UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA - CAMPUS D'ALCOI

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE GRADO

JUEGO DE PLATAFORMAS BASADO EN UNITY 2D

MEMORIA PRESENTADA POR:

Carlos Maiato Llorens

TUTOR:

Prof. Jordi Joan Linares Pellicer

Convocatoria de defensa: Septiembre del 2017

Resumen

Este proyecto consiste en la creación de un videojuego 2D del tipo plataformas endlessrun, bajo el motor de desarrollo Unity y usando el lenguaje de programación orientado a objetos C#. El juego estará pensado para su disfrute en los SO Windows, MacOS, iOS X, Android y Linux y en los dispositivos como PC, móviles y tablets. También se busca aportar una visión general a la hora de desarrollar un videojuego de este estilo paso a paso, mostrando sus distintas etapas en cuanto a su desarrollo.

Abstract

This project consists of the creation of a 2D platform endlessrun videogame, under the Unity game engine and using the C # object-oriented programming language. The game will be designed for your enjoyment in Windows OS, MacOS, iOS X, Android and Linux and on devices such as PC, mobile and tablets. It also seeks to provide an overview when developing a video game of this style step by step, showing its different stages in terms of its development.

Palabras clave: Videojuego, Unity 2D, plataformas, C#, dispositivos móviles, PC, MacOSX.

Keywords: Videogame, Unity 2D, platforms, C#, mobile devices, PC, MacOSX.

Tabla de contenido

1 Introducción

1.1 Motivación

Los videojuegos llevan en nuestra sociedad desde hace décadas, jugados tanto por jóvenes como por adultos. Pese a que los videojuegos siempre han estado relacionados por la sociedad como una forma de ocio enfocada a un público joven, esto está cambiando en los últimos años. Ya sea en PC, videoconsolas, dispositivos móviles o consolas portátiles la industria de los videojuegos se ha convertido en una gran potencia del entretenimiento llegando a superar a algunas tan importantes e históricas como el cine, música y televisión.

Una de las principales razones por las que la industria de los videojuegos se ha colocado en el podio del entretenimiento se debe en gran parte a herramientas como Unity, que posibilitan la creación de videojuegos con muy pocos recursos, estos son conocidos como juegos independientes. La gran mayoría de estos juegos independientes aparecen en PC y plataformas móviles por la cantidad de herramientas disponibles tanto para su creación, como la facilidad para su distribución, ya que en estas plataformas predomina el contenido digital mientras que en consolas sigue la costumbre del formato físico. Los juegos Ori y LIMBO son un ejemplo de juegos independientes que han triunfado y se han convertido en juegos de referencia con pocos recursos.

Ori

LIMBO

Además de mi gran afición por los videojuegos, con la realización de este proyecto también pretendo reforzar mis conocimientos adquiridos en la asignatura de programación de videojuegos, en la que fui alumno durante el 3 curso del grado. Mejorar mi dominio en el lenguaje de programación C# y en el uso de la herramienta de desarrollo Unity.

El estilo de videojuegos que busca la gente va cambiando con los años y con el también cambia el mercado, por ejemplo, la realidad virtual en la actualidad está pegando muy fuerte en la industria. En este proyecto he optado por hacer un juego de plataformas, modalidad que lleva ya décadas existiendo pero que no pasa de moda, y así abarcar a gran parte de público potencial que pueda jugarlo.

1.2 Objetivos

El objetivo principal del trabajo de fin de grado es el desarrollo de un videojuego 2D bajo la herramienta de desarrollo Unity 2D, construir la estructura del videojuego, sus componentes y características que permitan la creación de múltiples niveles diferentes cada vez que iniciemos una nueva partida. Para cumplir con este objetivo primero debemos alcanzar una serie de sub-objetivos:

- **Diseño del juego**: Objetos a recoger, contador de puntos, escenario dinámico, personaje principal.
- **Interfaz**: diseño y creación de una interfaz de juego sencilla y atractiva.
- **Funcionalidades**: programación de las funcionalidades del videojuego.
- **Sonido**: creación de los sonidos necesarios para el videojuego.
- **Creación del nivel**: el nivel se irá generando de forma aleatoria según avancemos y debe ser un reto para el jugador.

Deseamos profundizar en las características y posibilidades que nos ofrece el motor de Unity en el desarrollo de videojuegos. Se desea aprender sobre el proceso de creación de un juego, desde su creación hasta su finalización.

Aunque el juego será posible jugarlo en una gran variedad de dispositivos, el estilo de juego que vamos a realizar y su interfaz están más enfocados a su disfrute en dispositivos móviles y tablets.

El juego va a ser desarrollado en una plataforma de escritorio para a continuación exportarlo a la plataforma móvil. El control del personaje en los dispositivos de sobremesa será con el ratón mientras que en los dispositivos móviles se realizará de forma táctil mediante la pantalla.

1.3 Trasfondo

Apple man es un juego del tipo plataformas Endlessrun. Se tratan de juegos en el que existe un único nivel y en el que no controlamos el movimiento del personaje, sino que este corre de forma autónoma hacía delante, teniendo que controlar únicamente acciones como saltar. Nuestro objetivo como Apple man es ir recogiendo manzanas, las cuales nos proporcionan puntos, sin caernos de los bloques e intentando alcanzar el máximo número de puntos.

1.4 Alcance

El juego va enfocado a cualquier tipo de público y edad, basta con tener un dispositivo móvil o PC con Android, Windows, Linux o IOS X. La dificultad del juego es apta para cualquier persona con una mínima experiencia en videojuegos.

Apple Man no es el típico juego en el que pasamos horas y horas, es un juego en el que podemos aprovechar cualquier descanso o rato para jugar un par de partidas, por ejemplo, mientras esperamos el tren.

2 Herramientas utilizadas

2.1 Unity

Unity es el motor gráfico utilizado en la creación del TFG, nos proporciona una plataforma de desarrollo potente y flexible para la creación de juegos 2D y 3D y multiplataforma.

El editor de Unity nos permite exportar nuestros juegos a múltiples plataformas como:

- **Videoconsolas**: PS Vita, PS3, PS4, Xbox One, Xbox 360, WII U, Nintendo 3DS.
- **Plataformas de realidad virtual**: Steam VR, PlayStation VR, Gear VR, Oculus Rift.
- **Smart TV**: Android TV, Samung Smart TV, TV OS.
- **Plataformas de PC**: Windows, Linux/SteamOS, Mac.
- **Plataformas móviles**: iOS, Android, Windows Phone.

2.2 ¿Por qué Unity?

He decidido hacer uso de la plataforma de desarrollo Unity principalmente por el gran potencial a la hora de crear videojuegos y por ser multiplataforma, tanto para consolas, PC y dispositivos móviles, permitiéndonos desarrollar y exportar nuestros juegos a gran variedad de plataformas de forma realmente sencilla.

Esto, sumado con que Unity tiene una versión gratuita, que es la que hemos utilizado y que nos permite el desarrollo de contenido de gran calidad. Además de una gran comunidad de desarrolladores y mucha documentación extensa, tanto en libros físicos como en internet, hizo que Unity se convirtiese en nuestra opción número uno.

2.3 Lenguaje de programación C#

Unity nos permite trabajar con dos lenguajes de programación, Javascript y C#. Se ha elegido C# en lugar de Javascript por la razón de que la comunidad de usuarios de Unity que utilizan el lenguaje C# es mayor y es un plus a la hora de encontrar respuestas a posibles problemas.

A parte C# es un lenguaje orientado a objetos muy parecido a otros lenguajes vistos durante la carrera, como Java. Lo que nos facilitará la comprensión y el aprendizaje.

2.4 Monodevelop

Es un entorno de desarrollo integrado, libre y gratuito y diseñado principalmente para lenguajes como C#. Monodevelop es un IDE multiplataforma que cuenta con soporte para GNU/Linux, Windows y Mac.

2.5 Google Drive

Es una plataforma de almacenamiento en la nube y lugar donde hemos almacenado una de las copias de seguridad que hemos realizado del TFG.

2.6 UnityRemote

Es una aplicación Android que nos permite ejecutar el editor de Unity desde un dispositivo móvil, y así poder ver el funcionamiento de nuestro Videojuego sin necesidad de ir exportándolo e instalándolo cada vez que hagamos un cambio en el juego y deseemos ver el resultado.

3 Desarrollo y diseño del videojuego

3.1 Idea principal

Apple Man es un juego de plataformas 2D Endlessrunner en el que vamos avanzando por un mismo nivel, intentando conseguir un número máximo de puntos sin caernos de los bloques.

El jugador se pone en la piel de Apple Man, cuyo deseo es ser la persona que ha llegado más lejos en la zona más peligrosa de su mundo. Para demostrarlo, deberá ir recogiendo todas las manzanas que pueda. Sí consigue llegar lejos en este nivel y ser el que más manzanas ha recogido, se convertirá en una leyenda de su mundo. Apple Man corre hacía delante de forma automática y nosotros únicamente debemos de preocuparnos de controlar el salto para que no caiga al vacío, además de ir intentado recolectar el máximo número de manzanas ya que estas nos proporcionan una gran cantidad de puntos.

Para no caernos al vacío, nos encontraremos a largo del nivel unos bloques suspendidos en el aire y divididos en tres niveles de altura. Existen 3 tipos de bloques, pequeños, medianos y grandes y nuestro objetivo será ir saltando entre estos bloques, los cuales estarán estratégicamente colocados.

Las manzanas nos proporcionan 5 puntos, mientras que el ir saltando de bloque en bloque nos proporciona un solo punto, al caernos al vacío si hemos logrado superar el mayor record de puntos conseguidos por alguien ¡nos convertiremos en leyenda!.

3.2 El jugador

Nuestro personaje Apple Man está dotado de dos movimientos, que nos permitirán su explotar su jugabilidad en todo el juego.

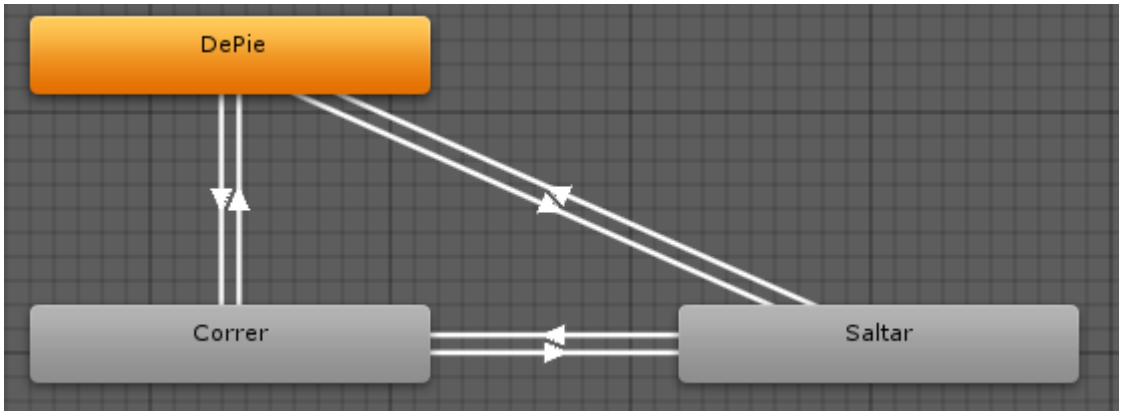
- **Pulsación de pantalla:** para iniciar la partida y nuestro personaje empiece a correr, solo tenemos que hacer una pulsación en la pantalla del dispositivo.

- **Pulsación de pantalla**: una vez nuestro personaje esté corriendo, podremos hacer saltar a nuestro personaje Apple Man hacía nuestros objetivos en el juego, manzanas y bloques.

- **Doble pulsación de pantalla**: Nos permite hacer doble salto, realizando dos pulsaciones seguidas en la pantalla del dispositivo podremos alcanzar aquellas manzanas y bloques que se encuentren a mayor distancia.

Como en la mayoría de juegos del estilo endlessrun, nuestro personaje correrá por el mapa de forma automática, nosotros solo deberemos preocuparnos de saltar entre bloques intentando coger las máximas manzanas que podamos sin caernos al precipicio e intentar superar nuestro record de puntos.

A continuación mostramos un diagrama del *Animator* donde configuraremos las acciones y movimientos que caracterizan el comportamiento de Apple Man en el juego.



El *animator* consta de dos variables y el cambio del valor de estas realizará una transición que determinará el estado de nuestro personaje.

- **VelocidadX**: es nuestro medidor de velocidad.
- **EnSuelo**: nos indica si nuestro personaje está tocando suelo o no.

Estas variables según su valor, harán que nuestro personaje cambie de animación.

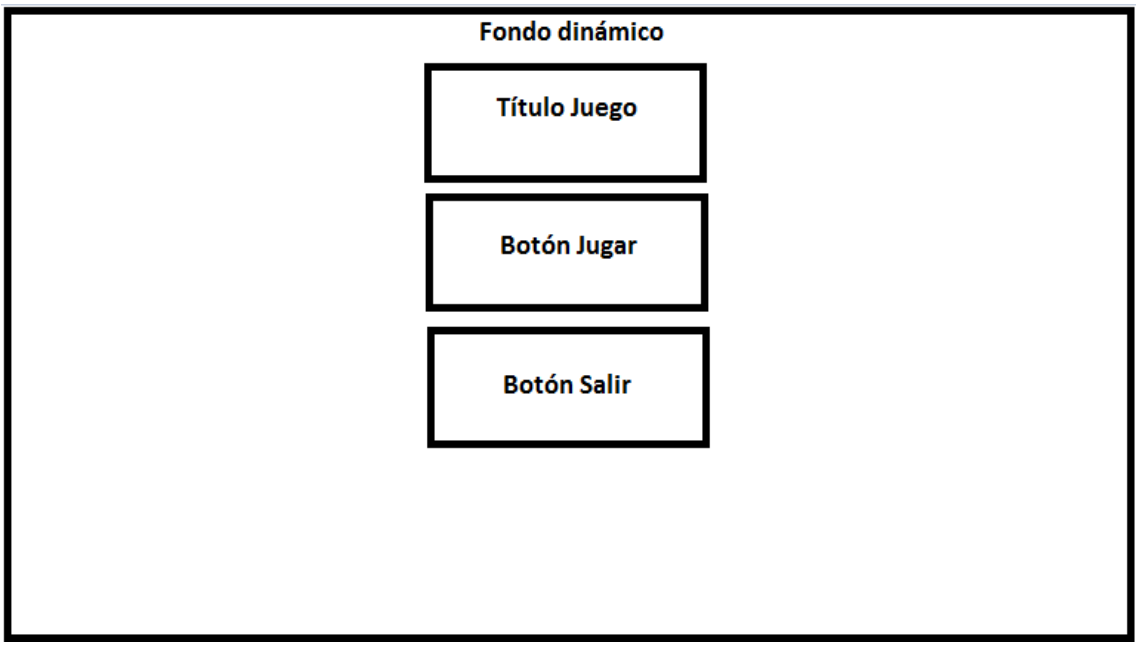
- **DePie**: Animación que nos indica que nuestro personaje está quieto, es la animación que tendremos al inicio de cada partida.

- **Correr**: Animación de correr, el personaje mueve las piernas y en general todo el cuerpo simulando la acción de correr.

- **Saltar**: Nuestro personaje está saltando**.**

3.3 Menú de inicio

Cuando iniciemos una nueva partida se nos cargará la escena de menú de inicio. En esta escena encontramos el título del juego, un fondo dinámico y dos botones con los que podemos interactuar.



- **Botón Jugar**: Iniciará una nueva partida, este botón al ser pulsado realizará cambio de escena trasladándonos a la de partida iniciada.

- **Botón Salir**: Al ser pulsado nos cerrará la aplicación del juego y dejará de ejecutarse.

3.4 Vista del juego en funcionamiento.

Si elegimos la opción de jugar nos aparecerá la vista del juego, esta vista está formada por diferentes elementos que aparecerán durante toda la partida.

1. **Contador de puntos**: nos indicará la cantidad de puntos que llevamos durante esa partida sin morir.

2. **El personaje**: figura que controlamos y con el que nos movemos por el juego.

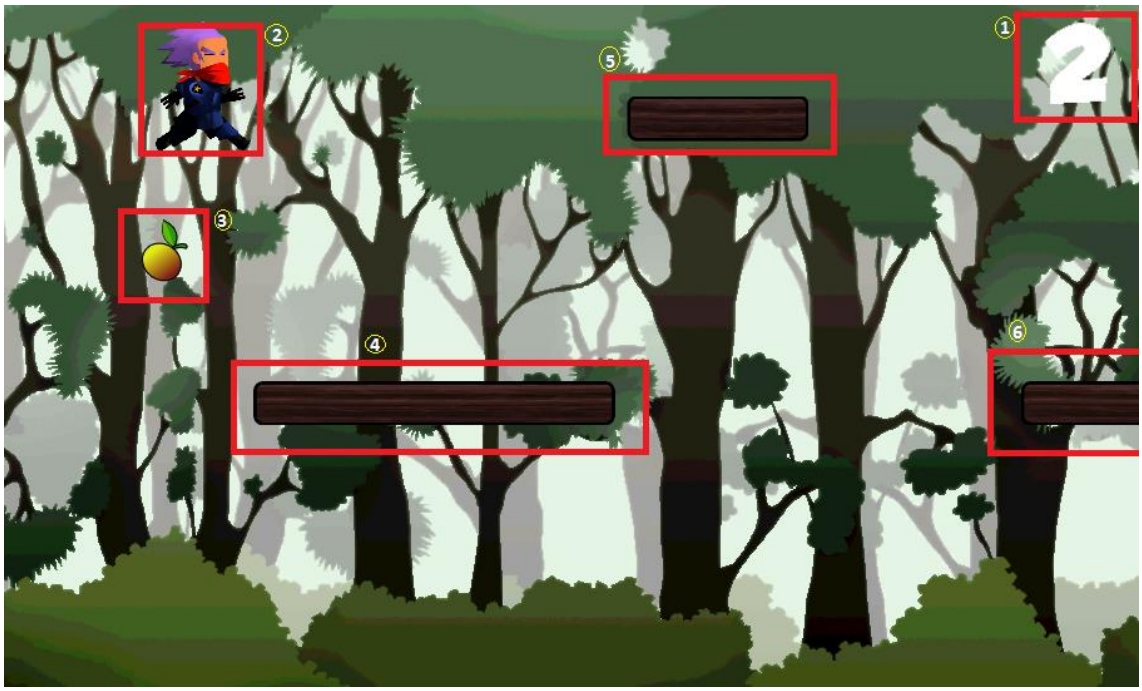
3. **Las manzanas**: son objetos que nos proporcionan puntos, un total de 5 puntos serán añadidos a nuestro contador por cada manzana recogida.

4. **Bloque largo**: objeto que nos permite no caer al vacío.

5. **Bloque corto**: objeto que nos permite no caer al vacío.

6. **Bloque mediano**: objeto que nos permite no caer al vacío.

A continuación podemos observar una imagen con los distintos elementos que aparecen en la vista de juego en funcionamiento:

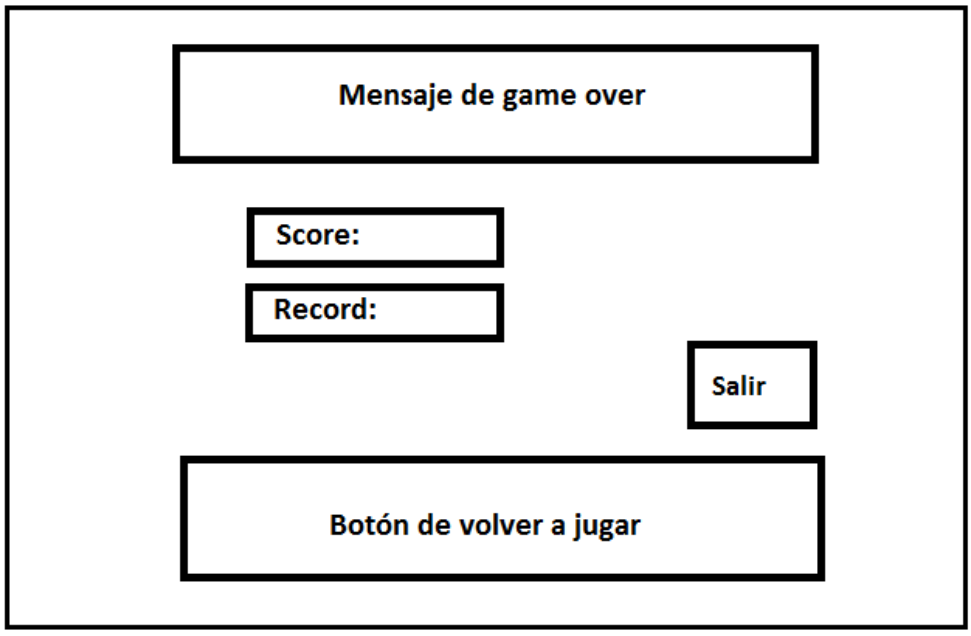


3.5 Menú de game over

Cuando el jugador caiga por el precipicio nos aparecerá el menú de game over, este menú consta de:

- **Mensaje de game over**: como dice el título, nos aparece un mensaje indicando que hemos perdido.
- **Score**: nos mostrará la puntación que hemos conseguido en esta partida.
- **Record**: nos muestra nuestra puntuación máxima que hemos logrado durante todas las partidas que hayamos jugado.
- **Salir**: salir de la aplicación y terminar su ejecución.
- **Botón de volver a jugar**: Nos devuelve a la escena de inicio de partida, para volver a realizar otro intento.

Imagen del menú de game over:



3.6 Nivel del juego

Existe un único nivel de juego, este nivel será infinito mientras consigamos no caernos de los bloques.

El nivel se va generando de forma aleatoria mientras vayamos avanzando y los elementos que se irán generando son los siguientes:

- **Bloques pequeños**: objeto de menor tamaño que nos permite no caer al vacío.
- **Bloques medianos**: objeto de tamaño mediano que nos permite no caer al vacío.
- **Bloques largos**: objeto de mayor tamaño que nos permite no caer al vacío.
- **Manzanas**: son objetos que nos proporcionan puntos, un total de 5 puntos serán añadidos a nuestro contador por cada manzana recogida.

4 Planificación

Se ha calculado que el proyecto será completamente funcional en periodo de tiempo de 4 meses. Este periodo de 4 meses podría reducirse a 2 o 3, pero el estar asistiendo a clases y la preparación de exámenes durante el curso y finales hace que pueda dedicarle únicamente el tiempo libre, sobretodo fines de semana, a la realización de este.

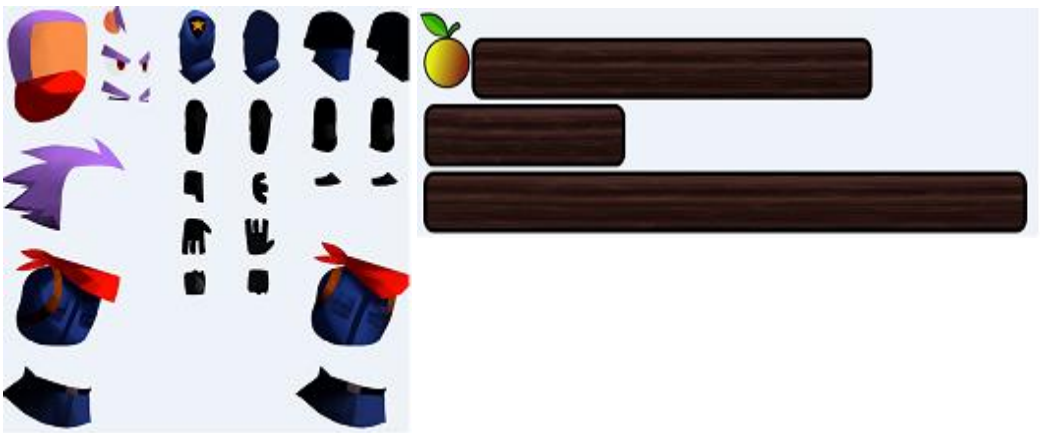
4.1 Creación del personaje

Lo primero que hicimos, antes incluso de crear el proyecto, fue buscar en internet recursos gratuitos de sprites y modelos de personajes hasta encontrar uno que nos gustase y fuese adecuado para el juego.

La duración para la realización de esta parte ha sido de 10 días, del 6 de marzo del 2017 al día 20 del mismo mes.

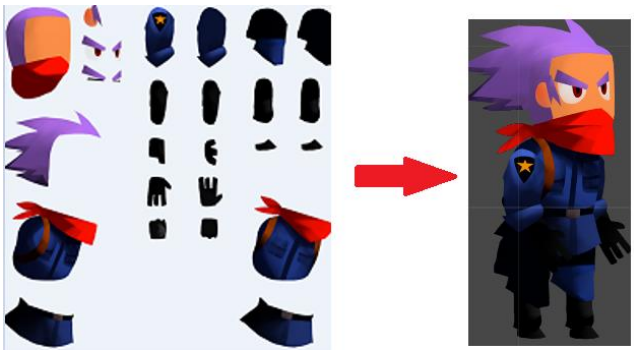
En este apartado de la planificación hemos realizado varias subtareas:

- **Búsqueda de sprites en internet:** tanto del modelo del personaje como de los componentes que aparecen en el juego e importarlos al proyecto y asignar centros de rotación a estos componentes. Estos centros de rotación son los puntos donde se mueven las articulaciones y dotaran de mayor realismo a las animaciones.
- En la siguiente imagen podemos observar los sprites que hemos utilizado:



- **Creación del protagonista:** Ya tenemos los sprites importados en el proyecto, el siguiente paso que hemos realizado ha sido la creación del personaje, este viene por piezas y por tanto tendremos que ir juntando manualmente las partes que lo conforman hasta que tenga la figura deseada.

A continuación se ofrece una imagen que muestra cómo queda nuestro personaje al juntar las piezas del sprite.



- **Creación de los prefabs:** Lo siguiente ha sido crear los prefabs del juego. los prefabs actuaran como una plantilla a partir de la cual se pueden crear nuevas instancias del objeto en la escena sin tener que crearlo de nuevo.
  Hemos creado un total de 5 prefabs, el del personaje, ítem, bloque largo, bloque mediano y bloque corto.

- **Elaboración de las animaciones:** Las animaciones son clips que proporcionan mayor realismo al juego modificando la posición y/o rotación de los objetos.
  Se han elaborado un total de 4 animaciones, 3 para el personaje, correr, saltar y estar de pie y otra para la manzana.

4.2 Diagrama de estados.

El diagrama de estado define mediante el uso de variables, el cambio de animaciones durante el juego.

Para ello hemos creado dos variables, ***VelocidadX***, que servirán para saber si nuestro personaje está corriendo o no. Y ***enSuelo*****,** con esta variable sabremos si nuestro personaje está saltando o está en el suelo.

Hemos realizado un total de 5 transiciones:

1. **Pasar de estar de pie a correr**: Para esta transición nos guiaremos mediante la variable *VelocidadX*, esta variable la hemos definido de serie con el valor 0.0.

   Si nuestro personaje está quieto, es decir, con una velocidad igual a 0.0 y el valor de la velocidad cambia por un valor mayor que 0.01 nuestro personaje empezará a correr.

2. **Pasar de correr a saltar**: esta transición será controlada por una variable, la que nos indica si nuestro personaje está tocando suelo o saltando. Por defecto la variable *enSuelo* está indicada como verdadera, lo que quiere decir que nuestro personaje está tocando el suelo. La transición entre correr y saltar se realizará cuando la variable *enSuelo* pase de verdadero a falso o lo que es lo mismo, que nuestro personaje no está tocando suelo.

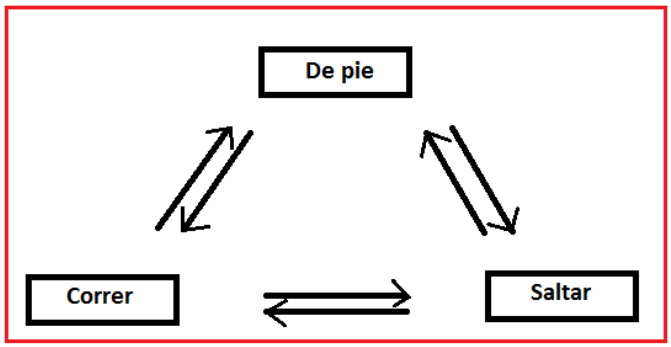
3. **Pasar de saltar a correr**: en la esta transición hacemos uso de las dos variables, tanto la que controla el salto como la velocidad. Nuestro personaje pasará de estar saltando a volver a correr cuando la variable *enSuelo* sea verdadera, es decir que esté tocando suelo y también cuando la velocidad en el eje X sea mayor que 0.01.

4. **Pasar de saltar a estar de pie**: nuestro personaje pasará de saltar a estar de pie cuando tengamos contacto con el suelo, o lo que es lo mismo, que la variable *enSuelo* sea verdadera. Y la velocidad en el eje X sea inferior a 0.01, o lo que es lo mismo, que este quieto.

5. **Pasar de de pie a saltar**: nuestro personaje realizará un salto cuando la variable *enSuelo* sea falsa.

Por defecto, la acción de pie se encuentra como estado de inicio y a partir de ella realizaremos las transiciones al resto.

En la imagen podemos observar las transiciones posibles que puede realizar nuestro protagonista:



4.3 Programación de movimientos del personaje

- **Programación del salto**: nuestro personaje hará la acción de saltar cuando detecte un clic con el botón izquierdo del ratón o bien toquemos la pantalla en los dispositivos táctiles.
  Nuestro personaje puede saltar dos veces seguidas en el aire como máximo, una vez toquemos suelo podremos volver a realizar el salto.

  Para que nuestro personaje sepa cuando puede saltar y cuando no, haremos uso de 3 variables, posición, radio y mascara del objeto.

  - **El objeto vacío**: que contendrá el *transform*, o lo que es lo mismo la posición en la que se encuentra el objeto vacío.

  - **Un radio**: al objeto vacío le asignamos un radio de colisión.

  - **Mascara de capa**: donde comprobamos que el radio está colisionando contra un objeto perteneciente a la máscara.
    La máscara es un tipo de lista a la que le asignamos una serie de objetos que deseamos que tengan determinado comportamiento. Nuestra máscara se llama suelo y esta contiene 3 objetos, los 3 bloques del juego, pequeño mediano y grande.

  Cuando el radio del objeto vacío entra en contacto con un objeto de la máscara, por ejemplo un bloque pequeño, significará que estamos tocando el suelo, por tanto el salto se reiniciará y podremos volver a realizar la acción de saltar.

  En la imagen podemos observar un ejemplo gráfico de lo anteriormente explicado:

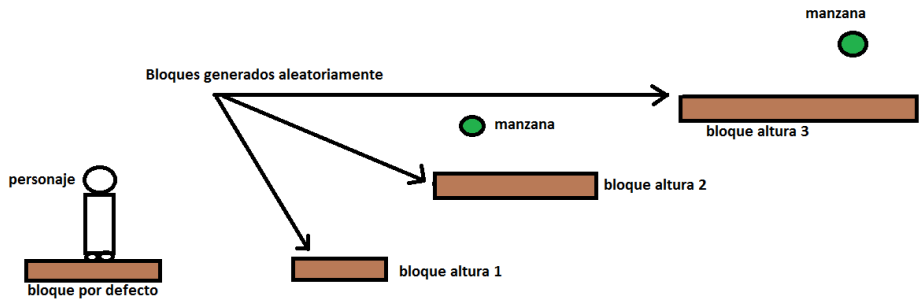
- **Programación de correr**: Este script básicamente aplica una fuerza en el ejeX, también le asignaremos una animación de correr a esta acción, para proporcionarle mayor realismo. La velocidad a la hora de correr es constante, y se mantiene durante toda la partida

4.4 Generador de niveles aleatorio

Llegados a este punto en el que ya tenemos creado a nuestro personaje, hemos programado sus movimientos y dotado de las animaciones para mayor realismo, lo siguiente es ponernos a crear el nivel por donde nos moveremos con nuestro personaje.

El nivel va generándose de forma aleatoria según vamos avanzando con nuestro personaje, y los bloques que vamos dejando atrás se irán destruyendo de forma automática.

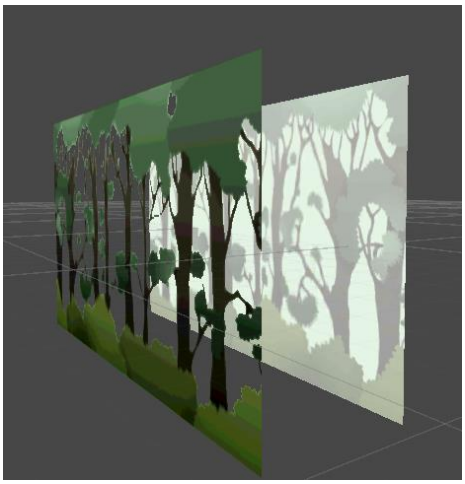
En total se irán generando los bloques en 3 alturas diferentes, las manzanas, que son los objetos a recoger también se generarán de forma aleatoria.



4.5 Marcador de puntos y Scroll Paradax

- **Contador y Marcador de puntos**: El objetivo principal de nuestro juego será el conseguir el mayor número de puntuación, para ello deberemos ir recogiendo las manzanas distribuidas por el nivel. Los bloques también nos proporcionan puntos, cada vez que saltemos a un bloque nuestro contador de puntos se verá incrementado en 1, las manzanas nos darán 5 puntos.
  Nuestra puntuación máxima se guardará en nuestro dispositivo, es decir, cada vez que superemos nuestra puntuación máxima esta se verá actualizada con el nuevo record.

- **Fondo dinámico Scroll Paradax**: es el término que recibe el efecto producido por un fondo en movimiento, proporcionando más dinamismo al juego. Para ello haremos uso de dos imágenes de fondo, una con más profundidad que la otra para dar mayor realismo.



Los dos fondos una vez iniciemos el juego, empezarán a moverse en bucle, el fondo cercano a mayor velocidad que el fondo lejano, esto dará al juego un mayor efecto de profundidad y realismo.

4.6 Testeo y corrección de errores

Terminado el punto anterior nuestro juego ya estaba preparado para ser jugado en PC, pero nuestro juego está enfocado principalmente para jugarse en dispositivos móviles y por tanto debemos exportarlo a una plataforma móvil, en mi caso a mi Tablet Android. Desde Unity podemos exportar nuestro juego a Android de forma muy sencilla, y aunque nuestro objetivo principal es acabar llevando el juego al Play Store, de momento debemos comprobar su funcionamiento y corregir o mejorar cosas que no estén funcionando correctamente.

Para testear el juego además de probarlo en mi dispositivo Android, proporcioné el juego a 2 compañeros, uno de ellos con un dispositivo móvil iOSX y otro con Android. De esta forma he conseguido testear el juego al 100% corrigiendo los errores encontrados y generando una nueva apk con los errores corregidos para volver a realizar el testeo hasta obtener la versión final.

En la siguiente tabla explicativa vemos el coste de tiempo de las tareas realizadas durante el proyecto:

Nombre de la tarea	Fecha de inicio	Fecha final
Parte 1 - Creación del personaje		
Subtarea 1 - Búsqueda de Sprites en internet	06/03/2017	07/03/2017
Subtarea 2 - Creación del protagonista	08/03/2017	09/03/2017
Subtarea 3 - Creación de los Prefabs	10/03/2017	11/03/2017
Subtarea 4 - Elaboración de las animaciones	12/03/2017	20/03/2017
Parte 2 - Diagrama de estados		
Subtarea 1 - Transición de de pie a correr	22/03/2017	22/03/2017
Subtarea 2 - Transición de correr a saltar	23/03/2017	23/03/2017
Subtarea 3 - Transición de saltar a correr	23/03/2017	23/03/2017
Subtarea 4 - Transición de saltar a estar de pie	24/03/2017	24/03/2017
Subtarea 5 - Transición de de pie a saltar	24/03/2017	24/03/2017
Parte 3 - Programación de los movimientos del pj		
Subtarea 1 - Programación de saltar	26/03/2017	31/03/2017
Subtarea 2 - Programación de correr	01/04/2017	10/04/2017
Parte 4 - Generador de niveles aleatorio		
Subtarea 1 - Creación del nivel	15/04/2017	29/04/2017
Parte 5 - Marcador de puntos y Scroll Paradax		
Subtarea 1 - Contador y marcador de puntos	01/05/2017	10/05/2017
Subtarea 2 - Fondo dinámico Scroll Paradax	11/05/2017	20/05/2017
Parte 6 - Testeo y correción de errores	22/05/2017	05/06/2017
Parte 7 - Redacción del proyecto	06/06/2017	10/07/2017

5 Implementación del proyecto

En esta parte del trabajo vamos a profundizar sobre lo referente a las partes y herramientas utilizadas para la implementación del proyecto. Nuestro proyecto consta de un total de 15 scripts con más de 600 líneas de código.

5.1 Plataforma Unity

Durante la creación del juego vamos a utilizar constantemente una serie de componentes Unity indispensables para la creación de este, los cuales son:

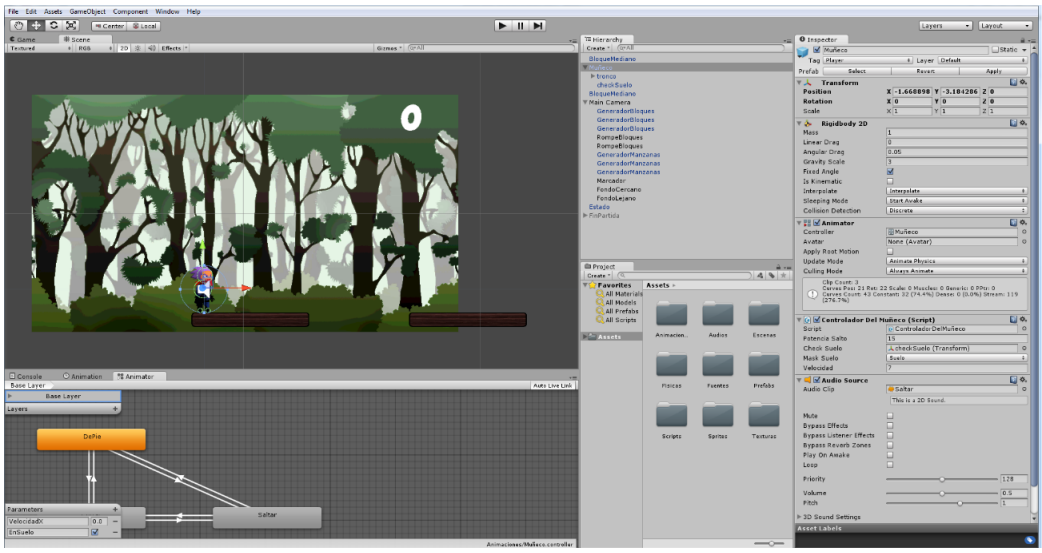
- **El GameObject***:* es el tipo de objeto más importante de Unity. Cada objeto en el juego es un *GameObject*. No obstante los *GameObjects* no hacen nada por sí mismos, estos necesitan propiedades especiales antes de que puedan volverse un personaje, una cámara, un bloque del suelo o los elementos de un menú.

- **Las escenas:** contienen los objetos del juego. Pueden ser usadas para crear un menú principal, niveles individuales, etc.

- **Las cámaras**: son utilizadas para mostrar el mundo del juego al jugador. Siempre debe haber al menos una cámara en una escena y es posible tener más de una a la vez.

- **El transform***:* es usado para almacenar la posición, rotación, escala y el estado de un *GameObject*. Un *GameObject* siempre va a tener de forma adjunta un componente *Transform* que almacene su posición.

Imagen de la interfaz principal de Unity:



5.2 La cámara

En Unity toda escena viene acompañada de una cámara. Son utilizadas para proyectar el mundo de la escena que tenga asignada la cámara, a nosotros, los jugadores.

En Apple Man utilizamos dos cámaras, una para la escena de juego iniciado y otra para la escena de pantalla de inicio.

5.2.1 Cámara de juego iniciado

Es la cámara asociada a la escena de partida iniciada, cuando controlamos a nuestro personaje. Esta cámara tiene contiene 4 scripts.

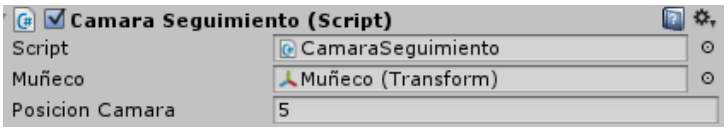
Imagen de la interfaz de la cámara.

- **AspectUtility**: es el script encargado de hacer que la pantalla del juego mantenga la misma relación de aspecto sin que afecte los distintos tamaños del dispositivo donde lo ejecutemos.
  Este script contiene una variable pública que nos permite modificar la relación de aspecto desde el mismo Unity. El ratio relación de aspecto que le hemos asignado es de 1.6, esto quiere decir que sin importar el dispositivo donde ejecutemos el juego tendrá 1.6 píxeles de ancho por cada 1 píxel de alto.

- **Cámara de seguimiento**: es un script que actualiza la posición de la cámara en base a la posición de nuestro personaje, más el valor de la *PosicionCamara*. En nuestro caso el valor de *PosicionCamara* moverá la cámara ligeramente a la derecha, ajustando nuestro personaje a la izquierda de la pantalla.

```
public class CamaraSeguimiento : MonoBehaviour {


    public Transform Muñeco;
    public float PosicionCamara = 5f;
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        transform.position = new Vector3 (Muñeco.position.x+PosicionCamara, transform.position.y, transform.position.z);
    }
}
```

En la vista del script de cámara de seguimiento dentro de Unity, tenemos 3 componentes.



1. **La camaraSeguimiento**: que será el script donde estará el código encargado de posicionar la cámara.

2. **Muñeco transform**: es el componente que contienen la posición de nuestro muñeco en todo momento y que utilizamos en el script para obtener dicha información.

3. **Posición Cámara**: variable pública que nos permite regular la posición de la cámara manualmente y a nuestro gusto.

- **Contador puntos**: es el script encargado de controlar todo lo relacionado con la puntuación, desde mostrar nuestro record hasta ir calculando la puntuación obtenida durante la partida.

- **Pantalla fin de partida**: cada vez que nuestro personaje caiga al vacío la cámara se moverá de posición situándose en el lugar donde tenemos el aviso de fin de partida.

Imagen de la pantalla de game over:

5.2.2 Cámara de pantalla de inicio

Esta cámara es la encargada de mostrarnos el contenido de la escena de inicio de juego, donde nos aparece el título del juego y el botón jugar que iniciará la partida y nos cambiará a la escena de juego iniciado.

Esta cámara únicamente contiene el script *AspectUtility*.

- **AspectUtility**: es el script encargado de hacer que la pantalla del juego mantenga la misma relación de aspecto sin que afecte los distintos tamaños del dispositivo donde lo ejecutemos.
  Este script contiene una variable pública que nos permite modificar la relación de aspecto desde el mismo Unity. El ratio relación de aspecto que le hemos asignado es de 1.6, esto quiere decir que sin importar el dispositivo donde ejecutemos el juego tendrá 1.6 píxeles de ancho por cada 1 píxel de alto.

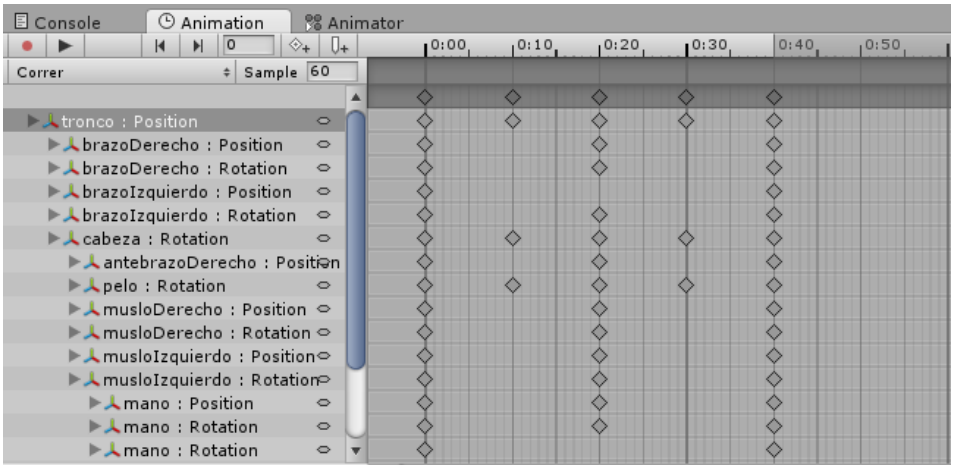
5.3 El jugador

Lo primero que hicimos en Unity fue crear a nuestro personaje, para ello una vez importado el *sprite* a Unity, juntamos las partes hasta darle la forma deseada. Ya creado le implementamos una serie de propiedades, animaciones y Scripts que harán a nuestro personaje más completo.

5.3.1 Animaciones

El siguiente paso es la creación de las animaciones que utilizaremos en nuestro personaje, esta opción viene implementada en la pestaña *Animation*, aquí crearemos un clip donde vamos a modificar el estado de la posición y rotación de las partes del cuerpo involucradas en la animación deseada.

En las siguientes imágenes podemos ver la interfaz de creación de la animación correr ya terminada y la animación en ejecución.



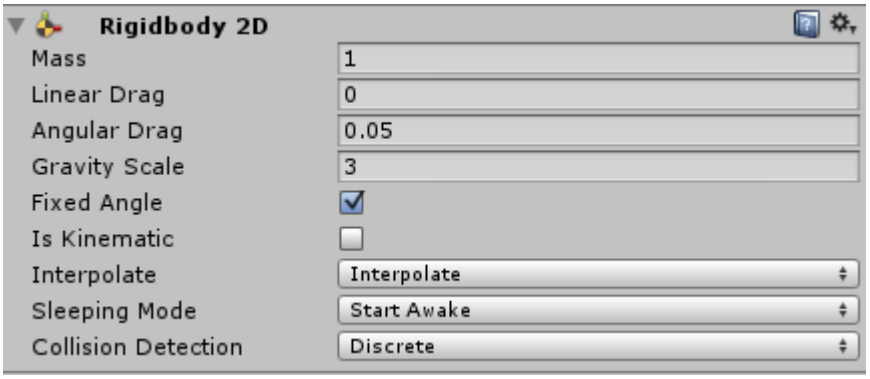
5.3.2 GameObjects

A nuestro personaje le hemos añadido un *GameObject* llamado *checkSuelo*, este objeto lo situaremos justo debajo de los pies de nuestro personaje y nos servirá para indicarle al juego si nuestro este está tocando suelo o no. La función principal de este *gameObject* es ayudarnos en la programación del salto. Si el *GameObject* está tocando un bloque significará que está tocando suelo y por tanto podremos saltar, si por el contrario detecta que no estamos tocando suelo significará que estamos en el aire y no podremos realizar el salto.

5.3.3 Rigidbody 2D

El componente *Rigidbody 2D* permite a nuestro personaje actuar bajo el control de la física. En nuestro caso lo utilizamos para que a la hora de saltar, la gravedad afecte a nuestro personaje y que la acción tenga mayor realismo.

Imagen de la interfaz del Rigidbody:

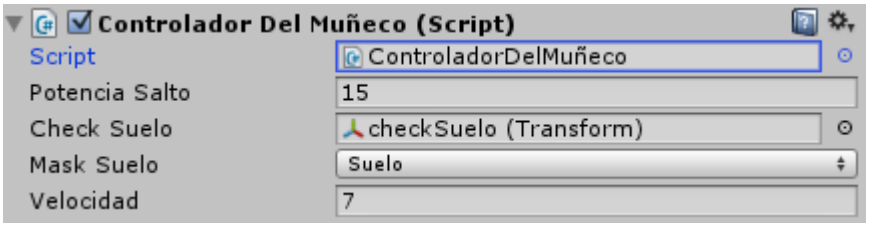


5.3.4 Animator

Para que las animaciones funcionen en nuestro personaje debemos añadirle el componente *animator*, esto nos ayudara a unir las animaciones mediante transiciones y que se activen al ser llamadas.

5.3.5 El personaje

Por último nuestro personaje tiene el componente script llamado *ControladorDelMuñeco*. Este script es donde tenemos la programación de las acciones de nuestro personaje como saltar, doble salto, correr y sus animaciones.



En esta imagen podemos observar variables públicas y ciertos atributos que tenemos asignados en nuestro controlador del personaje y que vamos a explicar a continuación:

- **Potencia Salto**: este es el valor donde indicamos la fuerza a aplicar en el *Rigidbody*, esta fuerza al ser de salto se realiza en el eje Y.

```
if (Input.GetMouseButtonDown (0)) {
    if (correr) {
        if (suelo || !saltoDoble) {  //funciona tanto haciendo click como tocando la pantalla con el dedo
            audio.Play ();
            rigidbody2D.velocity = new Vector2(rigidbody2D.velocity.x, potenciaSalto);
            if(!saltoDoble && !suelo){
                saltoDoble = true;

            }

        }
    }
```

  La variable que define el salto es *potenciaSalto*.

  Aplicamos la fuerza en el *vector2* ya que es un juego en 2 dimensiones y vamos a usar únicamente los ejes X e Y. *PotenciaSalto* es el segundo valor que va a tener el *Vector2*, el que hace referencia al eje Y, y contendrá la fuerza a aplicar a la hora de realizar el salto. Cuanto mayor sea el valor de la variable *potenciaSalto*, más alto saltará nuestro personaje.

- **checkSuelo**: es un objeto asociado a nuestro personaje, este objeto lo situaremos justo debajo de los pies de nuestro personaje y nos servirá para indicarle al juego si nuestro este está tocando suelo o no. La función principal de este *gameObject* es ayudarnos en la programación del salto. Si el *gameObject* está tocando un bloque significará que está tocando suelo y por tanto podremos saltar, si por el contrario detecta que no estamos tocando suelo significará que estamos en el aire y no podremos realizar el salto

- **Mask suelo**: contiene los objetos pertenecientes a la lista suelo, bloque largo, corto y mediano. Trabaja en conjunto con *checkSuelo*. Y nos sirve para indicar que cuando un objeto perteneciente a esta mascara llamada suelo entre en contacto con el *checkSuelo* de nuestro personaje, podamos realizar la acción de salto.

- **Velocidad**: es una variable pública donde indicamos la velocidad constante a aplicar cuando nuestro personaje corra.

```
public float velocidad = 1f;

if (correr) {
    rigidbody2D.velocity = new Vector2(velocidad, rigidbody2D.velocity.y);
        }
```

La velocidad la asignamos al eje X del *vector2* y esta velocidad será aplicada tanto al correr como al saltar.

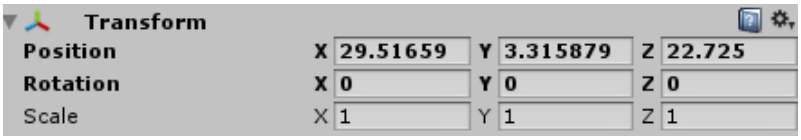
5.4 El nivel

En el nivel de juego se irán generando una serie de elementos aleatorios mientras dure la partida, estos elementos son los bloques y las manzanas.

5.4.1 Generador de bloques

Son los objetos encargados de ir generando los bloques, estos objetos se irán generando según nos vayamos moviendo con la cámara. Para que esto suceda el objeto debe ser hijo de la cámara, así se moverá junto a ella.

El objeto contiene una posición almacenada en el *transform*, esta posición según nos vayamos moviéndonos con la cámara irá cambiando. Este *transform* lo utilizaremos en cada momento para conocer la posición en la que crear el siguiente bloque.



Para generar los bloques hemos creado un script llamado *CreadorBloques.*

Este script contiene un *array* donde vamos a instanciar los bloques (corto, mediano y largo), también tenemos 2 variables *Tmin* y *Tmax*, que utilizaremos como intervalo de tiempo para la generación de estos bloques.

En la imagen podemos observar el código referente a lo anteriormente dicho:

```
public GameObject [] Bloques;
public float Tmin = 1.2f;
public float Tmax = 2.5f;
```

El siguiente método del *CreacionBloques* será el encargado de ir generando los bloques pertenecientes al *array*. Para ello le damos una referencia a los objetos que deseamos instanciar, este objeto será elegido de forma aleatoria entre los 3 objetos que tenemos en la lista de bloques. La posición donde se generará este nuevo bloque será en la que se encuentre el objeto generador de bloques.

Por último mediante la función *Invoke* llamaremos continuamente al método para que genere otro bloque aleatorio dentro del rango de tiempo también aleatorio.

Imagen del método *CreacionBloques*:

```
void CreacionBloques(){
        if (!end) {
                Instantiate (Bloques [Random.Range (0, Bloques.Length)], transform.position, Quaternion.identity);
                Invoke ("CreacionBloques", Random.Range (Tmin, Tmax));
        }
```

En nuestro juego queremos que los bloques se empiecen a generar cuando nuestro personaje empiece a correr. Esto lo haremos utilizando el script *NotificationCenter,* que nos permitirá de forma sencilla mandar mensajes entre los distintos objetos.

En el script creador de bloques le decimos a clase *NotificationCenter* que queremos enterarnos cada vez que el personaje empiece a correr.

```
void Start () {
    NotificationCenter.DefaultCenter ().AddObserver (this, "MuñecoEmpiezaMovimiento");
}
```

Para ello creamos la notificación *MuñecoEmpiezMovimiento* y cada vez que esta sea llamada desde cualquier otra clase del juego, se ejecutará el método perteneciente a la notificación.

```
void MuñecoEmpiezaMovimiento(Notification notificacion){
    CreacionBloques();
}
```

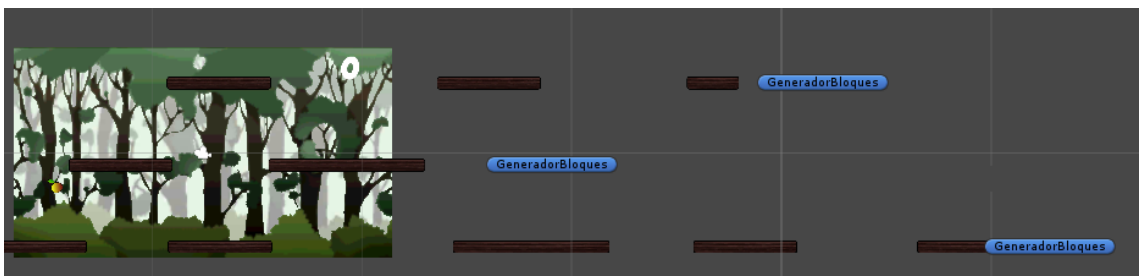
Una vez recibamos la notificación de que nuestro personaje ha empezado a correr, llamaremos al método *CreacionBloques* del script C*readorBloques* para que se empiecen a generar los bloques.

Ya tenemos el script terminado, ahora necesitamos asignarle al objeto generador de bloques el cual contiene el script *CreadorBloques,* los componentes del *array* y los tiempos máximo y mínimo para la creación de los bloques. Estas variables serán las que utilizaremos en el script.

Nuestro nivel consta de 3 alturas para los bloques, por tanto es necesario crear 3 objetos *GeneradorBloques* con su respectivo script *CreadorBloques* por cada altura.

La siguiente imagen nos muestra una escena del nivel de juego:

En esta imagen podemos observar más claramente cómo se van generando los bloques en las tres alturas del nivel según vamos avanzando.



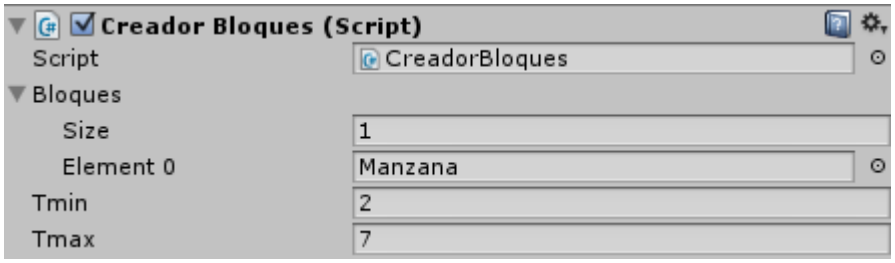
5.4.2 Generadores de manzanas

Son los objetos encargados de generar el objeto manzana en el nivel de juego.

Para la creación de las manzanas vamos a aprovechar el script *CreadorBloques* utilizado en la creación de bloques. Crearemos 3 objetos generadores de manzanas, uno por cada altura.

Recordemos que las variables del script *CreadorBloques* tienen unos valores adecuados para el objeto *GeneradorBloques*, pero no va a ser un problema ya que estas variables son públicas y desde la interfaz del objeto *GeneradorManzanas* podremos modificar estas variables sin afectar al script. Los valores de las variables asignados desde la interfaz de Unity se anteponen a los de las asignaciones en el código de los scripts.

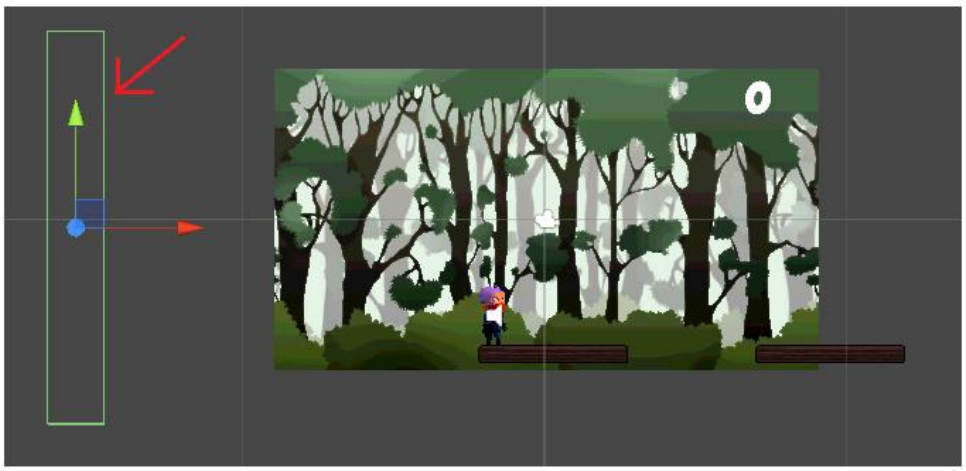
El *array* en este caso será de tamaño 1 ya que únicamente va a almacenar objetos de tipo manzana.



5.4.3 Destructor de bloques

Como ya sabemos, en nuestro juego los objetos bloques se van generando mientras vamos avanzando y en caso de que una partida se alargase mucho, podríamos llegar tener cientos de bloques generados. Esto consume recursos, que por mínimos que sean, cuanto más optimizado esté el juego mejor.

Para ir eliminando los bloques que vamos dejando atrás en el juego según avanzamos, hemos creado un objeto llamado destructor.



Este objeto, que será hijo de la cámara para que se mueva con nosotros mientras avanzamos, tiene los componentes *Collider* y *Rigidbody* para que detecte las colisiones con los bloques. También contiene el script encargado de destruir los objetos bloques que colisionen con el destructor, llamado *RompeBloques.*

```
void OnTriggerEnter2D(Collider2D other){
                Destroy (other.gameObject);

    }
```

Mediante este script llamamos a un método *OnTriggerEnter2D* con el que destruimos los objetos que colisionen con el destructor.

5.5 La Puntuación

En el juego iremos encontrándonos con diferentes elementos que nos proporcionarán puntos. Cada uno de estos objetos se comportará de forma distinta al colisionar con él, cada bloque que pisemos nos dará un punto mientras que las manzanas nos darán 5 puntos.

5.5.1 Obtención de puntos mediante bloques

En la siguiente imagen se muestra una captura del comportamiento del juego a la hora de obtener puntos mediante bloques.

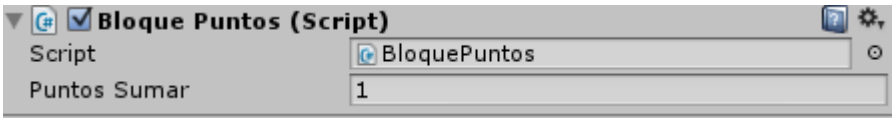
```
public class BloquePuntos : MonoBehaviour {
    private bool ColisionDelPlayer = false;
    public int puntosSumar = 1;
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    void OnCollisionEnter2D(Collision2D collision) {
        if (!ColisionDelPlayer && collision.gameObject.tag == "Player") {
                    ColisionDelPlayer = true;
                    GameObject obj = collision.contacts [0].collider.gameObject;
                    if (obj.name == "pieDerecho" || obj.name == "pieIzquierdo") {
                            ColisionDelPlayer = true;

                            NotificationCenter.DefaultCenter ().PostNotification (this, "AumentarPuntos", 1);
                    }

            }
    }
```

Dentro del código tenemos la variable *puntosSumar,* que indicará la cantidad de puntos a incrementar por cada bloque que toquemos.

Deseamos que solo se aumenten los puntos cuando nuestro personaje, con la etiqueta "*Player*", toque los bloques con las partes *pieDerecho* o *pieIzquierdo.* De esta forma evitamos que aumente un punto al marcador si nuestro personaje toca un bloque con la cabeza.

Cada vez que se cumpla todo lo anterior, notificaremos al método *AumentarPuntos* que sume los puntos.

Este script se almacena en los objetos bloques, todos y cada uno de los bloques que se vayan generando aleatoriamente tendrán este script.



5.5.2 Obtención de puntos mediante manzanas

Las manzanas serán los objetos que nos proporcionarán gran cantidad de puntos. En el siguiente script llamado *Manzana.cs* vemos el comportamiento nuestro personaje cuando colisiona con un objeto manzana.

```
public int puntosSumar  = 5;


void OnTriggerEnter2D(Collider2D collider) {
    if (collider.tag == "Player") {

        NotificationCenter.DefaultCenter ().PostNotification (this, "AumentarPuntos", puntosSumar);


    }
    Destroy(gameObject);
}
```

El código contiene la variable *puntosSumar*, que será la cantidad de puntos a incrementar cuando toquemos una manzana.

Cuando el objeto manzana colisione con nuestro personaje, le notificaremos al método *AumentarPuntos* que sume 5 puntos al marcador.

Cada manzana con la que colisionemos será destruida mediante la funciona *Destroy,* dando así la sensación de que hemos recogido dicho objeto.

El script se almacena en los objetos Manzana, todas y cada una de las manzanas que se vayan generando aleatoriamente tendrán este script.

5.5.3 Contador de puntos.

El script *ContadorPuntos* es el encargado de almacenar el número de puntos que llevamos durante la partida mediante el método *AumentarPuntos*. Este recibirá los datos de los scripts *BloquePuntos* y *Manzana*, los cuales le pasarán la cantidad de puntos a sumar.

```
public class ContadorPuntos : MonoBehaviour {
    public int puntos = 0;

void AumentarPuntos(Notification notificacion){
    int puntosIncrementar = (int)notificacion.data;
    puntos += puntosIncrementar;
    ActMarcador ();
}
```

Cada vez que su método *AumentarPuntos* sea llamado por un evento, irá actualizando con el valor recibido por parámetro y almacenado en *puntosIncrementar*, el resultado de los puntos conseguidos.

5.5.4 El marcador de puntos

El marcador de puntos es un objeto de tipo Texto 3D, este objeto contiene un componente llamado *Text Mesh*. Este componente el cual nos sirve para poner el texto de la puntuación es referenciado en el script *ContadorPuntos*.

```
public class ContadorPuntos : MonoBehaviour {
    public int puntos = 0;
    public TextMesh marcador;
```

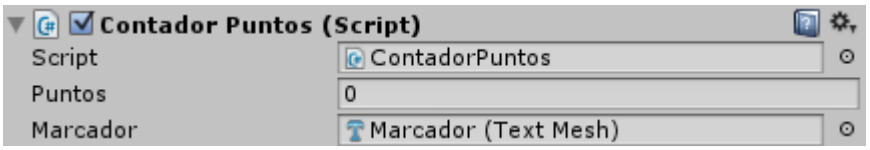
La referencia al *TextMesh* la hacemos mediante la variable llamada marcador.

```
void AumentarPuntos(Notification notificacion){
    int puntosIncrementar = (int)notificacion.data;
    puntos += puntosIncrementar;
    ActMarcador ();
}


void ActMarcador(){
    marcador.text = puntos.ToString ();
}
```

Cada vez que el método *AumentarPuntos* sea llamado para actualizar su valor, mediante *AtcMarcador* haremos una llamada al método *ActMarcador*. Este método obtendrá la cantidad de puntos que hemos conseguido y los guardará en el marcador.

Hemos asignado el script *ContadorPuntos* a la *Main Camera*, así en todo momento se irá moviendo con nosotros según avancemos en el nivel.

Para referenciar el script con el marcador hemos asignado el *Text Mesh* del objeto Marcador a la variable Marcador del script *ContadorPuntos.*



En la siguiente imagen podemos observar el contador marcador durante una partida.

5.6 Fondo Scroll Paradax

Nuestro fondo está conformado por dos niveles de imágenes, fondo cercano y fondo lejano, el fondo lejano estará más alejado en el eje Z.

El efecto *Scroll Paradax* consiste en dotar de una velocidad de movimiento a ambos fondos proporcionándonos una sensación de profundidad y de mayor sensación de movimiento.

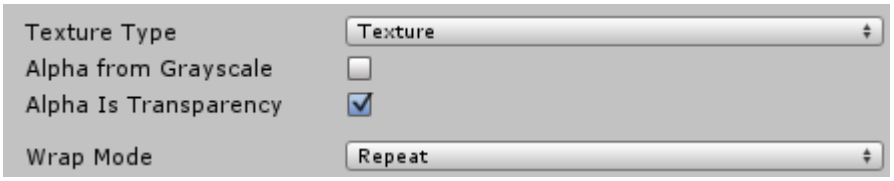

Fondo cercano


Fondo lejano

El fondo lejano lo vamos a crear a partir de un objeto cuadrado, este objeto será hijo de la cámara para que se mueva con nosotros mientras avanzamos.

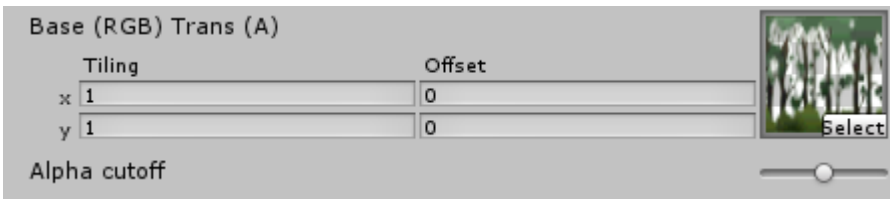
Este objeto va a tener exactamente el mismo tamaño que la cámara. Para terminar vamos a aplicarle la imagen de *FondoLejano* al objeto recién creado. Para el Fondo cercano repetiremos los mismos pasos, crear un objeto llamado *FondoCercano* con el mismo tamaño que la cámara y asignándole la imagen *FondoCercano*.

Ambos texturas tienen la propiedad *Repeat,* la cual hace que nuestra imagen de fondo se repita en bucle durante la partida.



El objetivo principal del fondo es proporcionar dinamismo al juego. El personaje se va a mover hacía la derecha, por lo tanto, vamos a programar el fondo para que se mueva hacía la izquierda dando la sensación de que estamos dejando el paisaje atrás mientras avanzamos.

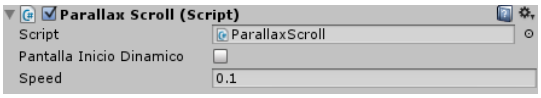
El componente encargado de controlar la velocidad a la que se mueve el objeto fondo se llama Offset, desde un script vamos a modificar este valor para controlar su movimiento.

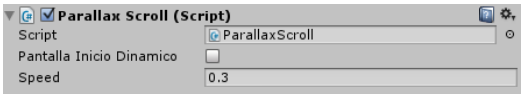


Los dos objetos, *FondoLejano* y *FondeCercano* van contener un script *ParallaxScroll.*cs que será el encargado de modificar la velocidad de movimiento del fondo mediante la variable Offset.

```
public class ParallaxScroll : MonoBehaviour {

    public bool PantallaInicioDinamico = false;
    public float speed = 0f;
    private bool Movimiento = false;
    private float tInicio = 0f;

    // Use this for initialization
    void Start () {
        NotificationCenter.DefaultCenter ().AddObserver (this, "MuñecoEmpiezMovimiento");

        if (PantallaInicioDinamico) {
            MuñecoEmpizaMovimiento();
        }
    }

    void MuñecoEmpizaMovimiento(){
        Movimiento = true;
        tInicio = Time.time;
        }

    // Update is called once per frame
    void Update () {
        if (Movimiento) {
                    renderer.material.mainTextureOffset = new Vector2 ((Time.time - tInicio) * speed %1, 0);
                }
    }
}
```

Como podemos observar en el código, el fondo solo iniciará su movimiento cuando nuestro personaje haya empezado a correr. Una vez recibamos la notificación de que el personaje ha iniciado su movimiento, modificaremos el valor del Offset en el eje X multiplicándolo por speed, que es la variable que mide la velocidad a la que se moverá el fondo.





Por último asignamos el script a ambos objetos de fondo y modificamos el valor speed, el *Fondolejano* se moverá más despacio que el *FondoCercano* que se moverá a 0.1. El que el fondo lejano se mueva más despacio que el cercano se debe a que dará mayor sensación de profundidad al juego.

5.7 Menú de inicio

Es la primera escena del juego y aparece automáticamente al iniciarlo.

Esta escena tendrá un texto 3D con el título del juego y un botón llamado Play con un *Rigidbody* para que al hacer clic encima del botón nos cambie de escena.

El botón *Play* contiene un script que es el encargado de realizar este cambio de escena al pulsar sobre el botón.

```
void OnMouseDown(){
    Application.LoadLevel ("EscenaJuego");
}
```

Como vemos en el script, al hacer clic sobre el botón nos cambiará a la escena *EscenaJuego* que es la escena de partida iniciada.

En la siguiente imagen podemos observar el resultado final de la escena de inicio.

5.8 Menú Game Over

Es el menú que se mostrará cuando nuestro personaje caiga al vacío, es decir, cuando perdamos.

El menú está formado por una cámara llamada *FinPartida* y con el tamaño exacto a la cámara principal.

En la siguiente imagen podemos observar los componentes que le hemos asignado a la cámara para conseguir el resultado deseado en el menú de *game over*.

Los componentes son:

- **Quad**: es un cuadrado que cubre la totalidad de la vista de la cámara, este cuadrado hará la función de filtro de transparencia entre la pantalla de *game over* y el fondo del juego.

- **MensajeFinPartida**: es un texto 3D que contiene el mensaje *game Over*.

- **TextoPuntosConseguidos**: Texto 3D con el mensaje Score.

- **TextoPuntosRecord**: texto 3D con el mensaje Record.

- **ValorPuntosRecord**: texto 3D que contendrá el máximo de puntos que hayamos obtenido todas nuestras partidas.

- **ValorPuntosConseguidos**: texto 3D que mostrará la cantidad de puntos conseguidos en la reciente partida.

- **ReiniciarPartida**: texto 3D con el mensaje INTENTAR DE NUEVO!, y que contiene el script *IniciarJuego*. Al pulsar sobre el texto 3D cambiaremos a la escena *EscenaJuego* para volver a jugar.

En la siguiente imagen podemos ver la parte del código encargado de cambiar de escena al pulsar sobre el mensaje de intentar de nuevo.

```
void CargarNivel(){
    Application.LoadLevel ("EscenaJuego");
}
```

La cámara que contiene el menú *game over* está desactivada por defecto para que aparezca en el momento deseado, es decir, al perder la partida. Esto lo hemos hecho mediante la *Main camera* de la escena, que se encarga de activar este menú de *game over* mediante un script llamado *PantallaFinPartida*.

```
void Start () {
    NotificationCenter.DefaultCenter ().AddObserver (this, "MuñecoMuerto");

}

void MuñecoMuerto(Notification notifiacion){

    FinPartida.SetActive (true);
}
```

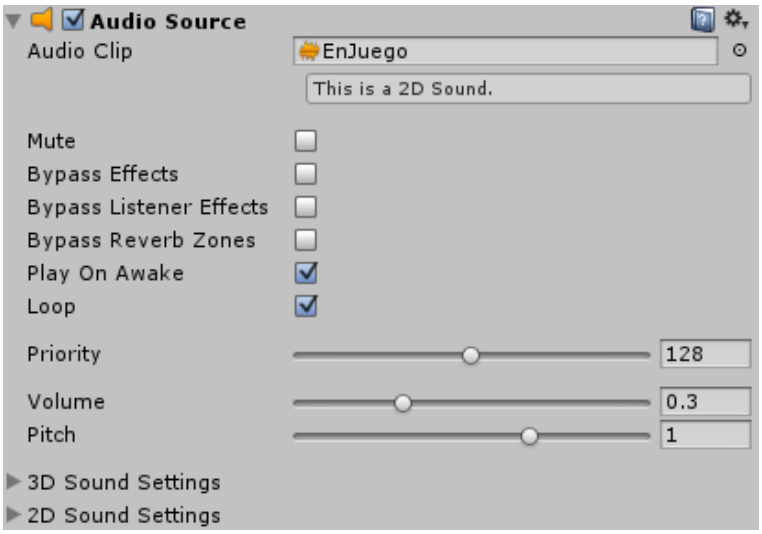
En la imagen del código podemos observar que estamos subscritos a la notificación *MuñecoMuerto*. Cuando nuestro personaje caiga al vacío este script será notificado de que nuestro personaje ha muerto y ejecutaremos el método del mismo nombre, es decir *MuñecoMuerto* que nos activará la escena de fin de partida.

5.9 Música y sonidos

En nuestro videojuego hemos incluido un total de 6 clips de audio:

1. **En el menú de inicio**
2. **Al pulsar un botón como jugar**
3. **Una canción en bucle durante la partida**
4. **Al realizar la acción de saltar**
5. **Al coger una manzana**
6. **Al caer al vacío**

El clip de sonido se coloca dentro del componente de Unity deseado, por ejemplo el sonido de saltar está dentro del objeto personaje y el clip que suena durante toda la partida se encuentra dentro de la *Main camera* de la escena del juego.



En la imagen podemos observar el componente *Audio Source* que contiene la canción que sonará durante toda la partida.

Este componente tiene 2 atributos activados:

1. **Loop**: que hará que la canción se repita en bucle mientras dure la partida.

2. **Play On Awake**: atributo utilizado para que la canción empiece a sonar en cuanto se inicie la escena de juego.

5.10 Objetos del nivel

Durante el nivel nos vamos a encontrar dos objetos, manzanas y bloques. Ambos objetos nos proporcionarán puntos, aunque la función principal de los bloques es ayudarnos a seguir avanzando por el nivel y las manzanas proporcionarnos una gran cantidad de puntos.

5.10.1 Bloques

Los bloques se van generando en posiciones y tamaños aleatorios en los tres niveles de altura a lo largo de la partida. Son objetos necesarios en el juego, ya que son objetos sólidos en los que nuestro personaje puede estar encima sin caer.

Cada bloque al que saltemos nos proporcionará un punto a nuestro marcador de puntos.

En la siguiente imagen podemos observar el aspecto de los distintos bloques en el juego, desde el corto al largo:

5.10.2 Las manzanas

Son objetos que al igual que los bloques se van generando aleatoriamente en los tres niveles de altura según vamos avanzando. La función principal de las manzanas es darnos puntos, ya que son la fuente principal de puntos en el juego proporcionándonos 5 por cada una que recojamos.

En la siguiente imagen podemos observar el aspecto de las manzanas en el juego:

6 Futuras mejoras

Durante el verano se tiene en mente ir añadiendo nuevas funcionalidades al juego. La idea es que estén terminadas para antes de la exposición del proyecto con fecha en septiembre, incluyéndolas y explicándolas durante esta.

Algunas de las ideas de mejora para el juego son:

- Subir Apple Man a la plataforma Google Play para que de forma gratuita pueda estar a disposición de quien desee jugarlo.

- Integrar en el juego un sistema de logros mediante Google Play, de esta forma los jugadores tendrán una recompensa al cumplir ciertos objetivos en el juego, como por ejemplo conseguir 500 puntos.

- Un sistema de ranking con una lista de las mejores puntuaciones obtenidas en el juego.

Para implementar todas estas nuevas funciones deberemos programar bastante código, así que, ¡manos a la obra!

7 Agradecimientos

Primero agradecer a Jordi Joan Linares Pellicer por lo aprendido con él, tanto en la asignatura de programación de videojuegos como durante la realización de este proyecto.

A los testers del juego por ayudarme a encontrar errores.

También agradecer a toda la comunidad de Unity y su foro, por ayudar de forma desinteresada a los que nos encontramos con algún problema puntual.

8 Bibliografía

Sprites de personajes y creación de animaciones
http://totallysweetredhoodie.blogspot.com.es

inScope Studios. Juego 2D plataformas
https://www.youtube.com/playlist?list=PLX-uZVK_0K_6VXcSajfFbXDXndb6AdBLO

Brackeys. How to make a 2D Plataformer
https://www.youtube.com/playlist?list=PLPV2KyIb3jR42oVBU6K2DIL6Y22Ry9J1c

Unity Technologies. 2D Character Controllers
https://unity3d.com/es/learn/tutorials/topics/2d-game-creation/2d-character-controllers

Dave Calabrese. Unity 2D game development, 2014.
ISBN 978-1849692564