                this.tree[i] = branchs[Math.floor(Math.random() * 2)];  //Si es una rama hacia la derecha o hacia la izquierda se asigna aleatoriamente al 50%
            }
        }
    }
},
```

La función **RemoveTrunk()** es llamada desde player.js al realizar un movimiento, esto lo que hace es reasignar el array poniendo en cada posición del array el contenido de la siguiente posición, excepto en la última posición del array que esto lo controla la función **AddTrunk()**.

```
RemoveTrunk: function(){
    for(var i=0; i<=5; i++){                                    //Recorremos el array de la posición 0 a la 5
        this.tree[i] = this.tree[i+1];                          //Ponemos el contenido de la posición siguiente del array en la anterior
    }
    this.AddTrunk();                                            //Llamamos a la función Addtrunk() para añadir un tronco a la posición 6 del array
   Draw();                                                      //Llamamos a la función draw
}
```

La función **AddTrunk()** funciona de la misma manera que **ConstrucTree()**, pero solo se aplica a la última posición del array, comprobando la penúltima posición del array para dibujar ramas o troncos sin rama.

```
AddTrunk: function(){
    var trunks = [this.varTrunk1, this.varTrunk2];                  //Array de variables para asignar aleatoriamente uno de cada tipo
    var branchs = [this.varBranchL, this.varBranchR];

    if(this.tree[5] == this.varBranchL || this.tree[5] == this.varBranchR){ //Si el tronco anterior es una rama directamente pone un tronco liso
        this.tree[6] = trunks[Math.floor(Math.random() * 2)];       //El tipo de tronco 1 o 2, asigna aleatoriamente al 50%
    }else{
        if(Math.random() * 4 <= 1){                                 //Hay una probabilidad entre 4 de que salga un tronco liso
            this.tree[6] = trunks[Math.floor(Math.random() * 2)];   //El tipo de tronco 1 o 2, asigna aleatoriamente al 50%
        }else{                                                      //Hay 3 probabilidades entre 4 de que salga una rama
            this.tree[6] = branchs[Math.floor(Math.random() * 2)];  //Si es una rama hacia la derecha o hacia la izquierda se asigna aleatoriamente al 50%
        }
    }
},
```

La función **Draw()** se encarga de dibujar los troncos en el canvas conforme a las normas dadas por **ConstrucTree()**, **RemoveTrunk() y AddTrunk()**, usamos una variable temporal donde vamos a ir guardando las variables de las imágenes de los troncos.

```
Draw: function (ctx) {
    ctx.save();
    ctx.scale(this.imgScale, this.imgScale);
    ctx.drawImage(this.stumpImg, 400, 1650);                    //Dibujamos la base del arbol
    for(var i=0; i<=6;i++){                                     //Recorremos el array del árbol para ir dibujando 1 a 1 los troncos
        if(this.tree[i]==1){                                    //Si la posición i del array = 1 significa que hay un tronco1
            this.varTemp = this.trunk1;                         //Guardamos en la variable varTemp el contenido de trunk1 que es la imagen
            ctx.drawImage(this.varTemp, this.treeX, this.treeY[i]); //Dibujamos la imagen correspondiente a la variable en la posiciónY correspondiente del array
        }else if(this.tree[i]==2){                              //Si la posición i del array = 2 significa que hay un tronco2
            this.varTemp = this.trunk2;                         //Guardamos en la variable varTemp el contenido de trunk1 que es la imagen
            ctx.drawImage(this.varTemp, this.treeX, this.treeY[i]); //Dibujamos la imagen correspondiente a la variable en la posiciónY correspondiente del array
        }else if(this.tree[i]==3){                              //Si la posición i del array = 3 significa que hay una rama izquierda
            this.varTemp = this.branchL;                        //Guardamos en la variable varTemp el contenido de trunk1 que es la imagen
            ctx.drawImage(this.varTemp, this.treeX, this.treeY[i]); //Dibujamos la imagen correspondiente a la variable en la posiciónY correspondiente del array
        }else if(this.tree[i]==4){                              //Si la posición i del array = 4 significa que hay una rama derecha
            this.varTemp = this.branchR;                        //Guardamos en la variable varTemp el contenido de trunk1 que es la imagen
            ctx.drawImage(this.varTemp, this.treeX, this.treeY[i]); //Dibujamos la imagen correspondiente a la variable en la posiciónY correspondiente del array
        }
    }
    ctx.restore();
},
```

Pablo Trascasa Alcalde
VDJ 3.3

CORDOVA

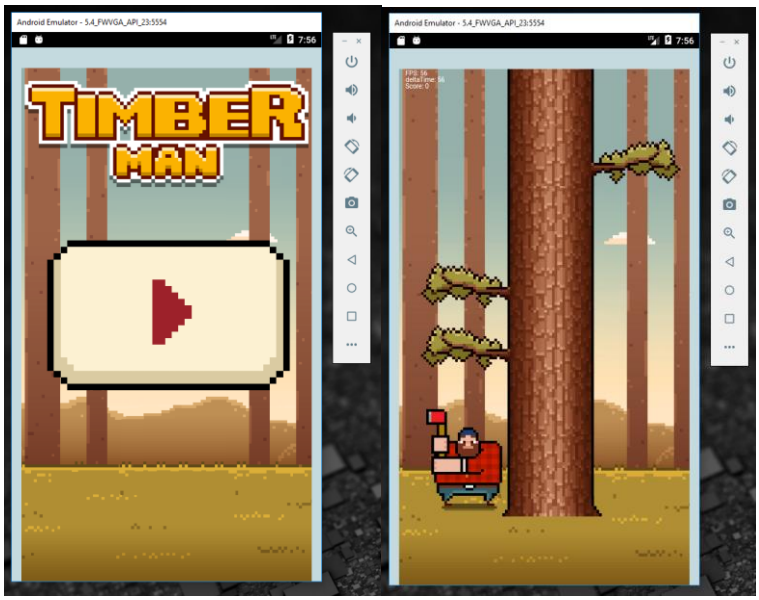
Para instalar Cordova y exportar para android necesitaremos Node.js y el SDK de Java
También he usado la aplicación de Android Studio.

Abrimos un Terminal y escribimos **npm install –g cordova**, esto nos instalará Cordova en nuestro equipo.
Cuando esté instalado Cordova, crearemos un proyecto escribiendo en el terminal **cordova create appcordova**. Le podemos poner cualquier nombre, yo la he llamado appcordova.

Nos movemos por línea de comandos a la carpeta del proyecto y escribimos
cordova platform add android, también podremos ver las plataformas soportadas escribiendo **cordova platforms**.

Ahora abrimos Android Studio y le damos a Import Project (Eclipse ADT, Gradle, etc), vamos a la carpeta del proyecto y seleccionamos la carpeta android.
Una vez abierto podemos ejecutar la app y nos aparecerá la plantilla por defecto en el terminal.
Para que cargue nuestro proyecto en vez de la plantilla por defecto, arrastramos el contenido de la carpeta principal de nuestro proyecto a la carpeta **www** de Cordova.
La ruta de la carpeta www seria **android -> assets -> www**.
Yo he tenido un problema con esto y es que la app no me cargaba bien en el emulador de un Nexus 5X con android 7.0. Con android 6.0 me ha funcionado sin problemas.

Los controles táctiles no funcionan porque no están configurado que funcione con los MouseEvents en el propio código.



Pablo Trascasa Alcalde
VDJ 3.3

API DE FACEBOOK

Para integrar la API de Facebook se han seguido las indicaciones dadas por la documentación en el campus.
Pero también se han tenido que crear una cuenta de desarrollador, con esta cuenta creamos una “app” y la asignamos un “Inicio de sesión con Facebook y plataforma web”.

Una vez creado usamos la guía de inicio rápido, en el primer apartado tenemos que meter la url del sitio web donde está alojado nuestro juego, es importante que sea HTTPS, puesto que en la configuración obliga a ello y no deja cambiarlo.

En el apartado 2, en el primer fragmento de código que nos da, tendremos que cambiar dos parámetros de la función **FB.init()** en nuestro archivo facebook.js.
AppId hace referencia al ID de nuestra aplicación.
Version hace referencia a la versión del SDK.