Creación de un videojuego 2D

Grado de Multimedia

Comunicación visual y creatividad

Autor: José Raúl Jiménez Lama
Consultor: Llogari Casas Torres
11/01/2016

Índice

José Raúl Jiménez Lama
TFG
11/01/2016

Creación de un videojuego 2D

1. Introducción

El presente trabajo se presenta la realización de un videojuego en 2D.

Mi motivación por escoger este tema es el interés en las máquinas recreativas que hicieron furor en la década de los 80. Como reto personal, en este proyecto deseo realizar un videojuego como con los que jugaba en mi infancia, volviendo a los juegos clásicos en 2D.

2. Descripción

A lo largo de este trabajo se pretende crear un videojuego de plataformas en 2D con diferentes niveles y dificultades.

Durante el desarrollo del videojuego se aplicarán los conocimientos adquiridos durante todo el transcurso del Grado de Multimedia, como conocimientos de la animación, sonidos, diseño gráfico, Físicas, usabilidad.

En el desarrollo del videojuego se realizará mediante el motor, software de desarrollo de videojuegos, que nos permitirá realizar este proyecto.

Durante el transcurso del videojuego se presentarán diferentes retos a los usuarios.

3. Objetivo

A continuación describo los objetivos que consigo creando el video juego.

3.1. Principales

Creación de un videojuego totalmente funcional

3.2 Secundarios

Creación de niveles

Creación del personaje principal

Desarrollo de scripts mediante C#

Aplicación de Prefabs

Animación de los personajes

Aplicación de la interactividad

Aplicación de la física en videojuegos

Aplicación de sonidos

Creación del menú inicial

4. Escenario

El videojuego creado en este proyecto de TFG hace referencia a los videojuegos arcade que hicieron furor en las décadas de los 80 y 90.

Maquina arcade de Nintendo.
https://es.wikipedia.org/wiki/Arcade

En la actualidad estos videojuegos se han vuelto a poner de moda, ahora se les conoce como videojuegos retro y se les define como los videojuegos de diversión inmediata. Tal y como comentó el fundador de Atari, *Nolan Bushnell*, “fáciles de entender y difíciles de dominar”, esta fórmula convierte a los clásicos arcade en videojuegos muy adictivos.

5. Contenidos

El videojuego que se presenta es un videojuego 2D de plataforma de acción en el que el personaje principal tendrá derrotar todos los enemigos en cada nivel. Los extremos de cada nivel nos permitirán empezar un nuevo nivel y el extremo opuesto avanzar al siguiente nivel. Cuanto más niveles consigamos pasar se irá aumentará el número de enemigos y su dificultad.

· El videojuego propuesto será un videojuego con 2 niveles de dificultad.
· En los niveles se introducirán elementos de física, como colisiones.
· Se añadirá sonidos.
· El personaje principal se podrá desplazar por el nivel y podrá saltar sobre las diferentes plataformas para superar los retos.
· El personaje podrá disparar para eliminar los diferentes enemigos.
· Se creará un punto de recuperación “checkpoint” en cada uno de los niveles, por si el usuario queda eliminado poder re empezar el nivel desde ese punto.
· Se añadirá contador de puntos de los enemigos que se vayan eliminando.
· Se creará un menú principal.

6. Metodología

Fase 1. Selección

El primer paso a realizar es los análisis de los diferentes recursos que se ofrecen en el mercado para poder realizar un video juego en 2D. Tras el análisis se realiza la selección de la herramienta más adecuada para desarrollar el videojuego.

Fase 2. Formación

En una segunda fase se localizan los manuales de uso de la herramienta escogida para poder llevar a cabo el videojuego.

Fase 3. Definición del videojuego

En esta tercera fase describimos el argumento del videojuego, creando diferentes historias, pantallas, personajes, objetivos, interactividad, etc.

Fase 4. Desarrollo del videojuego

Creación del videojuego definido mediante la herramienta escogida.

Fase 5. Test con usuarios

Comprobación del videojuego mediante el uso con diferentes usuarios, detectando los errores y sugerencias del usuario para aplicarlos en el diseño final.

Fase 6. Publicación

Subir el videojuego a una plataforma web para hacerlo disponible para el usuario final.

7. Arquitectura de la aplicación

La aplicación será un videojuego que se podrá jugar sobre plataforma PC, tanto en plataforma Windows como en Mac. Solo hace falta disponer de un ordenador básico para poder ejecutar la aplicación que realizará las funciones de Cliente/Servidor.

8. Plataforma de desarrollo

En el mercado se ofrecen diferentes engines (motores para poder crear videojugos) con el que se podría realizar un videojuego en 2D y 3D.

Los analizados en la medida de lo posible han sido:

- 001 Engine -> http://www.engine001.com
- 3D Game Marker -> http://www.thegamecreators.com
- Aleph One -> https://alephone.lhowon.org
- Axiom -> http://axiom3d.net
- G4 Engine -> http://www.terathon.com
- Euphoria -> http://www.naturalmotion.com
- Unity 3D -> http://unity3d.com

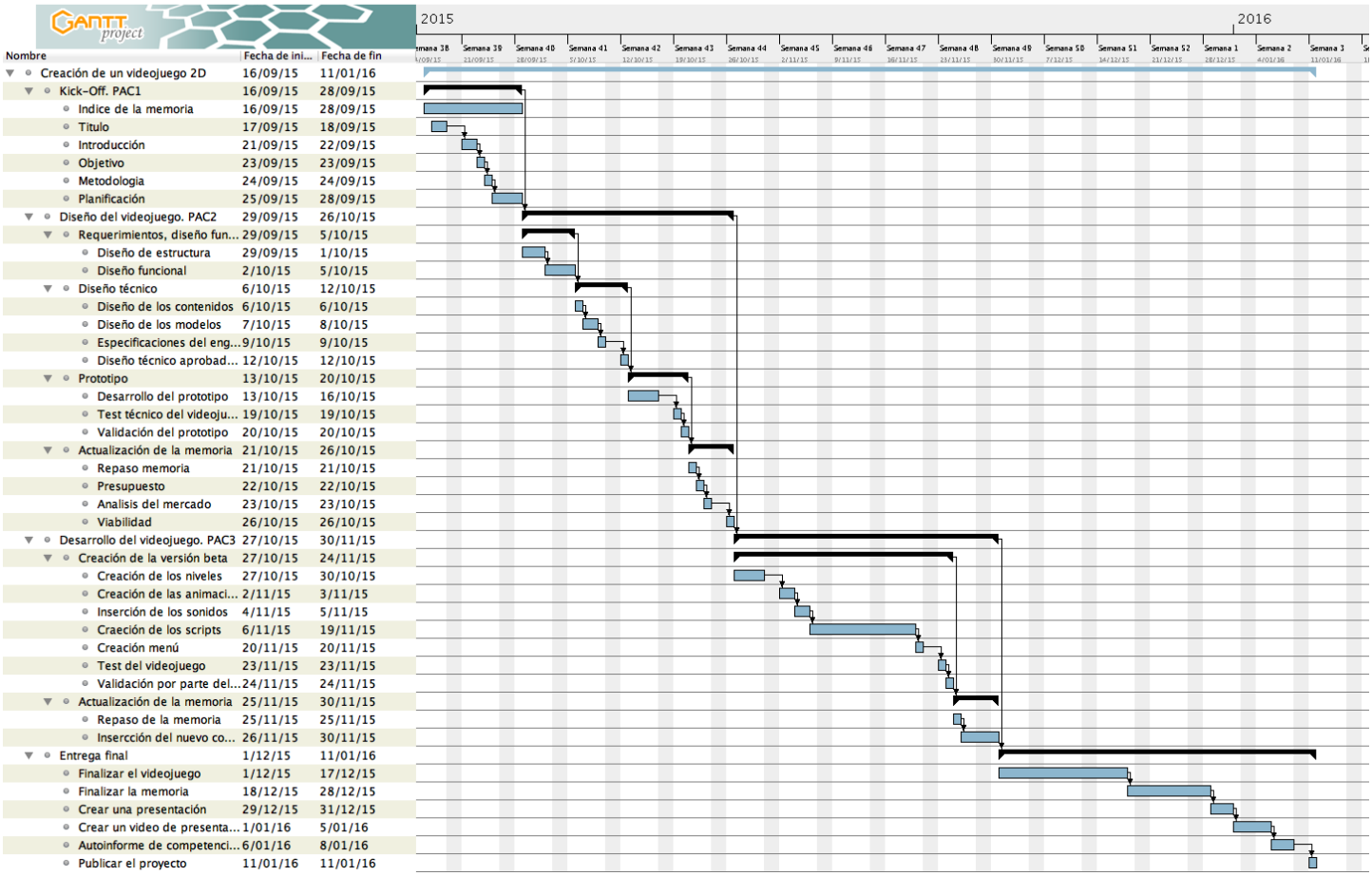
El engine que he decidido utilizar para realizar el videojuego ha sido Unity3D, este me permite crear videojuegos en un entorno gráfico, crear una serie de rutinas de programación que me permite el diseño y la creación del videojuego.

http://unity3d.com/es/unity

Unity3D provee de un motor de renderizado para gráficos 2D y 3D, motor físico para detectar colisiones, programación mediante una plataforma Open Source .Net (Mono Developer), animación, audio, inteligencia artificial, streaming, generador de partículas, luces, edición de menús, construcción de escenarios y un escenario grafico que nos facilita la edición de todo el contenido del videojuego.

He decidido utilizar este engine para crear el videojuego porque nos permite poder exportar el videojuego a las diferentes plataformas actuales en el mercado como Windows, Mac, Linux, Web, IOS, Android y el resto de videoconsolas. También porque la versión personal nos permite realizar el videojuego completo, aunque con algunas limitaciones.

9. Planificación



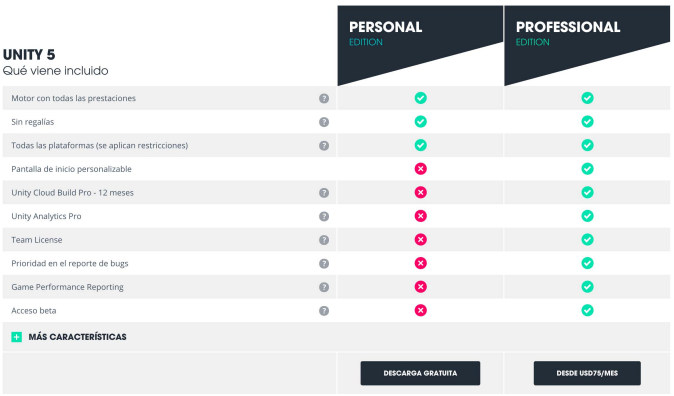
Nombre	Fecha de ini...	Fecha de fin
Creación de un videojuego 2D	16/09/15	11/01/16
Kick-Off. PAC1	16/09/15	28/09/15
Indice de la memoria	16/09/15	28/09/15
Titulo	17/09/15	18/09/15
Introducción	21/09/15	22/09/15
Objetivo	23/09/15	23/09/15
Metodologia	24/09/15	24/09/15
Planificación	25/09/15	28/09/15
Diseño del videojuego. PAC2	29/09/15	26/10/15
Requerimientos, diseño fun...	29/09/15	5/10/15
Diseño de estructura	29/09/15	1/10/15
Diseño funcional	2/10/15	5/10/15
Diseño técnico	6/10/15	12/10/15
Diseño de los contenidos	6/10/15	6/10/15
Diseño de los modelos	7/10/15	8/10/15
Especificaciones del eng...	9/10/15	9/10/15
Diseño técnico aprobad...	12/10/15	12/10/15
Prototipo	13/10/15	20/10/15
Desarrollo del prototipo	13/10/15	16/10/15
Test técnico del videoju...	19/10/15	19/10/15
Validación del prototipo	20/10/15	20/10/15
Actualización de la memoria	21/10/15	26/10/15
Repaso memoria	21/10/15	21/10/15
Presupuesto	22/10/15	22/10/15
Analisis del mercado	23/10/15	23/10/15
Viabilidad	26/10/15	26/10/15
Desarrollo del videojuego. PAC3	27/10/15	30/11/15
Creación de la versión beta	27/10/15	24/11/15
Creación de los niveles	27/10/15	30/10/15
Creación de las animaci...	2/11/15	3/11/15
Inserción de los sonidos	4/11/15	5/11/15
Craeción de los scripts	6/11/15	19/11/15
Creación menú	20/11/15	20/11/15
Test del videojuego	23/11/15	23/11/15
Validación por parte del...	24/11/15	24/11/15
Actualización de la memoria	25/11/15	30/11/15
Repaso de la memoria	25/11/15	25/11/15
Insercción del nuevo co...	26/11/15	30/11/15
Entrega final	1/12/15	11/01/16
Finalizar el videojuego	1/12/15	17/12/15
Finalizar la memoria	18/12/15	28/12/15
Crear una presentación	29/12/15	31/12/15
Crear un video de presenta...	1/01/16	5/01/16
Autoinforme de competenci...	6/01/16	8/01/16
Publicar el proyecto	11/01/16	11/01/16

Diagrama de Gantt – Planificación del proyecto.
NOTA: adjunto fichero PNG (PAC1_Jimenez_JoseRaul_Gantt) del diagrama para que se pueda visualizar mejor.

10. Desarrollo

1- Descargar Unity3D e instalación.

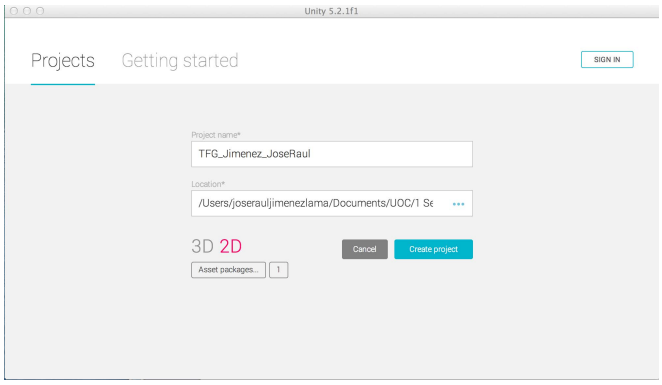
Descargo el engine de Unity en la propia web del fabricante, http://unity3d.com/es/get-unity, descargarnos la versión Personal que es gratuita.



Para realizar la instalación, simplemente seguimos el asistente que nos aparece en la pantalla.

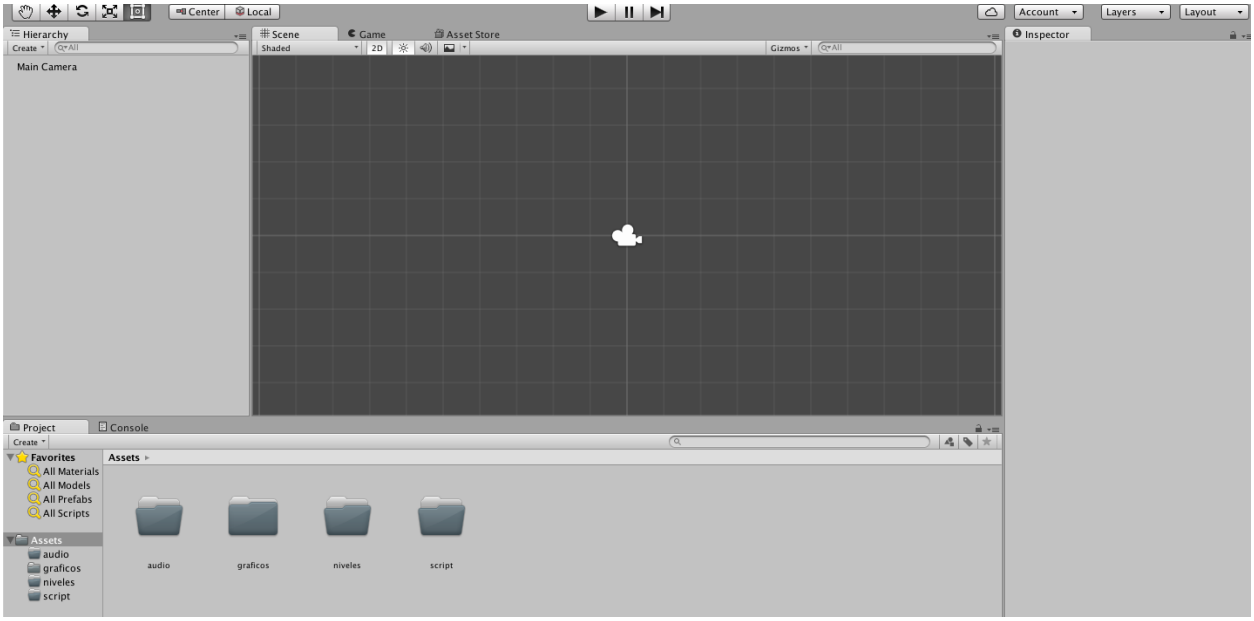
2- Creación del proyecto del videojuego

Al ejecutar Unity3D, deberemos especificar donde queremos guardar el nuevo proyecto y escoger que vamos a crear un videojuego en 2D.



3- Orientación en Unity3D

Ejecutamos Unity3D y nos encontramos con la siguiente interface:



En la interfaz de Unity podemos diferenciar cuatro apartados, en la izquierda aparecerá todos los objetos o gameobjects que forman parte del videojuego. En la parte derecha nos encontramos con el Inspector, que nos facilitará la información de los objetos que pertenecen al proyecto, desde aquí podremos modificar los parámetros de los diferentes objetos. En la parte inferior nos encontramos una estructura de carpetas donde podremos ordenar los objetos del proyecto, que podrán formar parte del proyecto o no. Aquí añadiremos los gráficos, los sonidos, los prefabs, scritps, etc. Finalmente en la ventana central nos encontramos con el editor del videojuego visualmente.

4- Importamos material al proyecto

Para importar objetos al proyecto podemos optar por descargar objetos desde la propia store de Unity3D (Asset Store - https://www.assetstore.unity3d.com/en/) o descargar los objetos que desea uno y añadirlos en el proyecto.

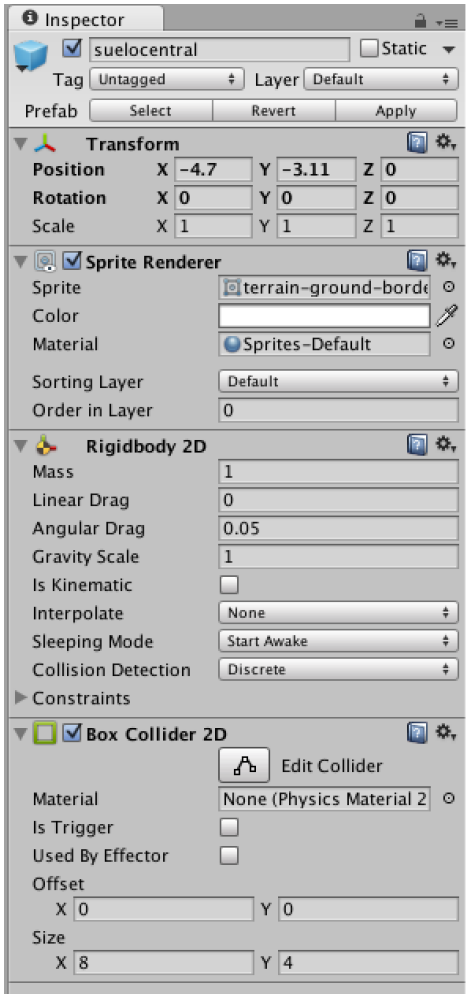
5- Creación del primer nivel

Importamos desde el Asset Store material 2D para crear el suelo del primer nivel.

Para reproducir correctamente el suelo sobre que nuestro personaje se podrá desplazar, se deben añadir características de física.

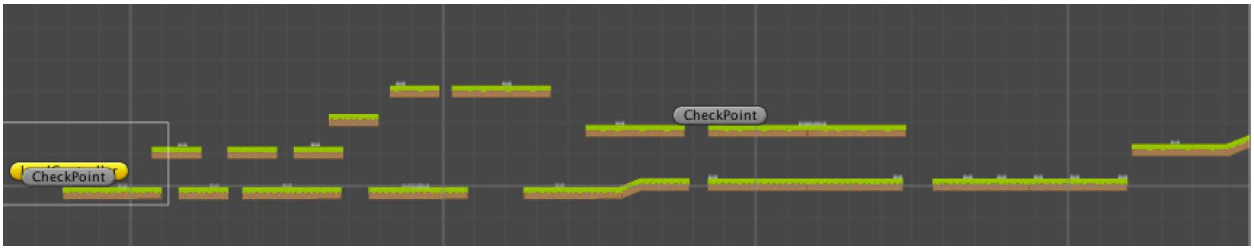
1- Rigidbody 2D: esta propiedad aplica la física de la naturaleza, como por ejemplo la masa del objeto. En nuestro diseño es importante aplicar la opción de Is Kinematic, para que no se le aplique la gravedad y se nos displace el suelo hacia abajo empujado por la gravedad.

2- Collider 2D: esta propiedad nos permitirá envolver el objeto con una malla transparente para poder controlar eventos, es decir, podremos controlar a través de esta malla si el objeto entra en contacto con otra malla o la atraviesa para poder lanzar un nuevo evento. En el suelo creamos un Box Collider 2D que se adapta perfectamente al suelo.



Para facilitar la creación del suelo o objetos que se pueden repetir en el videojuego como por ejemplo enemigos, Unity tiene objetos prefabs. Estos objetos permiten añadir características a un único objeto y luego poder traspasar estas características al resto de objetos de la misma familia. Para crear un objeto prefab únicamente se ha de crear una carpeta prefab y mover ahí el objeto que queremos que sea prefab.

Una vez creado los prefabs procedo a crear el primer nivel.



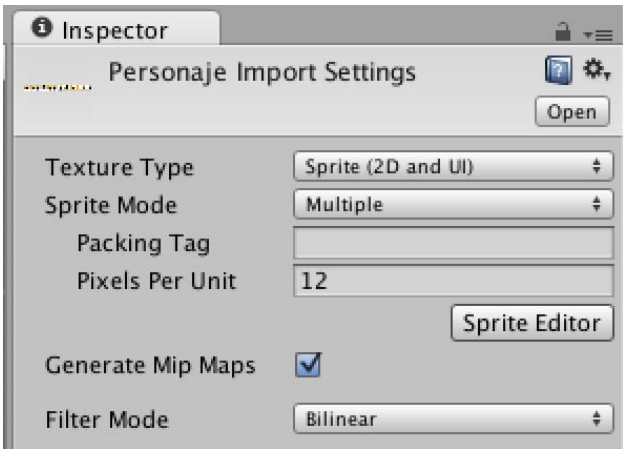
Para distribuir rápidamente el gameobject del suelo he ido duplicando los objetos con cmd + d.

6- Insertamos el personaje principal.

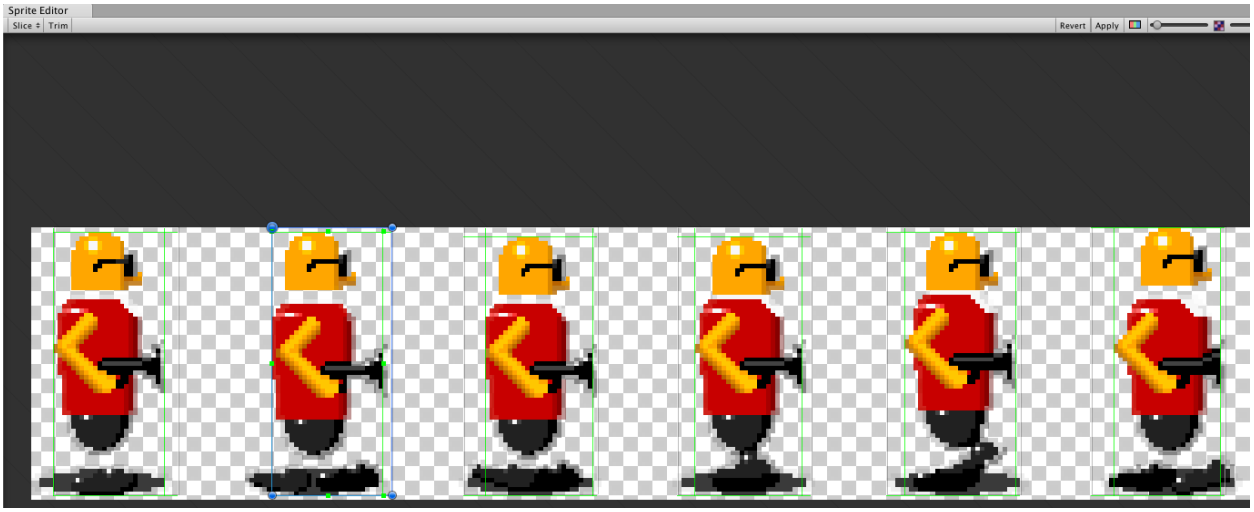
Después de buscar por internet, he decidido añadir el usuario del videojuego de Bonanza.

http://spritedatabase.net/file/15069

Para editarlo y hacer que el personaje se anime tenemos que ir al inspector y escoger la opción de Sprite Mode Multiple y editar la imagen con el Sprite Editor.



Y dentro del Sprite Editor tenemos que cortar la imagen en trocitos para hacer una animación cuando el personaje se desplace.



Una vez que ya tenemos el personaje segmentado con diferentes frames, ya lo podemos añadir en el videojuego.

7 – Caminar y saltar del personaje principal

Para conseguir que el personaje se pueda desplazar deberemos añadir un script para conseguirlo. Así que añadimos un script con el nombre playerController.cs, sobre la carpeta script. El script se realizará con C#. Este se debe añadir al personaje, simplemente seleccionando el personaje principal y arrastrar el scritp en el inspector.

El escript añadido es:

```
using UnityEngine;
using System.Collections;
public class playerController : MonoBehaviour {
//Creamos las variables publicas para controlar el salto y velocidad
    public float fuerzaSalto;
    public float velocidadMovimiento;
    // Use this for initialization
    void Start () {
    }
    // Update is called once per frame
    void Update () {
//Si pulsamos tecla espaciadora, el personaje saltará
        if(Input.GetKeyDown(KeyCode.Space)){
            GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D
>().velocity.x,fuerzaSalto);
            }
//Si pulsamos la D, desplazamos el personaje a la derecha
        if(Input.GetKey(KeyCode.D)){
            GetComponent<Rigidbody2D>().velocity = new Vector2(velocidadMovimiento,GetCom
ponent<Rigidbody2D>().velocity.y);
        }
//Si pulsamos la A, desplazamos el personaje a la Izquierda
        if(Input.GetKey(KeyCode.A)){
            GetComponent<Rigidbody2D>().velocity = new Vector2(-
velocidadMovimiento,GetComponent<Rigidbody2D>().velocity.y);
        }
}
}
```

Con el script, ya hemos conseguido que el personaje se desplace de derecha a izquierda y saltar.

8- Mejorando el doble salto

Con el script anterior hemos detectado que cada vez que pulsamos sobre la tecla espaciadora, el personaje salta.

Tenemos que mejorar este comportamiento, ahora haremos mediante programación, que el personaje detecte si está tocando el suelo para iniciar el salto y si el personaje ya está saltando solo pueda realizar un salto doble.

Para ello debemos utilizar la propiedad del Circle Collider 2D, con la opción de física hysics2D.OverlapCircle, podremos detectar cuando el personaje principal esté tocando el suelo. También deberemos añadir todo el suelo del nivel en la capa suelo para poderlo detectar.

Así que deberemos añadir la siguiente programación:

```
using UnityEngine;
using System.Collections;
public class playerController : MonoBehaviour {
//Creamos las variables publicas para controlar el salto y velocidad
    public float fuerzaSalto;
    public float velocidadMovimiento;
// Variables para controlar el doble salto, este solo en el suelo
    public Transform sueloCheck;
    public float sueloCheckRadio = 0.05f;
    public LayerMask sueloLayer;
    private bool enSuelo;
    private bool dobleSalto;
    // Use this for initialization
    void Start () {
    }
//Añadimos una nueva funcion para controlar la física
    void FixedUpdate() {
        enSuelo = Physics2D.OverlapCircle(sueloCheck.position,sueloCheckRadio,sueloLayer);
    }
    // Update is called once per frame
    void Update () {
//Verificamos si está el personaje en el suelo
        if (enSuelo) {
            dobleSalto=false;
        }
```

```
//Si pulsamos tecla espaciadora, el personaje saltará se controla tambien si está en el suelo
      if(Input.GetKeyDown(KeyCode.Space) && enSuelo){
         GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D
>().velocity.x,fuerzaSalto);
         }
      if(Input.GetKeyDown(KeyCode.Space) && !dobleSalto && !enSuelo){
         GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D
>().velocity.x,fuerzaSalto);
         dobleSalto=true;
      }
//Si pulsamos la D, desplazamos el personaje a la derecha
      if(Input.GetKey(KeyCode.D)){
         GetComponent<Rigidbody2D>().velocity = new Vector2(velocidadMovimiento,GetCom
ponent<Rigidbody2D>().velocity.y);
      }
//Si pulsamos la A, desplazamos el personaje a la Izquierda
      if(Input.GetKey(KeyCode.A)){
         GetComponent<Rigidbody2D>().velocity = new Vector2(-
velocidadMovimiento,GetComponent<Rigidbody2D>().velocity.y);
      }
}
}
```

9 – Introduciendo los primeros elementos de dificultad

En el nivel, nuestro personaje deberá saltar entre plataformas y eliminar enemigos, pero también se encontrará con obstáculos. En este paso vamos añadir el primer obstáculo, unos pinchos que al tocarlos nos eliminará.

Así que introducimos los pinchos al videojuego y los convertimos en un objeto prefab para poderlo distribuir por todo el escenario sin que tengamos que repetir la configuración una y otra vez.

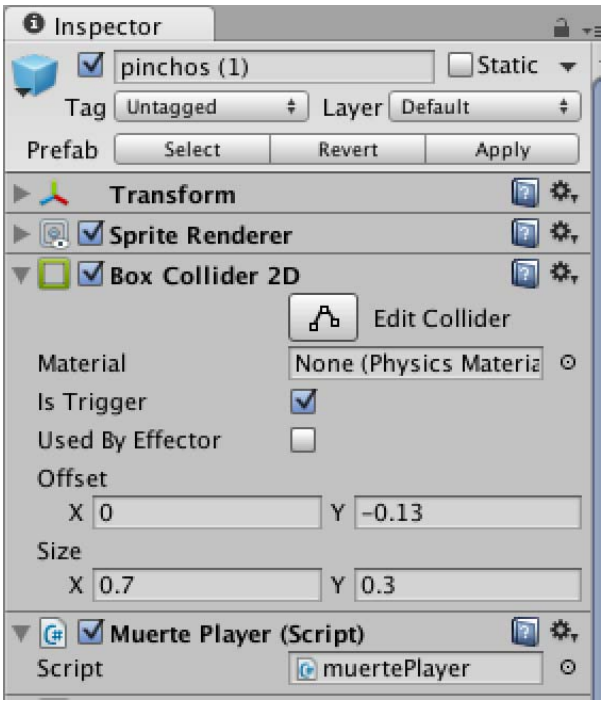
En el objeto se le deberé añadir una física, en este caso se le añade un Box Collider y escogemos la opción de Is Trigger, esta es la manera de detectar la colisión del Player con los pinchos. Ahora el Player al atravesar el Box Collider de los pinchos lanzará un evento, programado en los scripts.

Para que los pinchos eliminen el personaje, creamos un par de scripts. Un script que se llamará levelController.cs y otro que se llamará muertePlayer.cs.

muertePlayer.cs

```
using UnityEngine;
using System.Collections;
public class muertePlayer : MonoBehaviour {
// Creamos una variable para llamar a otro script
   private levelController controlDeNivel;
   void Start () {
//Buscamos el primer objeto de levelController
      controlDeNivel = FindObjectOfType<levelController> ();}
   // Update is called once per frame
   void Update () {
   }
// Declaramos una nueva función para detectar cuando el player pase por los pinchos
   void OnTriggerEnter2D(Collider2D col){
      if (col.name == "Player") {
         controlDeNivel.CheckRespawn();
      }
   }
}
```

El script de muertePlayer.cs se lo aplicamos al objeto pinchos para que cuando se detecte la colisión del Player con los pinchos llame al escript de levelController.cs y se inicie de nuevo el videojuego.



10 – Creación del checkPoints

Para crear puntos de checkpoints debemos crear objetos, denominados checkPoints, que lo que hacemos es detectar si el player ha colisionado con el checkPoint, y si es así haremos que el Player se inicialice desde esa posición. Para realizarlo he tenido que añadir dos scripts, checkPoint.cs y levelController.cs.

checkPoint.cs

```
using UnityEngine;
using System.Collections;
public class checkPoint : MonoBehaviour {
// Creamos una variable publica para acceder al levelController
    public levelController nivel;
    // Use this for initialization
    void Start () {
        nivel=FindObjectOfType<levelController>();
    }
    // Update is called once per frame
```

```
    void Update () {
    }
// Controlamos el paso de un chekpoint
    void OnTriggerEnter2D(Collider2D col){

    if(col.name =="Player"){
          nivel.actualCheckpoint = gameObject;
    }
}
}
```

levelController.cs

```
using UnityEngine;
using System.Collections;
public class levelController : MonoBehaviour {
//Creamos nuevas variables para controlar el CheckPoint
    public GameObject actualCheckpoint;
    private playerController jugador;
    // Use this for initialization
    void Start () {
//Hacemos que a jugador busque un tipo de playerController
       jugador = FindObjectOfType <playerController>();
    }
    // Update is called once per frame
    void Update () {
    }
    public void CheckRespawn(){
       Debug.Log("Morissteees");
//Colocamos la posición del personaje al Checkpoint
       jugador.transform.position = actualCheckpoint.transform.position;
    }
}
```

11 – Animamos nuestro personaje

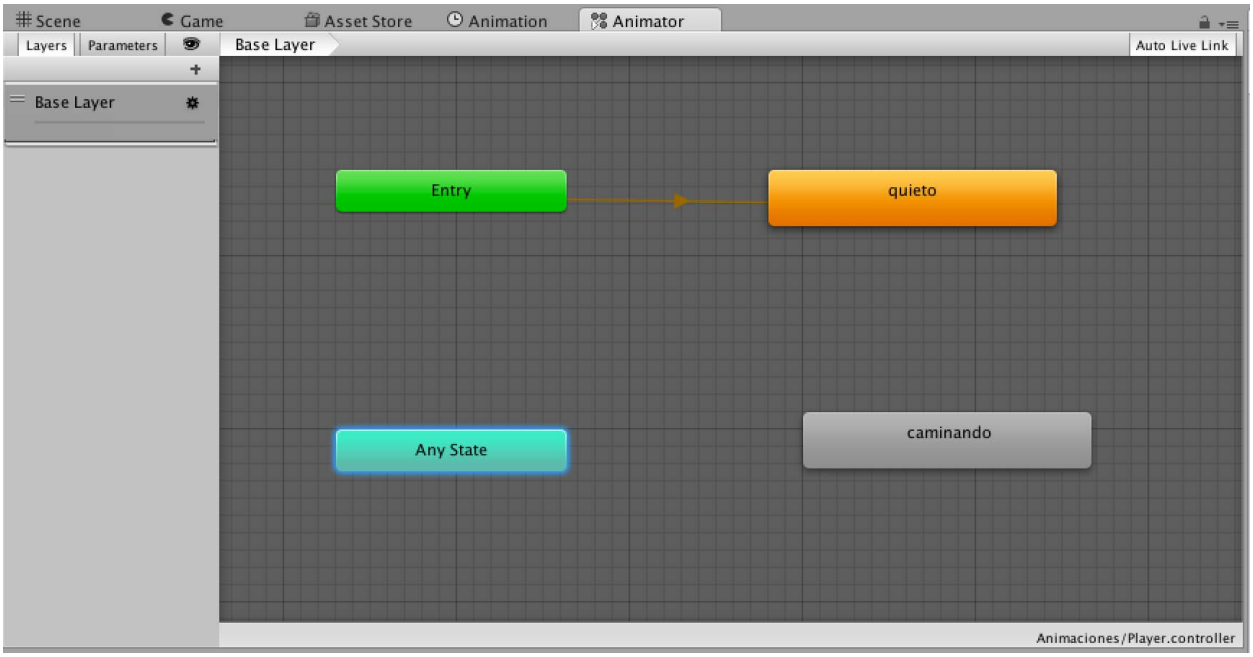
Lo primero que tenemos que hacer es ir a la ventana de Animation, dentro de Windows -> Animation.

Creamos dos animaciones la de quieto y caminando.

La animación de caminando se realiza mediante una animación de interpolación de movimiento, insertando los diferentes frames del movimiento del Player.

Una vez creadas las animaciones accedemos a la carpeta de animaciones donde las hemos guardado y observamos que tenemos un fichero llamado Player.

Si realizamos doble clic sobre el fichero Player, nos aparece lo siguiente:



Ahora tendremos que crear transacciones entre los estados.

quieto → caminando

caminando → quieto

Con esto ya tenemos la animación de parado a caminando y de caminando a parado e introducimos un valor float con un parámetro speed.

Ahora debemos aplicar programación para reproducir esta animación, así que modificamos el playerController.cs.

```
using UnityEngine;
using System.Collections;
public class playerController : MonoBehaviour {
```

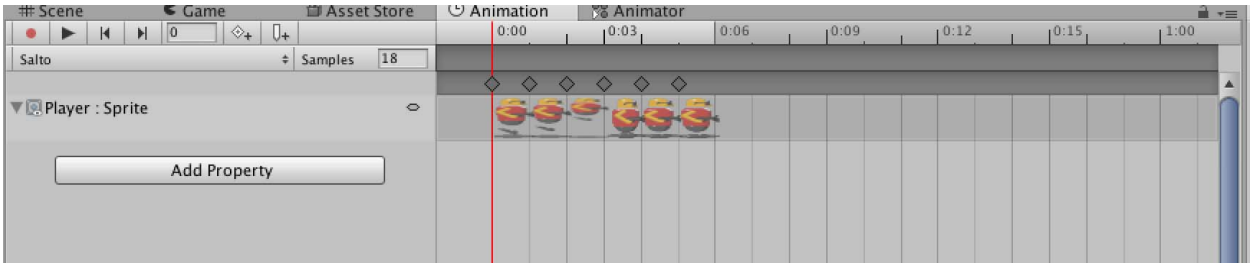
```
    //Creamos las variables publicas para controlar el salto y velocidad
    public float fuerzaSalto;
    public float velocidadMovimiento;
    // Variables para controlar el doble salto, este solo en el suelo
    public Transform sueloCheck;
    public float sueloCheckRadio = 0.05f;
    public LayerMask sueloLayer;
    private bool enSuelo;
    private bool dobleSalto;
    //Variables para controlar la animación del player
    private Animator animacion;
    // Use this for initialization
    void Start () {
        //Iniciamos la animación del player
        animacion = GetComponent<Animator>();
    }
    //Añadimos una nueva funcion para controlar la física
    void FixedUpdate() {
        enSuelo = Physics2D.OverlapCircle(sueloCheck.position,sueloCheckRadio,sueloLayer);
    }
    // Update is called once per frame
    void Update () {
        //Verificamos si está el personaje en el suelo
        if (enSuelo) {
            dobleSalto=false;
        }
        //Si pulsamos tecla espaciadora, el personaje saltará se controla tambien si está en el suelo
        if(Input.GetKeyDown(KeyCode.Space) && enSuelo){
            GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D
>().velocity.x,fuerzaSalto);
        }
        if(Input.GetKeyDown(KeyCode.Space) && !dobleSalto && !enSuelo){
            GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D
>().velocity.x,fuerzaSalto);
            dobleSalto=true;
        }
        //Si pulsamos la D, desplazamos el personaje a la derecha
        if(Input.GetKey(KeyCode.D)){
```

```
        GetComponent<Rigidbody2D>().velocity = new Vector2(velocidadMovimiento,GetCom
ponent<Rigidbody2D>().velocity.y);
    }
    //Si pulsamos la A, desplazamos el personaje a la Izquierda
    if(Input.GetKey(KeyCode.A)){
        GetComponent<Rigidbody2D>().velocity = new Vector2(-
velocidadMovimiento,GetComponent<Rigidbody2D>().velocity.y);
    }
    //Controlamos el parametro speed, de atributo de la animación para que detectemos si cami
na o no mediante la posicion X
    animacion.SetFloat("speed",Mathf.Abs(GetComponent<Rigidbody2D>().velocity.x));
  }
}
```

12 – Animamos nuestro personaje en el salto

Creamos la animación del salto tal y como habíamos realizado en el caminar.

Abrimos las pestaña de Animation y configuramos la animación del salto.



Repetimos el proceso de animación de interpolación de movimiento, insertando los diferentes frames que reproducen la animación del salto.

Ahora tendremos que crear transacciones entre los estados.

Any State → Salto

Sato → Quieto

13 – Creando puntos al coger ítems

Para introducir un contador de ítems, vamos a la opción de GameObject -> UI -> Text.

Utilizaremos este texto para indicar los ítems que cogemos, en nuestro videojuego monedas, e incluimos un nuevo scritp, puntosController.cs. Este script se lo aplicaremos a la UI de Text de textoPuntos.

puntosController.cs

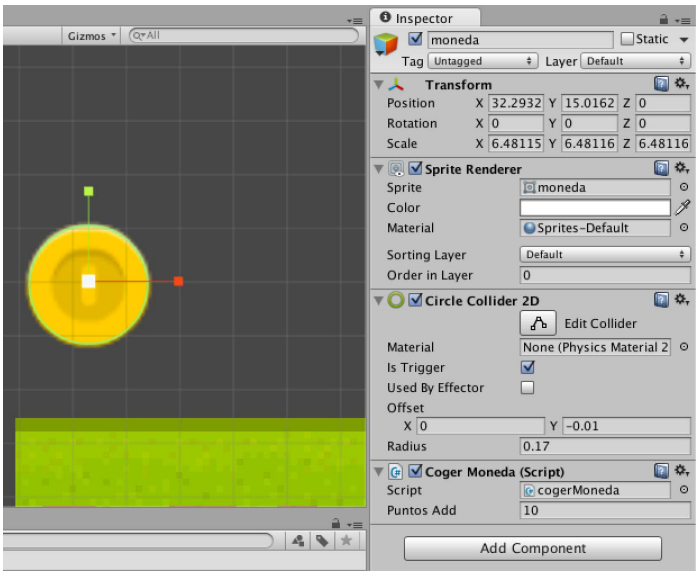
```
using UnityEngine;
using System.Collections;
//Habilito la funcion de UI para poder trabajar con ella
using UnityEngine.UI;
public class puntosController : MonoBehaviour {
//Defino la variable para la puntación
    public static int puntos;
    Text texto;
    // Use this for initialization
    void Start () {
// Inicio los valores del marcador
        texto = GetComponent<Text>();
        puntos = 0;
    }
    // Update is called once per frame
    void Update () {
        if (puntos < 0)
            puntos = 0;
        texto.text = "" + puntos;
    }
//Añado un nuevo metodo para sumar puntos cuando cojamos una moneda
    public static void AgregarPuntos(int puntosParaSumar)
    {
        puntos += puntosParaSumar;
    }
```

```
//Añado un nuevo metodo para resetear el marcador
    public static void Reset()
    {
        puntos = 0;
    }
}
```

También añadimos las monedas en el escenario.

A la moneda le añadimos un Circle Collider para detectar cuando el personaje toque la moneda.

Creamos un nuevo script monedaCoger.cs y lo añadimos a la moneda. Verificamos el funcionamiento y si está todo OK, pasamos la moneda a un prefab, así ahora podemos distribuir la moneda por todo el nivel y cada vez que el player coja la moneda tendremos el valor que especifiquemos en la variable pública Puntos Add.



monedaCoger.cs

```
using UnityEngine;
using System.Collections;
public class cogerMoneda : MonoBehaviour {
//creamos variable para sumar los puntos
    public int puntosAdd;
    // Use this for initialization
```

```
    void Start () {
    }
    // Update is called once per frame
    void Update () {
    }

//Añadimos el metodo para que se controle cuando el Player pase por la moneda
    void OnTriggerEnter2D(Collider2D col){
//Controlamos que haya colisión con el Player
        if (col.name == "Player") {
            puntosController.AgregarPuntos (puntosAdd);
//destruimos la moneda
            Destroy (gameObject);
        }
    }
}
```

14 – Creamos los enemigos

Lo primero que tenemos que hacer es escoger el enemigo, en nuestro videojuego he optado por coger un Orco.

http://spritedatabase.net/files/arcade/640/Sprite/Orc.gif

Repetimos el proceso de escoger la imagen y hacerla multiple para luego poder realizar una animación sobre el enemigo.

Aplicamos un RigidBody2D para añadir la física al orco y le añadimos un BoxCollider para detectar la colisión del enemigo con el Player.

Ahora para que nuestro Player quede eliminado cuando choque con el enemigo, añado un nuevo script, enemigo.cs.

Para poner limites al enemigo, añadimos GameObject -> Empty con sus BoxCollider y así interceptamos cuando el enemigo se cruza con el limite. Los limites lo añadimos en prefabs para que los podamos añadir por todo el escenario y le añadimos una etiqueta de limite para poder detectar el tag en los prefabs.

Enemigo.cs

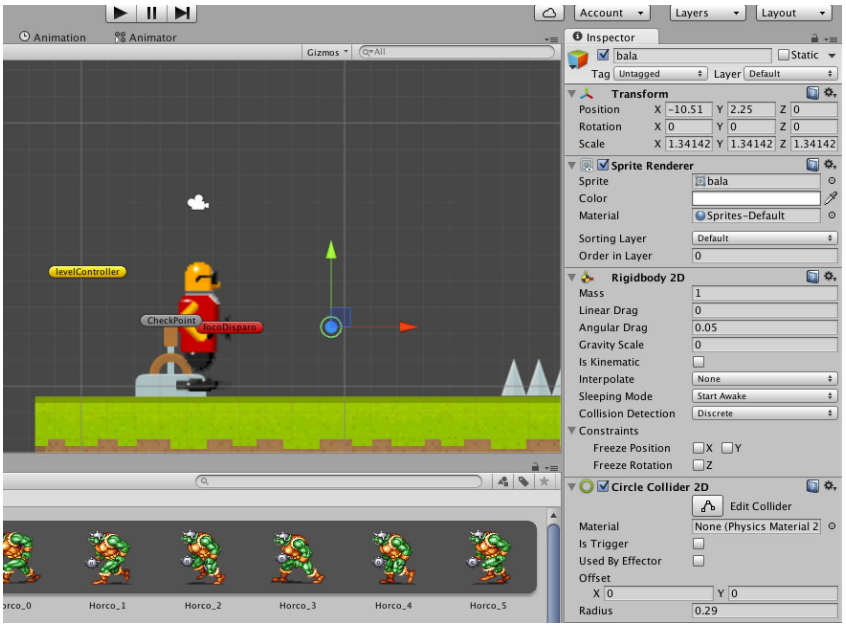
```
using UnityEngine;
using System.Collections;
public class enemigo : MonoBehaviour {
//Variable para controlar el movimiento del enemigo
   public float velocidadMovimiento;
   public bool moverDerecha;
   // Use this for initialization
   void Start () {
   }
   // Update is called once per frame
   void Update () {
      if (moverDerecha) {
         GetComponent<Rigidbody2D> ().velocity = new Vector2 (velocidadMovimiento, GetCo
mponent<Rigidbody2D> ().velocity.y);
      }
      else {
         GetComponent<Rigidbody2D> ().velocity = new Vector2 (-
velocidadMovimiento, GetComponent<Rigidbody2D> ().velocity.y);
```

```
    }
  }
  //Ponemos un limite para que el enemigo choque y cambie de sentido y lo identificamos con el tag limite
  void OnTriggerEnter2D(Collider2D col)
  {
    if (col.tag == "limite" && moverDerecha == true) {
      moverDerecha = false;
    } else if (col.tag == "limite" && moverDerecha == false) {
      moverDerecha = true;
    }
  }
}
```

15 – Poder disparar

Primero tengo que crear un GameObject -> Create Empty y le ponemos el nombre de focoDisparo y se lo asignamos al Player.

Creamos la bala mediante Fireworks, un simple donut de color gris, y lo añadimos a los gráficos del proyecto del videojuego y la convertimos en una prefab. Le asignamos un RigidBody 2D y un Circle Collider2D con la siguiente configuración:



Ahora mediante programación asignamos la funcionalidad de la bala. Editamos nuevamente controllerPlayer.cs.

```
//Variable para controlar el disparo
public Transform focoDisparo;
//Variable para controlar la bala
public GameObject bala;

//Creamos balas cuando pulsamos la letra M
 if (Input.GetKey(KeyCode.M)) {
Instantiate(bala,focoDisparo.position,focoDisparo.rotation);
 }
```

Y en el objeto de Player nos aparece nuevos campos creados en el script, estos los debemos llenar con los objetos que hemos creado de la siguiente manera:

Ahora para controlar el movimiento de la bala, procedo a crear un nuevo scritp disparoController.cs y lo asignamos a la bala.

disparoController.cs

```
using UnityEngine;
using System.Collections;
public class disparoController : MonoBehaviour {
   //Varaiable para controlar la velocidad de la bala
   public float velocidadBala;
   //Variable para controlar la desaparición de las balas
```

```
    private playerController Player;
    // Use this for initialization
    void Start () {
        //Buscamos el script de playerController
        Player = FindObjectOfType<playerController> ();
    //Hacemos que nuestro player pueda disparar hacia los 2 lados
        if (Player.transform.localScale.x > 0)
            velocidadBala = -velocidadBala;
    }
    // Update is called once per frame
    void Update () {
        //Hacemos desplazar la bala
        GetComponent<Rigidbody2D>().velocity = new Vector2(-
velocidadBala,GetComponent<Rigidbody2D>().velocity.y);
        //Hacemos que las balas desaparencan cuando estén 25 posiciones por delante del player
        if (transform.position.x > Player.transform.position.x+25){
            Destroy (gameObject);
        }
    }
    //Hacemos eliminar el enemigo
    void OnTriggerEnter2D(Collider2D col){
      //Con el tag de enemigo en el enemigo conseguimos que cuando la bala choque desaparezca
        if (col.tag == "enemigo") {
            //Indicamos eliminar el enemigo
            Destroy (col.gameObject);
            //Indicamos destruir la bala
            Destroy (gameObject);
        //Hacemos que las balas desaparezcan cuando choquen con elementos del escenario
        } else if (col.tag == "escenario") {
            Destroy (gameObject);
        }
    }
}
```

Añadimos un Tag al enemigo como enemigo, para poderlo detectar la colisión con las balas y así eliminar el enemigo.

16 – Sumar puntos al eliminar enemigos

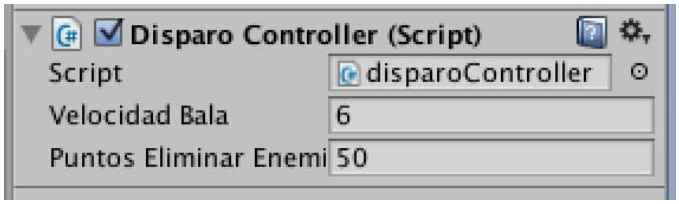
Para sumar puntos al eliminar a un enemigo debemos modificar el script de disparoController.cs, debemos añadir una variable:

```
//Variable para sumar puntos al eliminar enemigos
public int puntosEliminarEnemigos;
```

para sumar los puntos y añadir el siguiente código:

```
//sumar puntos al eliminar enemigos
puntosController.AgregarPuntos(puntosEliminarEnemigos);
```

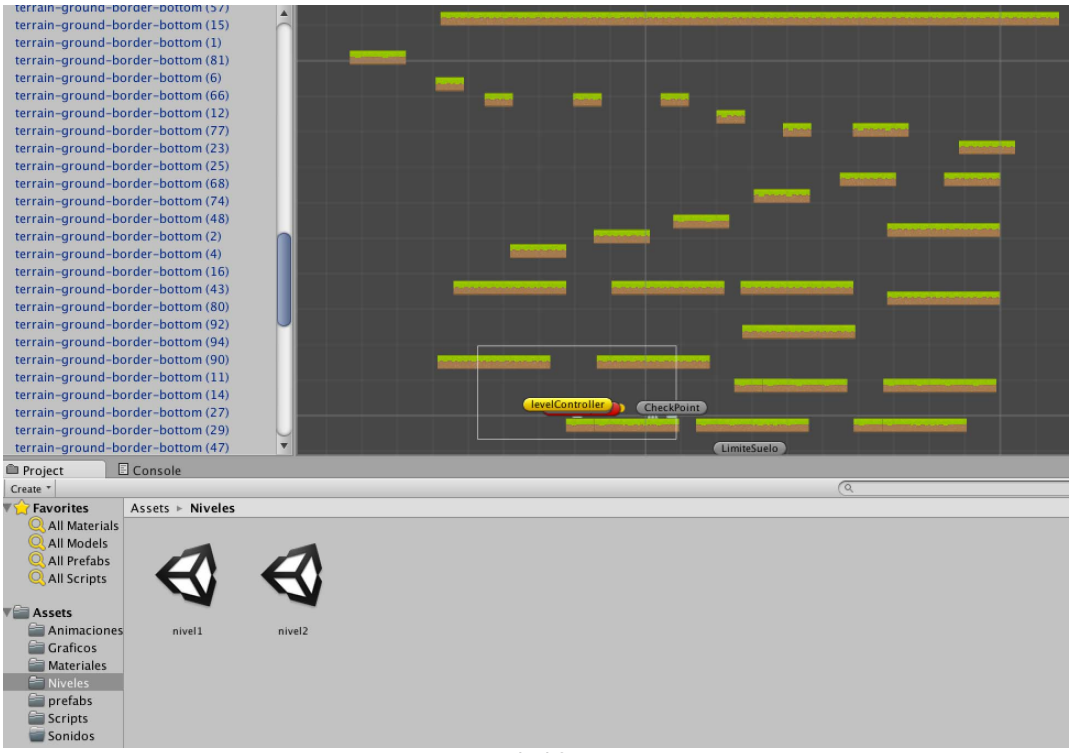
Ahora al eliminar un enemigo sumaremos puntos, como la variable puntosElimianrEnemigos es pública podremos poner el valor que queramos, en nuestro ejemplo ponemos 50 puntos al eliminar un enemigo.



17 – Creamos el segundo nivel

Creamos un segundo nivel, haciendo menú File -> New Scene. Guardamos el nuevo nivel como nivel2 dentro de la carpeta creada en los Assets Niveles.

Ahora procedo ha añadir los gameobjects que tenemos creados del primer nivel, como el Player, los pinchos, los limites, las monedas, el suelo, etc.



nivel 2

18 – Pasamos al segundo nivel

Lo primero que tenemos que hacer es añadir un gráfico en el nivel que indique que al cruzarlo pasaremos al segundo nivel. En nuestro caso una puerta que al situarnos encima y pulsar la tecla K, pasaremos al siguiente nivel.

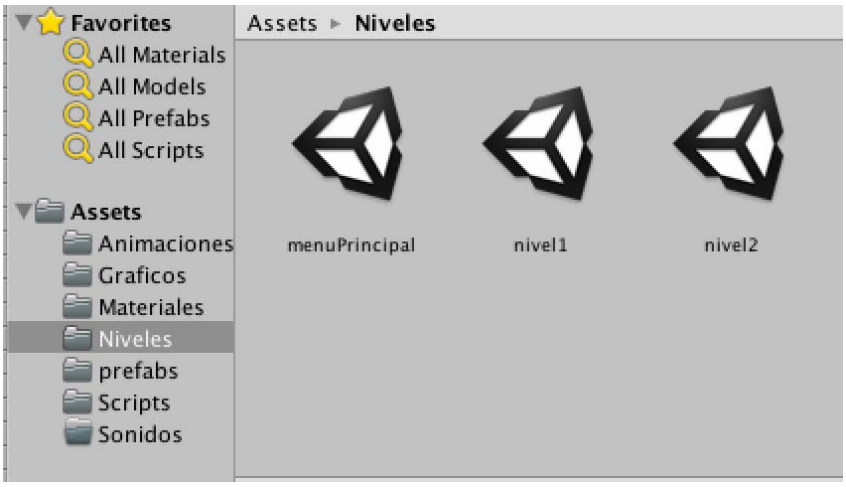
Puerta Nivel 2

Después creamos el siguiente script (pasarNivel.cs) para que cuando crucemos la puerta pasemos al siguiente nivel.

```
using UnityEngine;
using System.Collections;
public class pasarNivel : MonoBehaviour {
//Creamos las variables para controlar donde estar el usuario
   private bool  playerenNivel;
//Variable para especificar el nuevo nivel
   public string nuevoNivel;
   // Use this for initialization
   void Start () {
      playerenNivel = false;
   }
   // Update is called once per frame
   void Update () {
//Si nos situamos en la puerta y pulsamos K, se cargará el nuevo nivel
      if (Input.GetKeyDown(KeyCode.K) && playerenNivel){
         Application.LoadLevel(nuevoNivel);
      }
   }
// Controlamos si el player está en el nivel o no.
   void OnTriggerEnter2D(Collider2D col){
      if (col.name == "Player"){
         playerenNivel = true;
      }
   }
   void OnTriggerExit2D(Collider2D col){
      if (col.name == "Player"){
         playerenNivel = false;
      }
   }
}
```

19 – Creamos el menú principal

Para crear el nuevo menú debemos añadir una nueva escena al proyecto, mediante File -> New Scene.



Y añadirla al proyecto mediante File -> Build Settings.

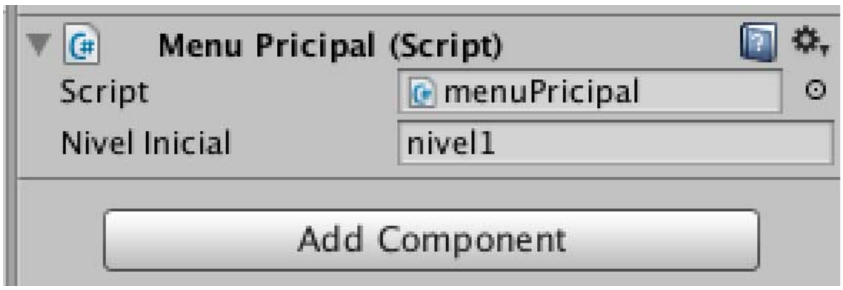
Una vez que ya tenemos el nivel en el proyecto, debemos ir a GameObject -> UI -> Canvas. Una vez creada la zona de canvas, vamos a la opción de GameObject -> UI -> Button. Modificamos los batones para añadir el texto que queremos, etiqueta, tamaño, etc.

Creamos un nuevo script, menuPrincipal.cs.

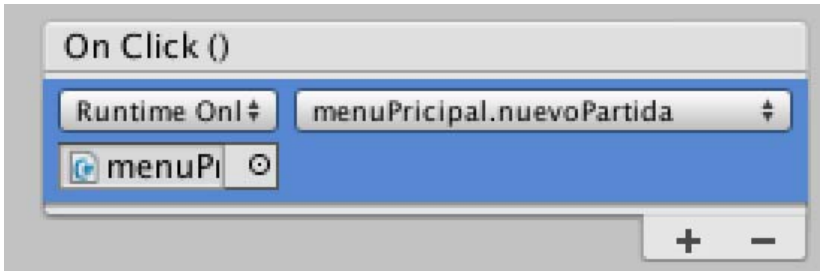
menuPrincipal.cs

```
using UnityEngine;
using System.Collections;
public class menuPricipal : MonoBehaviour {
//eliminamos los metodos Start y Update, no hacen falta.
   //Especifico el nivel en el que se empieza el videojuego
   public string nivelInicial;
   //Especifico metodo para el botón empezar el videojuego
   public void nuevoPartida(){
      Application.LoadLevel(nivelInicial);
   }
   //Especifico método para el botón Exit
   public void exitGame(){
      Application.Quit();
   }
}
```

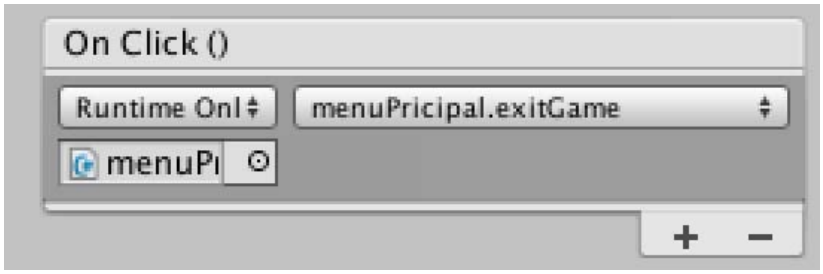
Añadimos el script al objeto Canvas creado con el nombre menuPricipal y en la variable pública creada, le añadimos con que nivel queremos que empiece el nivel, nivel1.



Después configuramos el botón Empezar, añadiendo una acción, esta nueva acción la debemos especificar el script de menuPrincipal y en la opción de Runtime Only especificar al función de nuevo.Partida defenido en el script.



En el botón de Salir, añadimos una nueva acción le especificamos el script de menuPrincipal y en la opción de Runtime Only especificar la función de menuPrincipal.exitGame.



20 - Añadimos sonido envolvente en el videojuego

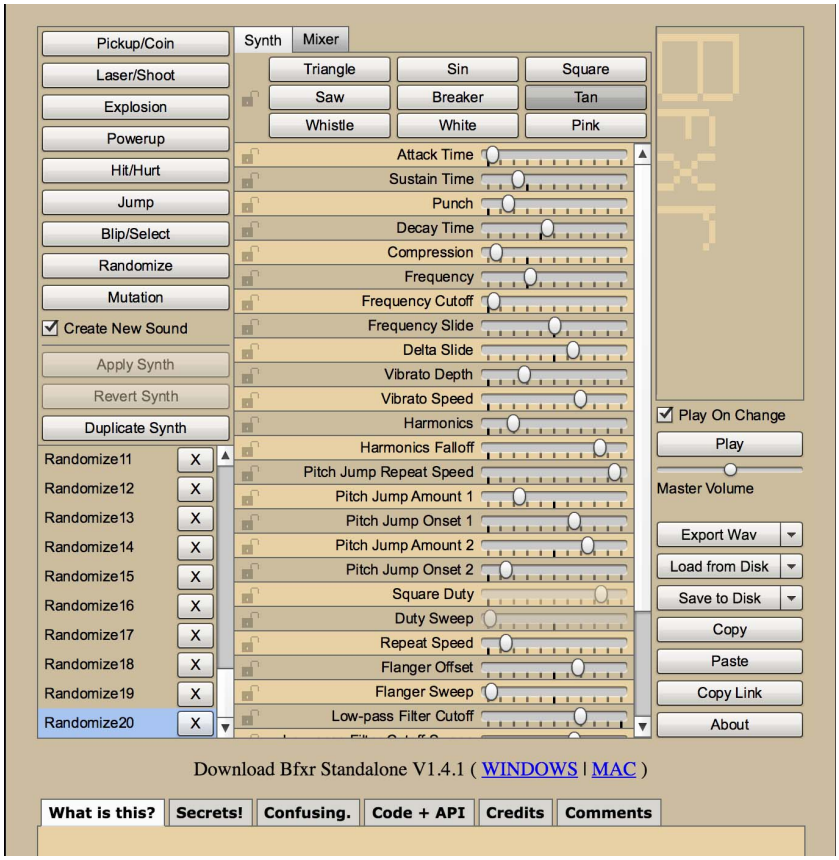
Para crear el sonido de fondo, creamos un GameObject Empty en el escenario, le ponemos el nombre de SonidoFondo y le añadimos un componente de Audio, Audio -> Audio Source y le añadimos la melodía que deseamos que suene cuando empiece el

videojuego. Es importante seleccionar la opción de Loop para que una vez que haya acabado la melodía vuelva a reproducirse.

La melodía la descargamos de la web http://incompetech.com/music/royalty-free.

21 - Añadimos efectos sonoros en el videojuego

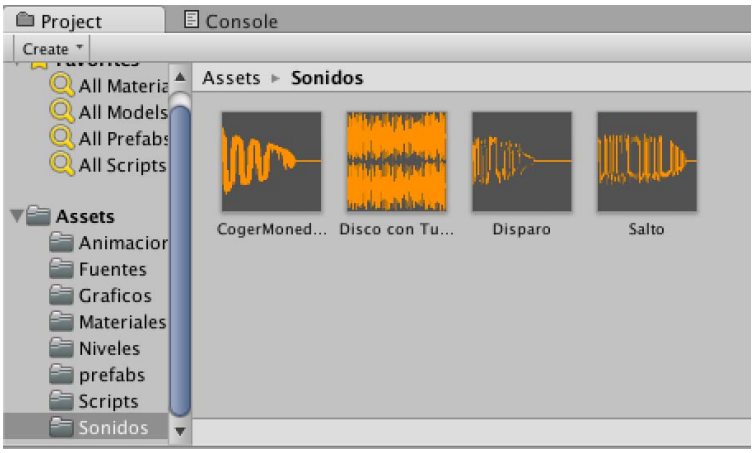
Para crear los efectos sonoros del videojuego, los hemos creado con la web http://www.bfxr.net/.



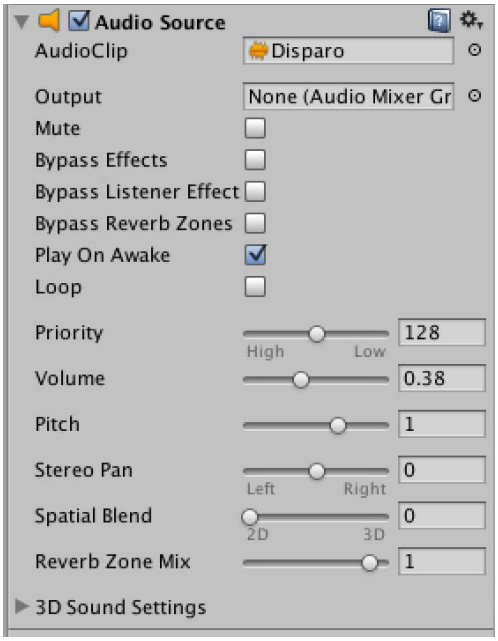
Este web nos permite crear sonidos sonoros de forma muy sencillamente, hay ejemplos y disponemos de barras desplazadoras para poder ajustar el sonido a nuestras necesidades. Una vez conseguido el sonido deseado, lo podemos exportar en formato wav.

Para el videojuego he creado el sonido del disparo, salto y coger monedas.

Una vez creados los efectos de sonido mediante la herramienta descrita, debemos añadir los sonidos al videojuego, así que los arrastramos al apartado Assets, la carpeta creada de sonidos.

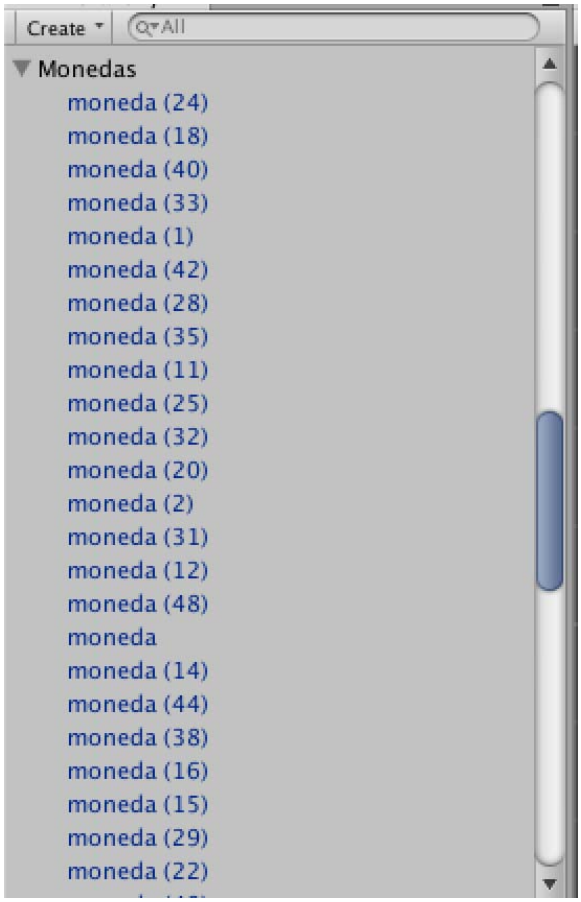


Para crear el sonido cuando disparamos, debemos ir al prefab creado de la bala y añadir un componente de audio -> Audio source y le especificamos el sonido del disparo.



Es importante poner la opción Play On Awake, ahora cada vez que disparamos conseguimos que se reproduzca el sonido del disparo.

Para crear el sonido al coger las monedas, primero tenemos que ir al prefab de la moneda y añadir el componente de audio -> Audio source y especificar el sonido de coger la moneda. Después debemos agrupar todas las monedas en un gameobject empty, así agrupamos todas las monedas y así poder asignar fácilmente el sonido a todas las monedas.



Y finalmente debemos modificar el script cogerMonedas.cs de la siguiente manera.

```
using UnityEngine;
using System.Collections;
public class cogerMoneda : MonoBehaviour {
//Creamos variable para sumar los puntos
   public int puntosAdd;
//Creamos una variable para reproducir un sonido al coger las monedas
   public AudioSource cogerMonedaSonido;
```

```
    // Use this for initialization
    void Start () {
    }
    // Update is called once per frame
    void Update () {
    }
//Añadimos el metodo para que se controle cuando el Player pase por la moneda
    void OnTriggerEnter2D(Collider2D col){
//Controlamos que haya colisión con el Player
        if (col.name == "Player") {
//Definimos la reproducción del sonido al coger la moneda
            cogerMonedaSonido.Play();
//Definimos agregar puntos
            puntosController.AgregarPuntos (puntosAdd);
//Destruimos la moneda
            Destroy (gameObject);
        }
    }
}
```

Ahora al coger las monedas se reproducirá el sonido.

Para añadir el sonido del efecto del salto, debemos crear un nuevo gameobject empty dentro del Player, este le llamaremos Salto y le asignaremos el sonido de salto.

Y modificamos el script playerController.cs de la siguiente manera para que se reproduzca el sonido del salto.

```
using UnityEngine;
using System.Collections;
public class playerController : MonoBehaviour {
    //Creamos las variables publicas para controlar el salto y velocidad
    public float fuerzaSalto;
    public float velocidadMovimiento;
    // Variables para controlar el doble salto, este solo en el suelo
    public Transform sueloCheck;
    public float sueloCheckRadio = 0.1f;
    public LayerMask sueloLayer;
    private bool enSuelo;
```

```
    private bool dobleSalto;
    //Variables para controlar la animación del player
    private Animator animacion;
    //Variable para quitar la sensación de patinar el Player
    private float moveVelocity;
    //Variable para controlar el disparo
    public Transform focoDisparo;
    //Variable para controlar la bala
    public GameObject bala;
    //Variable para controlar el sonido del salto
    public AudioSource sonidoSalto;
    // Use this for initialization
    void Start () {
        //Iniciamos la animación del player
        animacion = GetComponent<Animator>();
    }
    //Añadimos una nueva funcion para controlar la física
    void FixedUpdate() {
        enSuelo = Physics2D.OverlapCircle(sueloCheck.position,sueloCheckRadio,sueloLayer);
    }
    // Update is called once per frame
    void Update () {
        //Verificamos si está el personaje en el suelo
        if (enSuelo) {
            dobleSalto=false;
        }
        animacion.SetBool("grounded",enSuelo);
        //Si pulsamos tecla espaciadora, el personaje saltará se controla tambien si está en el suelo
        if(Input.GetKeyDown(KeyCode.Space) && enSuelo){
            GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D
>().velocity.x,fuerzaSalto);
            //Reproducimos el sonido del salto
            sonidoSalto.Play();
        }
        if(Input.GetKeyDown(KeyCode.Space) && !dobleSalto && !enSuelo){
            GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D
>().velocity.x,fuerzaSalto);
            dobleSalto=true;
```

```
        }
        //Iniciamos la variable a 0
        moveVelocity = 0f;
        //Si pulsamos la D, desplazamos el personaje a la derecha
        if(Input.GetKey(KeyCode.D)){
            //GetComponent<Rigidbody2D>().velocity = new Vector2(velocidadMovimiento,GetCo
mponent<Rigidbody2D>().velocity.y);
            //Con este metodo conseguimos que no patine el player
            moveVelocity=velocidadMovimiento;
        }
        //Si pulsamos la A, desplazamos el personaje a la Izquierda
        if(Input.GetKey(KeyCode.A)){
            //GetComponent<Rigidbody2D>().velocity = new Vector2(-
velocidadMovimiento,GetComponent<Rigidbody2D>().velocity.y);
            //Con este metodo conseguimos que no patine el player
            moveVelocity=-velocidadMovimiento;
        }
        //Creamos balas cuando pulsamos la letra M
        if (Input.GetKeyDown(KeyCode.M)) {
            Instantiate(bala,focoDisparo.position,focoDisparo.rotation);
        }
        GetComponent<Rigidbody2D>().velocity = new Vector2 (moveVelocity, GetComponent<
Rigidbody2D> ().velocity.y);
        //Controlamos el parametro speed, de atributo de la animación para que detectemos si cami
na o no mediante la posicion X
        animacion.SetFloat("speed",Mathf.Abs(GetComponent<Rigidbody2D>().velocity.x));
        // Controlamos que la imagen se gire cuando nos desplacemos a la izquierda
        if (GetComponent<Rigidbody2D>().velocity.x>0)
            transform.localScale = new Vector3(1f,1f,1f);
        else if (GetComponent<Rigidbody2D>().velocity.x <0)
            transform.localScale = new Vector3(-1f,1f,1f);
    }
}
```

22 - Añadimos partículas al personaje principal

Para mejorar el diseño visual del videojuego voy añadir un par de sistemas de partículas que ofrece Unity3D.

Para añadir partículas, debemos ir al menú GameObject -> Particle system. Creamos dos sistemas de partículas diferentes, una para cuando nuestro personaje fallece (particulaMuerte) y otro sistema de partículas (particulaRevive) para cuando nuestro personaje reaparece en pantalla. Podemos jugar con la configuración de las partículas dejándolo de la siguiente manera:

Una vez creadas las partículas deberemos añadirlos al levelController, pero para poderlos añadir, debemos modificar primero el script de levelController.cs, de la siguiente manera:

```
using UnityEngine;
using System.Collections;
public class levelController : MonoBehaviour {
```

```
//Creamos nuevas variables para controlar el CheckPoint
    public GameObject actualCheckpoint;
    private playerController jugador;
//Creamos nuevas variables para controlar la ejecución de las particulas
    public GameObject particulaMuerte;
    public GameObject particulaRevive;
//Creamos variable para retrasar la apareción del player
    public float reviveDelay;
    // Use this for initialization
    void Start () {
//Hacemos que a jugador busque un tipo de playerController
        jugador = FindObjectOfType <playerController>();
    }
    // Update is called once per frame
    void Update () {
    }
    public void CheckRespawn(){
//Para controlar el tiempo de revivir.
        StartCoroutine("RespawnPlayerCorutina");
    }
    public IEnumerator RespawnPlayerCorutina()
    {
//Ejecutamos la particula de muerte y hacemos que desaparezca el player
        Instantiate(particulaMuerte,jugador.transform.position,jugador.transform.rotation);
        jugador.enabled = false;
        jugador.GetComponent<Renderer>().enabled = false;
//Retrasamos el revivir del jugador
        yield return new WaitForSeconds(reviveDelay);
        jugador.transform.position = actualCheckpoint.transform.position;
//Ejecutamos la particula de revivir y hacemos que aparezca el player
        Instantiate(particulaRevive,jugador.transform.position,jugador.transform.rotation);
        jugador.enabled = true;
        jugador.GetComponent<Renderer>().enabled = true;
    }
}
```

23 – Modificamos el marcador

Para hacer el marcador de puntos más atractivo, descargamos la fuente Comic Sans de la página web http://www.fontspace.com/category/comic%20sans, y la cargamos en el videojuego. Modificamos el objeto GameObject -> UI -> Text y lo colocamos a la derecha de la pantalla.

24 – Finalizamos el menú principal

En el menú principal añadimos los botones de ayuda y créditos.

Para crear el nuevo menú debemos añadir una nuevas escenas al proyecto, mediante File -> New Scene.

Y añadirla al proyecto mediante File -> Build Settings.

Una vez que ya tenemos los niveles en el proyecto, debemos ir a GameObject -> UI -> Canvas. Una vez creada la zona de canvas, vamos a la opción de GameObject -> UI -> Button. Modificamos los batones para añadir el texto que queremos, etiqueta, tamaño, etc.

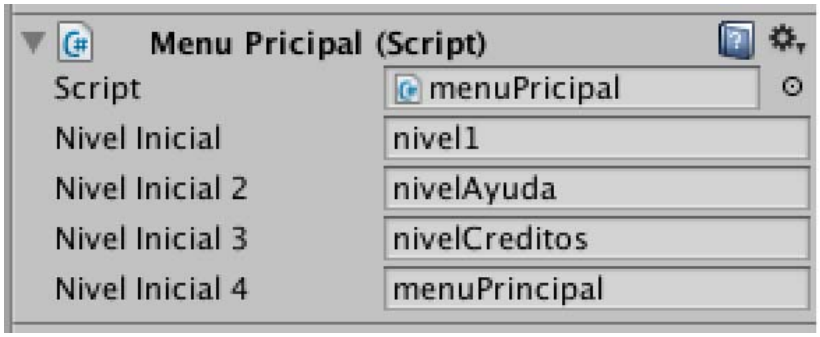
Modificamos el script, menuPrincipal.cs de la siguiente manera:

menuPrincipal.cs

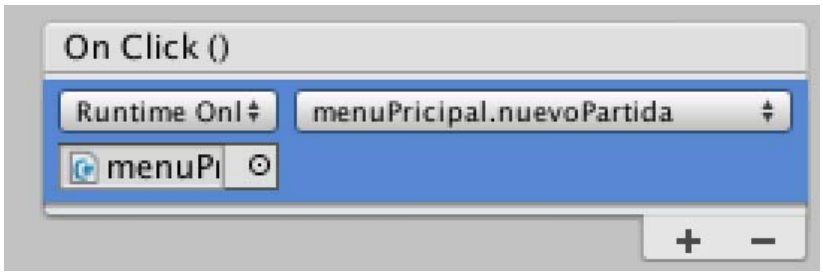
```
using UnityEngine;
using System.Collections;
public class menuPricipal : MonoBehaviour {
//eliminamos los metodos Start y Update, no hacen falta.
   //Especifico el nivel en el que se empieza el videojuego
   public string nivelInicial;
   //Variable para la pantalla de ayuda
   public string nivelInicial2;
   //Variable para la pantalla de créditos
   public string nivelInicial3;
   //Variable para volver al menú principal
   public string nivelInicial4;
   //Especifico metodo para el botón empezar el videojuego
   public void nuevoPartida(){
      Application.LoadLevel(nivelInicial);
   }
   //Especifico metodo para el botón cargar pantalla ayuda
   public void nuevoPartida2(){
      Application.LoadLevel(nivelInicial2);
   }
   //Especifico metodo para el botón cargar pantalla créditos
   public void nuevoPartida3(){
      Application.LoadLevel(nivelInicial3);
   }
   //Especifico metodo para el botón cargar menú principal
   public void nuevoPartida4(){
      Application.LoadLevel(nivelInicial4);
   }
```

```
    //Especifico método para el botón Exit
    public void exitGame(){
        Application.Quit();
    }
}
```

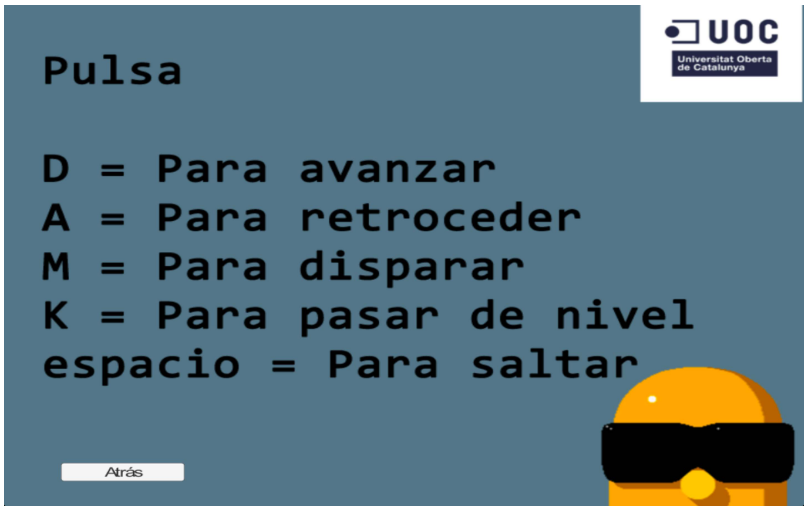
Modificamos el objeto Canvas creado con el nombre menuPricipal y en las variables públicas creadas, le añadimos con que nivel queremos que empiece cada una de las variables de nivel.



Después configuramos los botones, añadiendo una acción, esta nueva acción la debemos especificar el script de menuPrincipal y en la opción de Runtime Only especificar al función de nuevo.PartidaX defenidos en el script.



Ahora al pulsar sobre el botón créditos y ayuda se cargarán las pantallas correspondientes y nos aparecerá un botón de Atrás, para volver nuevamente al menú principal. Para crear este botón de Atrás he tenido que repetir el proceso con la variable nivelInical4.



nivel ayuda

nivel créditos

25 – Pantalla Fin del juego

Una vez lleguemos al final del videojuego, deberemos pulsar la tecla K para traspasar la última puerta y pasaremos a la pantalla final del videojuego que nos permitirá volver al menú principal.

11. Prototipos

Al realizar el diseño de un videojuego, los primeros prototipos serán Lo-Fi, prototipos de baja fidelidad, para explotar e ir mejorando todo el sistema de funcionalidad y programación sobre el videojuego. En cuanto se vaya finalizando toda la parte funcional del videojuego se podrá entregar los primeros prototipos Hi-Fi o de alta fidelidad.

11.1 Lo-Fi

· Primer prototipo con el primer nivel básico y movilidad del personaje principal.

· Segundo prototipo con el segundo nivel básico y movilidad del personaje.

· Tercer prototipo con el menú principal

11.2 Hi-Fi

· Primer prototipo con el nivel finalizado y mejora de la movilidad del personaje principal.

· Segundo prototipo con el segundo nivel finalizado y mejora de la movilidad del personaje principal.

· Tercer prototipo producto totalmente finalizado con menú principal y totalmente funcional.

12. Perfiles de usuarios

El público objetivo o target del videojuego son:

· Personas de una edad comprendida entre los 30 y 40 años.

· Con unos ingresos que pueden oscilar entre el salario mínimo interprofesional y sueldo de ejecutivos.

· Personas con un mínimo de formación, desde graduados escolar hasta carreras universitarias.

· Con unos hábitos de compra en los que se refleje un interés por la electrónica en general y especialmente los videojuegos.

· Con una profesión o ocupación en la cual que sea habitual el manejo de ordenadores.

· La situación familiar es indiferente, si es una persona con hijos les podrá enseñar a sus hijos como se entretenía en la infancia, si no tiene hijos podrá disfrutar él.

· Con una situación geográfica que lo sitúe en uno de los países del primer mundo, con el que puedan tener acceso a un ordenador para poder jugar a los videojuegos.

· Finalmente la más importante son sus aficiones, le debe gustar el sector del entrenamiento digital.

13. Usabilidad

En cuanto la usabilidad del videojuego, este se ha diseñado utilizando UX Design (User Experience Design) o “Diseño de Experiencia de Usuario”. Este método de diseño nos permite

realizar un videojuego que cubre las necesidades de los usuarios finales consiguiendo una mayor satisfacción y una mejor experiencia de uso con un menor esfuerzo.
Cada decisión del diseño debe cubrir las necesidades, expectativas, motivaciones, capacidades y objetivos de los usuarios. Por lo que se debe conocer el target al que va dirigido el videojuego e ir mejorándolo y corrigiéndolo mediante test con usuarios.

Al analizar la usabilidad de nuestro videojuego, podemos observar lo siguiente:

· Facilidad de aprendizaje: un usuario que nunca se haya encontrado con este videojuego le será muy sencillo aprender como funciona.

· Facilidad: determina la rapidez que un usuario puede ejecutar las funcionalidades básicas del videojuego.

· Satisfacción subjetiva: indica la satisfacción de los usuarios cuando están jugando al videojuego y este les parece fácil y simplicidad en el uso de sus opciones.

14. Test

Para realizar el test de nuestro videojuego nos hemos basado en realizar una evaluación heurística, que consiste en probar el videojuego siguiendo unos principios de usabilidad reconocidos. La revisión de estas pruebas se realizan de manera individual y asumiendo que son el consumidor final.

Para realizar el test se crea una plantilla que recoge información de análisis relativos a las siguientes características:

· La jugabilidad

· La dificultad

· El control del personaje

· Guía del usuario

· La información proporcionada por el juego

· Diseño visual

· La coherencia

Con el objetivo de conseguir un análisis completo, se evalúa cada uno de los apartados del 1 al 5, siendo la puntuación del 5 como la más favorable y la 1 como la menor.

Test 1:

· Individuo 1

Sexo	Mujer
Edad	37
Nivel de estudios	Licenciada en economía
Aficiones	Lectura

· Resultados

Tareas	Puntuación
Jugabilidad	3
Dificultad	4
Control del personaje	3
Guía del usuario	3
La información proporcionada por el juego	1
Diseño visual	2
La coherencia	4

· Individuo 2

Sexo	Hombre
Edad	38
Nivel de estudios	Ingeniero informático
Aficiones	MBK

· Resultados

Tareas	Puntuación
Jugabilidad	2
Dificultad	3
Control del personaje	2
Guía del usuario	3
La información proporcionada por el juego	2
Diseño visual	2
La coherencia	4

· Individuo 3

Sexo	Hombre
Edad	39
Nivel de estudios	Empresariales
Aficiones	Música

· Resultados

Tareas	Puntuación
Jugabilidad	3
Dificultad	3
Control del personaje	3
Guía del usuario	3
La información proporcionada por el juego	3
Diseño visual	3
La coherencia	3

· Individuo 4

Sexo	Hombre
Edad	44
Nivel de estudios	Ingeniero informático
Aficiones	Natación, Running

· Resultados

Tareas	Puntuación
Jugabilidad	3
Dificultad	3
Control del personaje	1
Guía del usuario	3
La información proporcionada por el juego	4
Diseño visual	3
La coherencia	4

· Individuo 5

Sexo	Mujer
Edad	28
Nivel de estudios	Licenciada derecho
Aficiones	Música, deporte

· Resultados

Tareas	Puntuación
Jugabilidad	3
Dificultad	2
Control del personaje	3
Guía del usuario	2
La información proporcionada por el juego	3
Diseño visual	3
La coherencia	4

Resultado del primer test, realizado sobre el primer prototipo del videojuego.

Tareas	Puntuación
Jugabilidad	2,8
Dificultad	3
Control del personaje	2,4
Guía del usuario	2,8
La información proporcionada por el juego	2,6
Diseño visual	2,6
La coherencia	3,8

Las puntaciones obtenidas más bajas son Jugabilidad, Control del personaje, La información proporcionada por el juego y Diseño visual. Por lo que en el desarrollo del segundo prototipo se debe mejorar estos apartados.

Test 2:

· Individuo 1

Sexo	Mujer
Edad	37
Nivel de estudios	Licenciada en economía
Aficiones	Lectura

· Resultados

Tareas	Puntuación
Jugabilidad	4
Dificultad	4
Control del personaje	4
Guía del usuario	3
La información proporcionada por el juego	3
Diseño visual	3
La coherencia	4

· Individuo 2

Sexo	Hombre
Edad	38
Nivel de estudios	Ingeniero informático
Aficiones	MBK

· Resultados

Tareas	Puntuación
Jugabilidad	4
Dificultad	3
Control del personaje	4
Guía del usuario	3
La información proporcionada por el juego	3
Diseño visual	4
La coherencia	4

· Individuo 3

Sexo	Hombre
Edad	39
Nivel de estudios	Empresariales
Aficiones	Música

· Resultados

Tareas	Puntuación
Jugabilidad	4
Dificultad	4
Control del personaje	4
Guía del usuario	3
La información proporcionada por el juego	3
Diseño visual	4
La coherencia	4

· Individuo 4

Sexo	Hombre
Edad	44
Nivel de estudios	Ingeniero informático
Aficiones	Natación, Running

· Resultados

Tareas	Puntuación
Jugabilidad	4
Dificultad	3
Control del personaje	4
Guía del usuario	3
La información proporcionada por el juego	4
Diseño visual	3
La coherencia	4

· Individuo 5

Sexo	Mujer
Edad	28
Nivel de estudios	Licenciada derecho
Aficiones	Música, deporte

· Resultados

Tareas	Puntuación
Jugabilidad	3
Dificultad	5
Control del personaje	4
Guía del usuario	3
La información proporcionada por el juego	3
Diseño visual	4
La coherencia	4

Resultado del segundo test, realizado sobre la versión Alpha del videojuego.

Tareas	Puntuación
Jugabilidad	3,8
Dificultad	3,8
Control del personaje	4
Guía del usuario	3
La información proporcionada por el juego	3,2
Diseño visual	3,6
La coherencia	4

Se ha mejorado la jugabilidad, Control del personaje y el diseño visual. Ahora las puntaciones más bajas son Guía del usuario, La información proporcionada por el juego y Diseño visual. Por lo que en el desarrollo del segunda versión (Beta) se debe mejorar estos apartados.

Test 3:

· Individuo 1

Sexo	Mujer
Edad	37
Nivel de estudios	Licenciada en economía
Aficiones	Lectura

· Resultados

Tareas	Puntuación
Jugabilidad	5
Dificultad	4
Control del personaje	4
Guía del usuario	4
La información proporcionada por el juego	4
Diseño visual	5
La coherencia	4

· Individuo 2

Sexo	Hombre
Edad	38
Nivel de estudios	Ingeniero informático
Aficiones	MBK

· Resultados

Tareas	Puntuación
Jugabilidad	4
Dificultad	5
Control del personaje	4
Guía del usuario	5
La información proporcionada por el juego	5
Diseño visual	5
La coherencia	4

· Individuo 3

Sexo	Hombre
Edad	39
Nivel de estudios	Empresariales
Aficiones	Música

· Resultados

Tareas	Puntuación
Jugabilidad	4
Dificultad	4
Control del personaje	4
Guía del usuario	5
La información proporcionada por el juego	4
Diseño visual	4
La coherencia	4

· Individuo 4

Sexo	Hombre
Edad	44
Nivel de estudios	Ingeniero informático
Aficiones	Natación, Running

· Resultados

Tareas	Puntuación
Jugabilidad	4
Dificultad	4
Control del personaje	4
Guía del usuario	5
La información proporcionada por el juego	4
Diseño visual	4
La coherencia	4

· Individuo 5

Sexo	Mujer
Edad	28
Nivel de estudios	Licenciada derecho
Aficiones	Música, deporte

· Resultados

Tareas	Puntuación
Jugabilidad	4
Dificultad	5
Control del personaje	4
Guía del usuario	4
La información proporcionada por el juego	4
Diseño visual	4
La coherencia	4

Resultado del segundo test, realizado sobre la versión Beta del videojuego.

Tareas	Puntuación
Jugabilidad	4,2
Dificultad	4,4
Control del personaje	4
Guía del usuario	4,6
La información proporcionada por el juego	4,2
Diseño visual	4,4
La coherencia	4

Se ha mejorado la Guía del usuario, la información proporcionada por el juego y el Diseño visual. Ahora ya podemos crear la versión 1.0 del videojuego.

15. Versiones de la aplicación

Versión Alpha

Versión Beta

Versión 1.0

16. Requisitos de la instalación

No se precisa ningún requisito para realizar la instalación del videojuego, este está encapsulado y se ejecuta desde sus propios ficheros de configuración. Simplemente se debe realizar doble click sobre el ejecutable del videojuego y disfrutar del videojuego.

17. Instrucciones de uso

En el videojuego se tendrá que manejar un personaje por diferentes pantallas, hacerlo saltar entre diferentes plataformas y disparar para eliminar los enemigos de cada uno de los niveles.

Así que para realizar las acciones del personaje, se utilizarán las teclas más habituales de los videojuegos.

Tecla D: avanzar

Tecla A: retroceder

Tecla Space: saltar

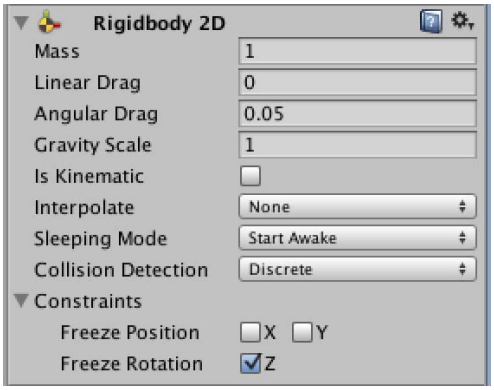
Tecla M: disparo

Tecla K: pasar de nivel

18. Bugs

1- Al desplazar el personaje principal y hacerlo saltar, este se balancea y vuelca.

Lo solucionamos modificando las propiedades del Rigibody 2D, parámetros Contraints -> Freeze Rotation Z.



2 – Al desplazar el personaje principal hacia la izquierda, no se da la vuelta.

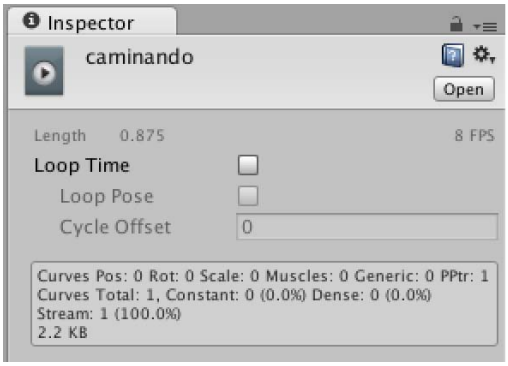
Tenemos que añadir en el script de playerController.cs el siguiente código para que realice el efecto espejo y que cuando el Player se desplace hacia la izquierda se gire el personaje.

```
if (GetComponent<Rigidbody2D>().velocity.x>0)
   transform.localScale = new Vector3(1f,1f,1f);
```

```
else if (GetComponent<Rigidbody2D>().velocity.x <0)
    transform.localScale = new Vector3(-1f,1f,1f);
```

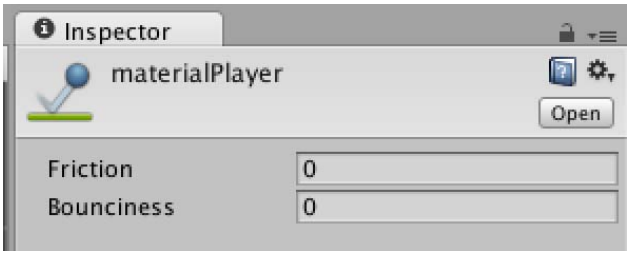
3 – Cuando dejamos de avanzar con el personaje, este sigue cominando.

En el apartado de animación, se debe quitar la opción de Loop Time para que cuando dejemos de pulsar la tecla el Player se pare y deje de caminar.



4 – El personage principal se detiene cuando choca lateralmente con las plataformas.

El Player al chocar con una plataforma se queda frenado por la fricción, para evitarlo debemos añadir un material al Player. Para crear el nuevo elemento creamos una nueva carpeta en Assets -> Materiales y creamos un nuevo material Phisics2D Material. Y configuramos el material de la siguiente manera:



5 – El Player cuando camina da sensación de que patina.

Para eliminar la sensación que el Player patine, debemos modificar el scritp de payerController, añadimos una nueva variable para controlar la velocidad del movimineto:

```
//Variable para quitar la sensación de patinar el Player
private float moveVelocity;
```

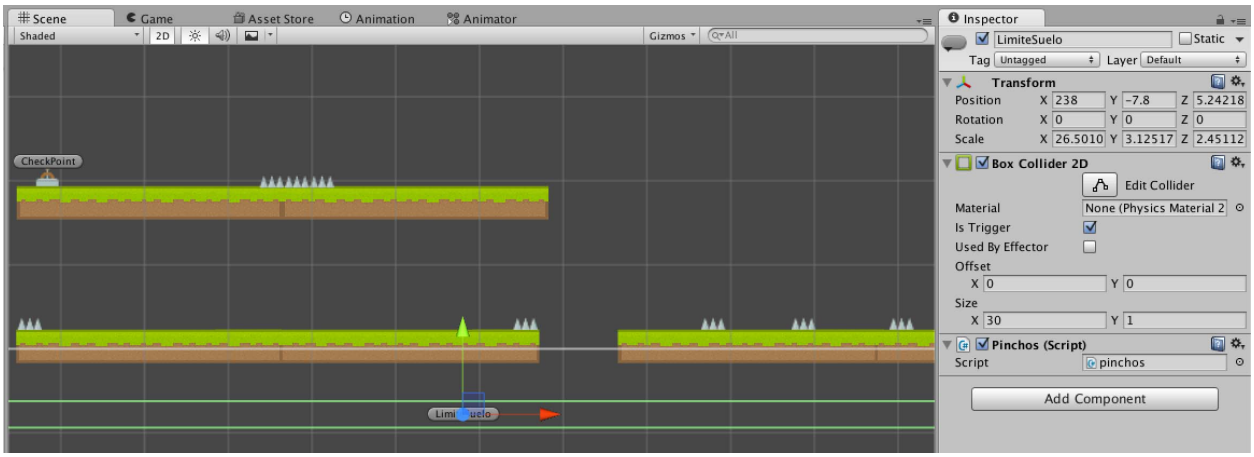
Y sustituimos la menera de hacer desplazar al personaje, sustituimos:

```
//Iniciamos la variable a 0
        moveVelocity = 0f;
        //Si pulsamos la D, desplazamos el personaje a la derecha
        if(Input.GetKey(KeyCode.D)){
            //GetComponent<Rigidbody2D>().velocity = new Vector2(velocidadMovimiento,GetCo
mponent<Rigidbody2D>().velocity.y);
            //Con este metodo conseguimos que no patine el player
            moveVelocity=velocidadMovimiento;
        }
        //Si pulsamos la A, desplazamos el personaje a la Izquierda
        if(Input.GetKey(KeyCode.A)){
            //GetComponent<Rigidbody2D>().velocity = new Vector2(-
velocidadMovimiento,GetComponent<Rigidbody2D>().velocity.y);
            //Con este metodo conseguimos que no patine el player
            moveVelocity=-velocidadMovimiento;
        }
        GetComponent<Rigidbody2D>().velocity = new Vector2 (moveVelocity, GetComponent<
Rigidbody2D> ().velocity.y);
```

6 – Cuando el player cae al infinito no fallece.

Lo primero que hago es crear un nuevo objeto, GameObject -> Create Empty.

A este objeto se le añade un BoxCollider para controlar cuando el Player lo traspase, le ponemos la opción de Trigger y le añadimos el scritp de C# de pinchos.cs a este nuevo objeto. Ahora cuando el Player se cae, este fallece.



Para optimizar el código renombro el script de pinchos.cs a muertePlayer.cs.

7 – Los enemigos no se giran.

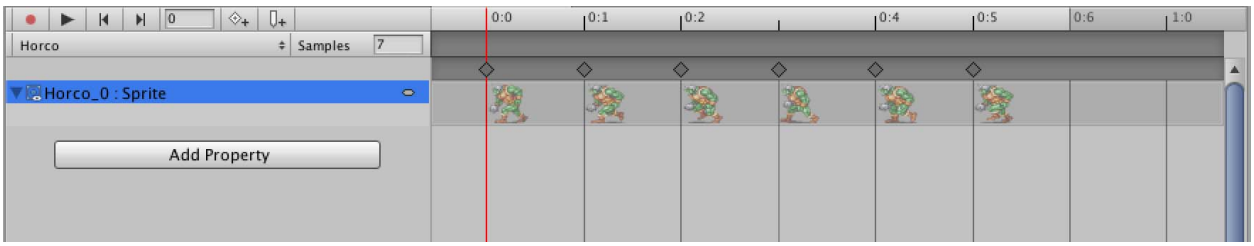
Para que el enemigo se gire debo añadir al script de enemigo.cs las siguientes lineas:

```
//Añadimos reflejo para cuando se gire el enemigo
    transform.localScale = new Vector3(-10f,10f,10f);
```

Y cuando se mueve a la izquierda

```
//Añadimos reflejo para cuando se gire el enemigo
    transform.localScale = new Vector3(10f,10f,10f);
```

Finalmente selecionamos al Orco y vamos al menú de animación, creamos una animación por Interpolación Básica y conseguimos el enemigo animado.

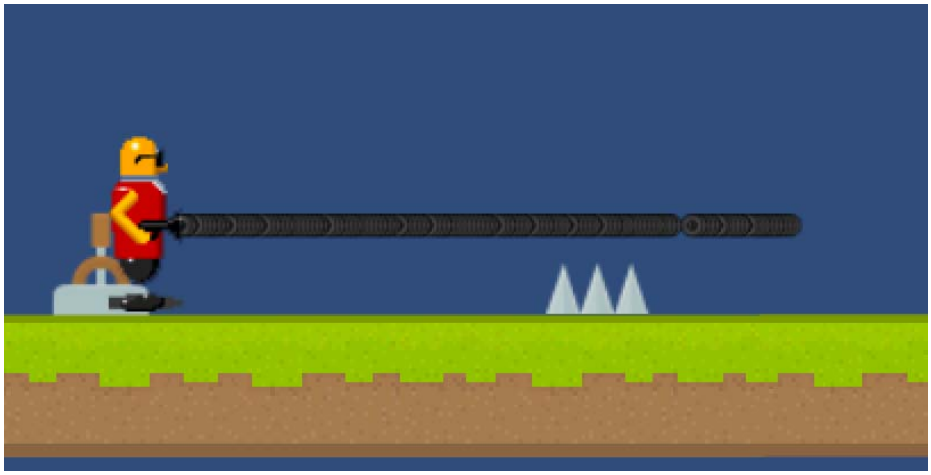


8 – Los enemigos no matan.

Para que los enemigos maten al Player, simplemente tengo que añadir el script de muertePlayer.cs al enemigo. Ahora cuando el Player choque con el enemigo este fallecerá.

9 – Evitar disparo continuo.

En la configuración actual se puede disparar continuamente manteniendo la tecla M pulsada y esto no debe ser así, ya que facilitaría demasiado poder eliminar los enemigos, se debe espaciar el tiempo del disparo.

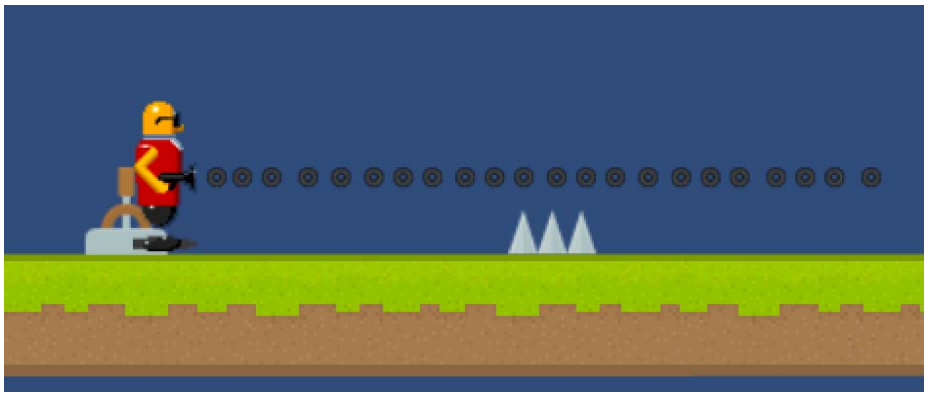


Para espaciar el tiempo de disparo debo modificar el script de playerController.cs, de:

```
if (Input.GetKey(KeyCode.M)) {
Instantiate(bala,focoDisparo.position,focoDisparo.rotation);
}
```

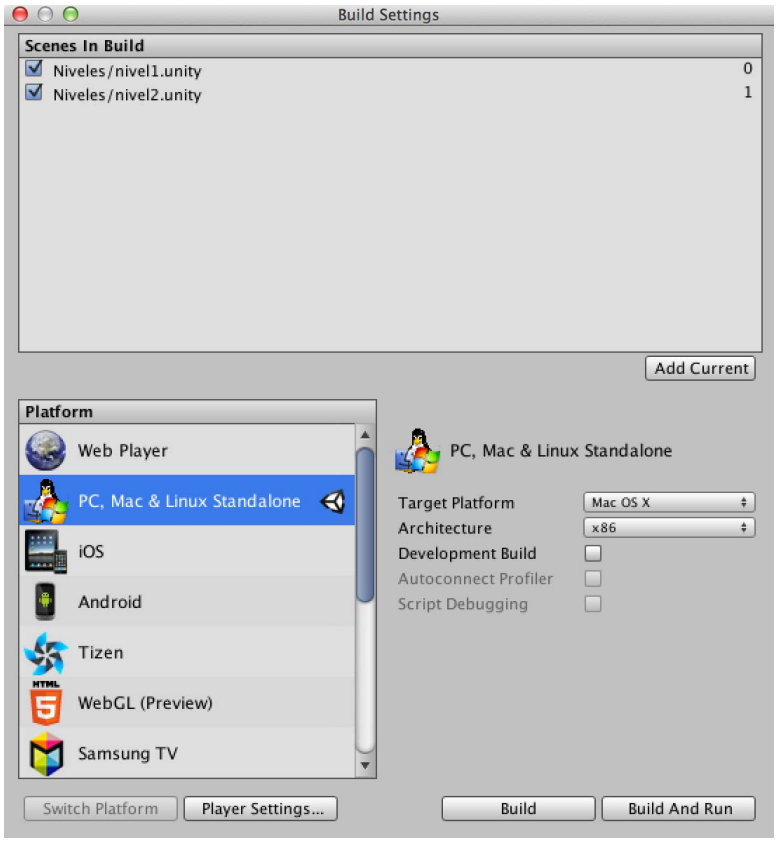
a

```
if (Input.GetKeyDown(KeyCode.M)) {
Instantiate(bala,focoDisparo.position,focoDisparo.rotation);
}
```



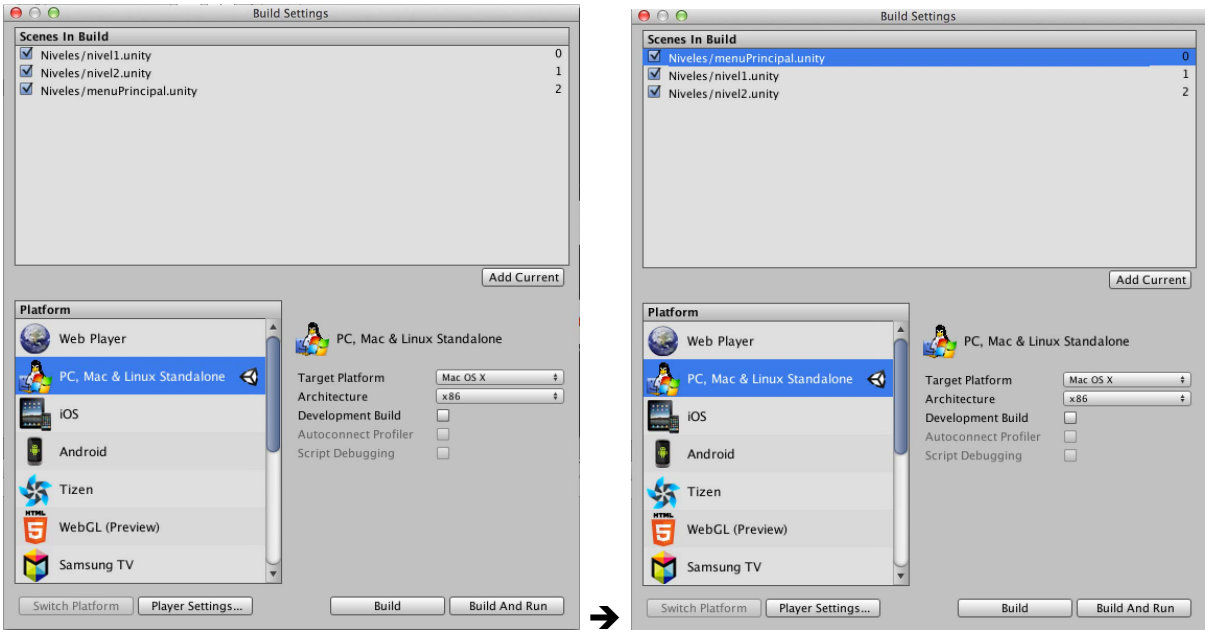
10 – Pasar de nivel.

Después de crear el gameobjet de la puerta y crear el script, este no funcionaba. Se ha solucionado añadiendo el nivel 2 en el proyecto. Para solucionarlo debemos ir al menú File -> Buind Settings y añadir el nivel 2 como parte del proyecto.



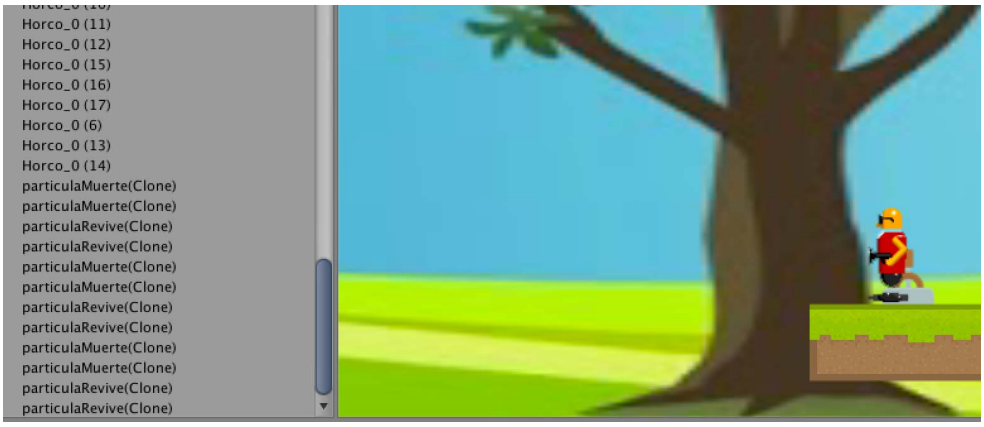
11 – No carga correctamente el menú principal.

Al ejecutar el videojuego nos aparece en el primer nivel y no en el menu principal. Para resolverlo se debe ir a File -> Building Settings y ordenar correctamente el orden de los niveles.



12 – Al generar los sistemas de particulas estas no se eliminan.

Al generar las partículas cuando el Player fallece o reaparece, estas se van generando en el videojuego aumentando los objetos del videojuego y aumentando su carga de proceso.



Para eliminar estas partículas creamos un nuevo scritp en C#, eliminarParticulas.cs.

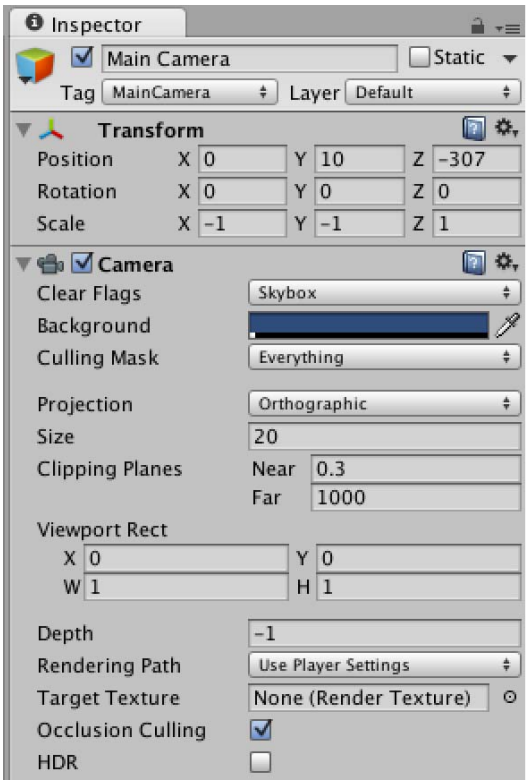
eliminarParticulas.cs.

```
using UnityEngine;
using System.Collections;
public class eliminarParticulas : MonoBehaviour {
//Creamos variable para el sistema de particulas
   private ParticleSystem sistemaDeParticula;
// Use this for initialization
   void Start () {
      sistemaDeParticula = GetComponent<ParticleSystem>();
   }
   // Update is called once per frame
   void Update () {
//Destruimos las particulas
      if (sistemaDeParticula.isPlaying){
         return;
      }
      Destroy(gameObject);

}

}
```

Una vez generado el script, lo debemos asignar a los prefabs de los dos sistemas de partículas creados (particulaMuerte, particulaRevive). Ahora cuando el Player fallece y revive, se crean las partículas y luego se eliminan automáticamente consiguiendo así optimizar el videojuego.

13 – Al generar el videojuego, la cámara pierde su configuración.

Al generar el videojuego hemos observado que la cámara se comporta de manera diferente que en el entorno de pruebas, así que no nos queda otra opción que ir modificando las opciones de la cámara y crear el videojuego hasta conseguir la visualización deseada para el videojuego.



19. Proyección a futuro

En le futuro se distribuirá el videojuego en diferentes plataformas digitales, como en los smartphones y tablets mediante los diferentes stores de cada uno de los sistemas operativos de estos dispositivos.

Google Play y App Store

Teniendo que adaptar la resolución de la pantalla y modificar los controles del personaje a estos nuevos dispositivos.

20. Presupuesto

Tareas	JP	DC	DG	AP	P	TS	MK	UT	Duración
Desarrollo del videojuego 2D									**99**
Lanzamiento del proyecto									**6**
Organización de los equipos	R								1
Definición del alcance	R	P							2
Planificación detallada del proyecto	R	P	P	P		C	C		3
Diseño del videojuego 2D									**45**
Requerimientos, diseño creativo y gráfico									**16**
Diseño de estructura y navegación	P	R	P	P					2
Diseño de la jugabilidad del videojuego	I	R	C	P					2
Diseño gráfico (prototipo)	I	P	R				I		6
Diseño metodología C#	I			R	P				4
Diseño sonido	I	I		I		R			2
Diseño técnico									**10**
Diseño técnico arquitectura del videojuego	I	R	R	R	I	I	I		2
Diseño de los contenidos	I	R	R	I		P	I		5
Diseño modelo de datos	I			R	P				1
Diseño técnico aprobado	R	I	I	I		I			2
Prototipo									**19**
Desarrollo del prototipo	I	R	R	R	P	P			15
Test técnico del prototipo	I	I	P	R	P	P			3
Validación del prototipo	R	P	P	P		P			1
Producción del videojuego									**32**
Desarrollo del videojuego	I	P	P	R	P				20
Desarrollo de los scripts C#	I	I		R	P				4
Presentación del videojuego	P						R		1
Plan de pruebas	I	R		I					1
Realización pruebas con los técnicos y correcciones	I	R	P	P	P	P			3
Realización pruebas con usuarios y correcciones	I	R						P	2
Aplicación aprobada	R	I	I	I			I		1
Difusión del videojuego									**6**
Publicación del videojuego en el buscador de Google	C						R		2
Publicación a través del portal estándar	I	P	P	P	P		R		2
Publicación en redes sociales	I						R		2
Arranque de la aplicación	R	I	I	I			P		**1**
Apoyo post-arranque	I	R	P	P					**5**
Definición futuros cambios	P	R	C	C		C	C		**3**
Cierre del proyecto	R	I	I	I		I	I		**1**

Perfiles profesionales

JP: Jefe de proyecto

DC: Diseñador creativo
DG: Diseñador gráfico
AP: Analista programador
P: Programador
TS: Técnico de sonido
MK: Marqueting
UT: Usuario tester

Implicación

R= Responsable
P= Participa
C= Se consulta
I= Se informa

Después de realizar una estimación del tiempo del proyecto sabemos que para realizar el proyecto son un total de 99 jornadas, cada perfil profesional tiene un coste por jornada, así que el coste sería la suma de todas las jornadas por cada uno de los perfiles.

Perfil	Coste por jornada	Jornadas	Total
JP: Jefe de proyecto	520	15	7800
DC: Diseñador creativo	420	40	16800
DG: Diseñador gráfico	420	28	11760
AP: Analista programador	450	49	22050
P: Programador	360	52	18720
TS: Técnico de sonido	360	4	1440
MK: Marqueting	360	7	2520
UT: Usuario tester	200	2	400
		Total:	81490

El coste total del proyecto sería de 81.490 €.

21. Análisis del mercado

Desde la introducción a la era digital, donde se ha desarrollado la computación, la industria de los videojuegos ha experimentado unas tasas de crecimiento. En la década 2000 los videojuegos han generado más dinero que el cine y la música juntos, como por ejemplo en España. En el año 2009 la industria de los videojuegos ha generado un total de 57.600 millones de euros en todo el mundo.

En el año 2012 la industria de los videojuegos facturó un 16% menos que en años anteriores, siendo un total de 822 millones de euros, retrocediendo a la facturación que se realizaba en el año 2005. Este descenso se produjo principalmente por la crisis a nivel mundial y por el principal descenso de ventas de videoconsolas, estas se vieron alteradas por situarse en un momento de transición. No obstante el sector de los videojuegos sigue siendo la primera industria de ocio audiovisual en España.

Entre los años 2011-2013, quince compañías de videojuegos que salieron a bolsa eran chinas, japonesas y surcoreanas, por lo que está apareciendo un nuevo empuje en el sector de los videojuegos, esto se traduce en unos beneficios de 5.600 millones de dólares en el 2013.

Para el año 2017 se prevé llegar a facturar en la industria de videojuegos un total de 100.000 millones de dólares, debido al incremento de videojuegos online y videojuegos para Smartphone, que podrían alcanzar una cuota del 60% de todos los videojuegos y se prevé un crecimiento continuo del 26% anualmente.

22. Viabilidad

El ocio digital está de moda, no hay hogar que no tenga un ordenador de sobre mesa o un portátil. Cada vez es más presente tener también tablets y smartphones.

El videojuego está desarrollado para ejecutarse en ordenador, pero con licencias de pago se podría crear la versión para los smartphones, tablets, web online, etc. Simplemente suscribiéndose a la herramienta profesional y realizar los cambios oportunos para adaptar el videojuego a estos nuevos dispositivos.

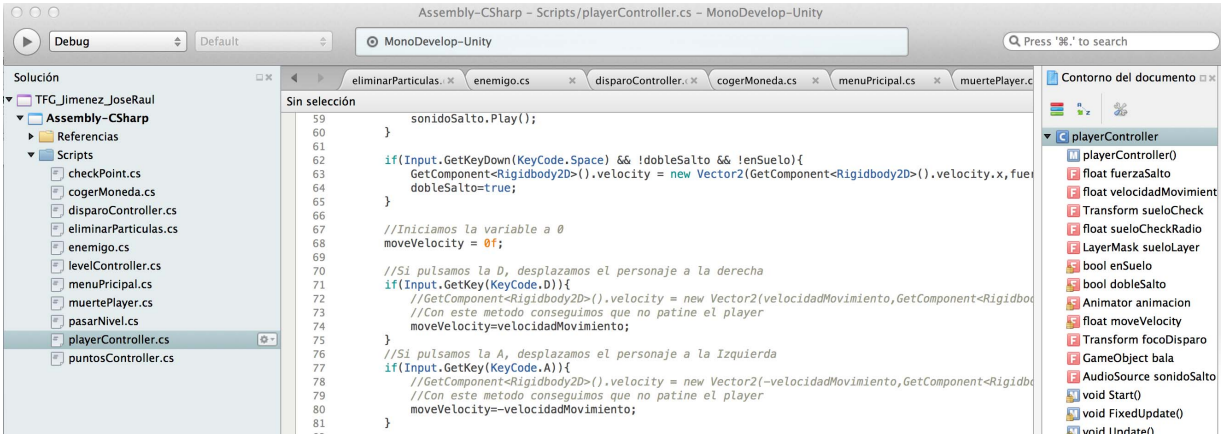
Por lo que viendo el crecimiento del consumo digital en smartphones, tablets y juegos online es una apuesta segura realizar un buen videojuego online.

23. Conclusión

El trabajo realizado ha sido un videojuego en 2D de plataformas, donde debemos manejar un personaje con vista en tercera persona. Deberemos hacer caminar, desplazar el personaje horizontalmente hacia la derecha, al player, saltar sobre una seria de plataformas, evitar trampas, eliminar enemigos y recoger monedas para poder completar el videojuego con el mayor número de puntos posible. Este género de videojuegos fue muy popular en la década de los 80 y en la actualidad son conocidos como videojuegos retro.

Para realizar el proyecto he utilizado Unity 3D, un engine que nos ha permitido realizar el videojuego en 2D. Este engine ofrece un entorno de desarrollo visual dividido en una pantalla principal, pestañas en la parte inferior y laterales.

Muchos de los elementos del videojuego se han podido realizar visualmente, como por ejemplo crear los niveles, donde fácilmente hemos podido arrastrar los gráficos deseados al contenido del videojuego. No obstante se ha destinado un gran esfuerzo a la programación, donde se ha tenido que aprender el lenguaje C Sharp (C#) debido a que la mayor parte de la interactividad del videojuego se realiza mediante la misma, siendo esta la única opción, mediante la herramienta de desarrollo que ofrece Unity 3D, MonoDevelop.



En referencia al diseño visual, se ha utilizado Adobe Fireworks y Adobe Photoshop para editar alguno de los diseños de los personajes del videojuego.

El resultado final ha sido un videojuego 2D que evoca a los videojuegos de los años 80 a los que tanto jugué en mi infancia.

Bibliografía

· Unity 3D
(http://unity3d.com/es/)

· Unity Manual
(http://docs.unity3d.com/Manual/index.html)

· Asset Store
(https://www.assetstore.unity3d.com/en/)

· Arcade
(https://es.wikipedia.org/wiki/Arcade)

· Industria de los videojuegos
(https://es.wikipedia.org/wiki/Industria_de_los_videojuegos)

· 001 Engine
(http://www.engine001.com)

· 3D Game Marker
(http://www.thegamecreators.com)

· Aleph One
(https://alephone.lhowon.org)

· Axiom
(http://axiom3d.net)

· G4 Engine
(http://www.terathon.com)

· Euphoria
(http://www.naturalmotion.com)

· Unity 3D
(http://unity3d.com)

· Gantt Project
(http://www.ganttproject.biz/)

· Sprite Database
(http://spritedatabase.net)

· Bfxr
(http://www.bfxr.net/)

· Font space
(http://www.fontspace.com/)

· La industria del videojuego valdrá más de 100.000 millones de dólares en 2017
(http://www.abc.es/tecnologia/videojuegos/20140116/abci-industria-videojuegos-milones-euros-201401161323.html%20La%20industria%20del%20videojuego%20valdr%C3%A1%20m%C3%A1s%20de%20100.000%20millones%20de%20d%C3%B3lares%20en%202017)

· Juguetes y videojuegos: ¿una moda pasajera o el futuro de la industria?
(http://www.expansion.com/empresas/2015/05/29/5568b49b46163f1d288b459b.html)

· La industria del videojuego facturó un 16% menos en 2012
(http://www.abc.es/tecnologia/videojuegos/20130312/abci-industria-videojuego-facturo-menos-201303121448.html)