UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE GRADO

Desarrollo de un videojuego 2D "Walkin, the sword hero"

Presentado por:

Guillermo Linares Mauri

Tutor:

Prof. Jordi Joan Linares Pellicer

Curso Académico 2015 / 2016

RESUMEN

Este proyecto consiste en la realización de un videojuego de plataformas y exploración 2D con el motor de desarrollo Unity, con un énfasis en la gestión y creación de pequeñas inteligencias artificiales que serán los enemigos. Se pretende crear un juego para plataformas móviles, especialmente para las plataformas Android aunque en su principio es diseñado para plataforma de escritorio. Busca también aportar una visión general de la creación de un videojuego y de las distintas etapas de su desarrollo.

ABSTRACT

This project consists to develop a 2D platform and exploration videogame using the Unity engine. There is an emphasis in the creation and management of small artificial intelligences which will be the enemies. The objective is to create a game for mobile platforms, especially to Android platforms although at the beginning the game is designed to desktop platforms. It is meant to provide an overview of the creation of a videogame and all the stages of its development.

Palabras clave: videojuego, 2D, Unity, plataformas, escritorio, móvil
Keywords: videogame, 2D, Unity, platforms, desktop, mobile

Tabla de contenidos

1 INTRODUCCIÓN

1.1 Motivación

A lo largo de mi trayecto por la carrera, el mundo de los videojuegos ha entrado cada vez más en mi entorno haciendo que sea la principal motivación para realizar este proyecto.

Todo es debido a conseguir un título académico realizado en la Escola Politècnica Superior d'Alcoi el cual me acreditaba con un Diploma de Extensión Universitaria en Diseño Creativo de Videojuegos. Además, durante mi estancia en Letterkeny, Irlanda, como estudiante Erasmus tuve la posibilidad de poder realizar dos videojuegos, uno 2D y otro 3D, para una asignatura que pretendía la enseñanza de sus alumnos en la creación de juegos sencillos.

La industria de los videojuegos se ha visto como una gran potencia de nuestro mundo actual, pudiendo llegar a superar tanto a la industria del cine como la de la música.
Esto es debido, por una parte, en la reducción de barreras para los creadores gracias a herramientas como Unity o Unreal Engine que posibilitan la creación de videojuegos o aplicaciones de alta calidad con muy pocos recursos.

Este proyecto pretende servir para reforzar los conocimientos adquiridos y obtener más experiencia con el motor de Unity y el lenguaje de programación C#, llegando a crear un videojuego completamente pulido, apto para todos los públicos y variado.

El mercado de los videojuegos va cambiando convenientemente con el gusto de los jugadores. Es por ello que se ha optado por la realización de un juego que tenga las características que atraen a un mayor número de jugadores: los juegos de plataformas y lo "retro". Es por ello que los personajes, los elementos del terreno, la música…están adaptados a videojuegos retro, como si pareciera de un videojuego antiguo.

La realización de un videojuego, precisa de conocimientos en muchos campos que ingeniero informático debe de dominar, como las arquitecturas software, teoría de autómatas, inteligencia artificial…además de otros rasgos más generales que son el diseño del juego, niveles, sonido, música, matemática o la física. Por todo se considera un proyecto completo que sirve para culminar de forma adecuada como Trabajo de Final de Grado en el que se puede apreciar los conocimientos adquiridos durante el transcurso de la carrera.

1.2 Trasfondo del Trabajo de Fin de Grado

Walkin The Sword Hero se basa en las aventuras de Walkin, un joven caballero que quiere embarcarse en busca de nuevos adversarios y nuevos mundos.
Con la ayuda de su espada deberá de ir abriéndose paso entre sus enemigos e ir superando las trampas que estos le ponen por el camino.
En cada mundo deberá de vencer a todos los enemigos que custodian un castillo, el cual será la apertura para un nuevo mundo, nuevos enemigos y nuevas aventuras.

1.3 Objetivos

El objetivo principal del proyecto es el diseñar e implementar un juego de plataformas 2D para plataforma móvil pasando primero por su creación para plataforma de escritorio. El juego contará con la jugabilidad completa del personaje principal y con todos los sistemas principales del juego. Debe de ofrecer una experiencia agradable a la hora de jugar para posibles ampliaciones.

Se desea profundizar en las características del motor de desarrollo Unity. También se busca ganar experiencia y aprender el proceso de creación de un videojuego completo, prestando atención en el diseño del juego y de los niveles que lo componen. Esto es necesario para la elaboración de un juego de calidad.

El uso de pequeñas inteligencias artificiales que permitan el funcionamiento para todos los tipos de enemigos que el personaje principal se irá encontrando a lo largo de todos los niveles.

Encontrar la mejor forma para conseguir los resultados que se desean para lograr hacer un juego con detalles “retro”, ya sea con música en 8 bits o personajes pixelados que atraigan a los jugadores.

Una vez este realizado para plataforma escritorio, conseguir convertir el juego a plataforma móvil implementando un sistema de botones para el control del personaje así como el control táctil para todo el juego.

Finalmente, lograr subir la apk del videojuego a la plataforma de servicio de Google para la instalación de aplicaciones y juegos conocida como “Play Store” en donde cualquier persona podrá disfrutar de los resultados obtenidos de este proyecto.

2 DESCRIPCIÓN DE LA HERRAMIENTAS UTILIZADAS

2.1 Unity

En la actualidad existen una gran cantidad de motores para la creación de videojuegos, cada uno con sus características. El utilizado para este proyecto es Unity.

Unity es un motor para la creación de aplicaciones o videojuegos que soporta tanto el 2D como el 3D. Posee una versión gratuita, apta para todo el público y una versión Pro de pago. La utilizada aquí es una versión gratuita.

Los proyectos desarrollados por versiones gratuitas de Unity pueden comercializarse siempre y cuando no se superen los 100 000 dólares al año.

El editor es multiplataforma y exporta a muchas plataformas distintas, como las siguientes:

- **Plataformas Web**: WebGL
- **Plataformas PC**: Windows, SteamOS, GNU/Linux…
- **Plataformas móviles**: Android, iOS, Windows Phone…
- **Smart TV**: Samsung Smart TV, Android TV…
- **Consolas**: Play Station 4, Xbox 360, Wii U
- **Dispositivos de realidad virtual**: Oculus Rift, Google Cardboard…

Además posee varias versiones, las cuales van mejorando conforme las van actualizando. Para este proyecto se ha utilizado la versión Unity 5.2.1f1.

Para la programación, soporta C# y Javascript. Unity posee una documentación muy amplia y detallada y posee una gran comunidad a nivel mundial. En este proyecto se ha optado por utilizar el lenguaje C# dada la experiencia obtenida durante el Diploma de Extensión Universitaria y los proyectos realizados durante el transcurso de la carrera además de su similitud con C++ o Java.

2.2 Microsoft Visual Studio 2015

Microsoft Visual Studio es un entorno de desarrollo integrado para sistemas operativos Windows, capaz de soportar bastantes tipos de lenguajes de programación.
A la hora de programar los scripts de este proyecto, Unity interactúa con este programa para la coordinación del código de los scripts con el proyecto sobre el que se está trabajando.

2.3 GX SCC

GX SCC es un programa que permite la transformación de canciones a formato de 8 bits. Para su transformación es preciso que la canción introducida esté en formato midi, el cual permite al programa coger la tablatura de la canción y convertirla.

2.4 Herramientas de servicios web

2.4.1 Piskelapp.com

Piskelapp es una página web que permite el desarrollo de imágenes, trabajando exclusivamente mediante los pixeles. Permite la exportación de tus proyectos en varios tipos de formato de imagen como png o gif.

Con esta página web se han realizado varios elementos y objetos que participan dentro del proyecto.

2.4.2 Mp3cut.net

Mp3cut.net es una página web que permite cortar canciones y volver a guardarlas en tu ordenador en formato mp3 con el corte que se haya realizado.

3 DISEÑO Y DESARROLLO DEL VIDEOJUEGO

3.1 Concepto

Walkin The Sword Hero es un juego plataformas 2D de aventuras que va avanzando por niveles. Existe un menú selector de niveles el cual muestra si los niveles se han pasado con una, dos o tres estrellas.

El jugador controla a Walkin, el joven caballero que va en busca de nuevas aventuras y nuevos enemigos.

Es posible morir durante el nivel. Si ocurre, el nivel se resetea y se vuelve a comenzar desde el inicio. Todo esto es debido a que los enemigos lo único que desean es acabar con Walkin y destruirlo. Para ello utilizaran todas sus armas, trampas y demás elementos para lograrlo.

Para avanzar por el nivel, Walkin deberá de moverse principalmente siempre hacia la derecha, pues el final del nivel estará en esa dirección siempre.

Para pasarse un nivel al 100% el jugador deberá de recoger todas las monedas que se encuentren por el nivel, ya sean por las que hay repartidas por todo el terreno o por las que otorgan los enemigos al derrotarlos.

Todo el juego está en inglés, ampliando así el número de jugadores a todo el planeta.

3.2 El jugador

El protagonista, Walkin, se compone de cuatro botones que permitirán su jugabilidad durante todo el juego.

- **Botón Flecha Izquierda**: permitirá al jugador mover a Walkin en dirección a la izquierda.

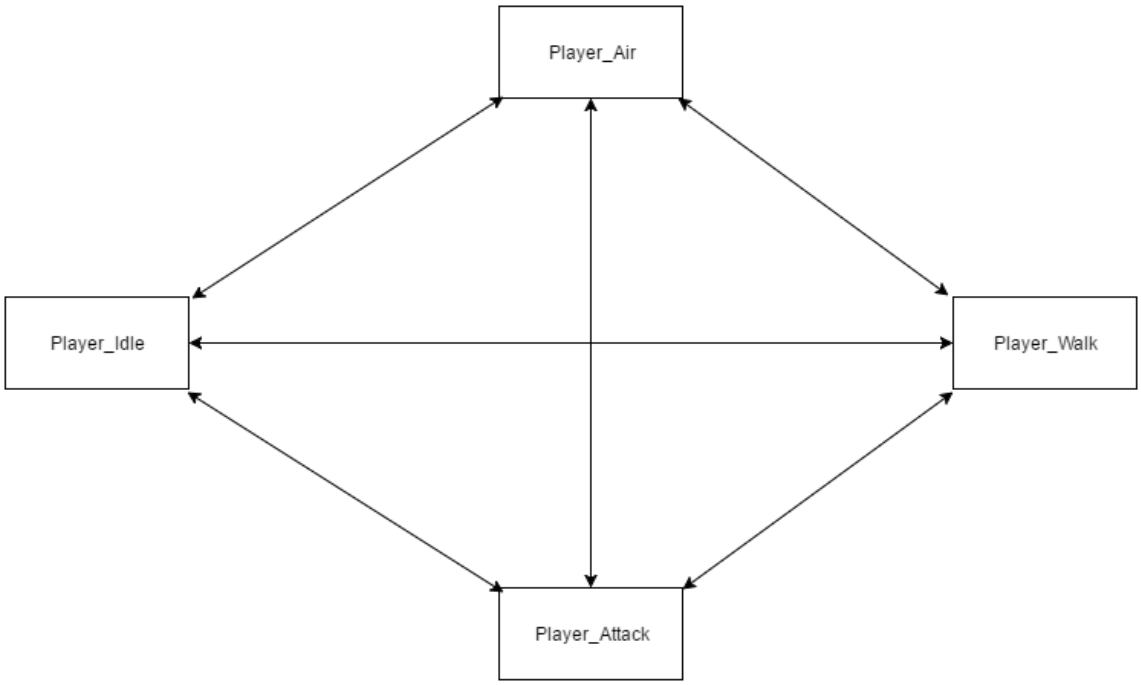
- **Botón Flecha Derecha**: permitirá al jugador mover a Walkin en dirección a la derecha.

- **Botón Flecha Arriba**: permitirá a Walkin poder saltar. Si el botón es presionado dos veces, Walkin conseguirá realizar un doble salto, elevándolo más alto y permitiéndole saltar a más distancia.

- **Botón Espadas**: permitirá a Walkin atacar. Esté botón servirá para enfrentar a Walkin cuerpo a cuerpo contra todos los enemigos o trampas que se encuentre por el camino y destruirlas.

Es posible la combinación de varios botones a la vez, como por ejemplo darle una dirección a los saltos, saltar y mientras está en el aire atacar con la espada… Poco a poco, mientras avanza por los niveles, el jugador deberá de perfeccionar su técnica para poder superar todos los obstáculos que ofrecerá cada nivel.

A continuación se ofrece un diagrama que muestra cómo funciona el comportamiento del personaje Walkin durante el juego.



Para entenderlo mejor vamos a proceder con una explicación:

El personaje Walkin posee tres variables que van a estar en constante cambio que funcionan para cambiar entre los cuatro diferentes estados del personaje.

- **Speed**: indica la velocidad
- **Grounded**: indica si está tocando el suelo o no
- **Attacking**: indica si está atacando o no

Estas variables van a indicar el paso entre una transición a otra dependiendo de las características que se le correspondan: presión de botones e interacción con el terrero del nivel.

Player_Idle significa que el personaje está quieto y está realizando la animación que le corresponde para cuando el personaje está detenido el juego.

Player_Walk significa que el personaje está andando y está realizando la animación de andar.

Player_Air significa que el personaje está saltando y se encuentra en el aire además de realizar la animación que le corresponde.

Player_Attack significa que el personaje se encuentra atacando y realizando su animación de atacar.

3.3 Enemigos

Los niveles están repletos de enemigos que intentaran acabar con la vida de Walkin para impedir su propósito. Estos poseen características que les permitirán ir a por Walkin y atacarle.

Podemos diferenciar dos tipos de enemigos: enemigos de cuerpo a cuerpo y enemigos de ataque a distancia.

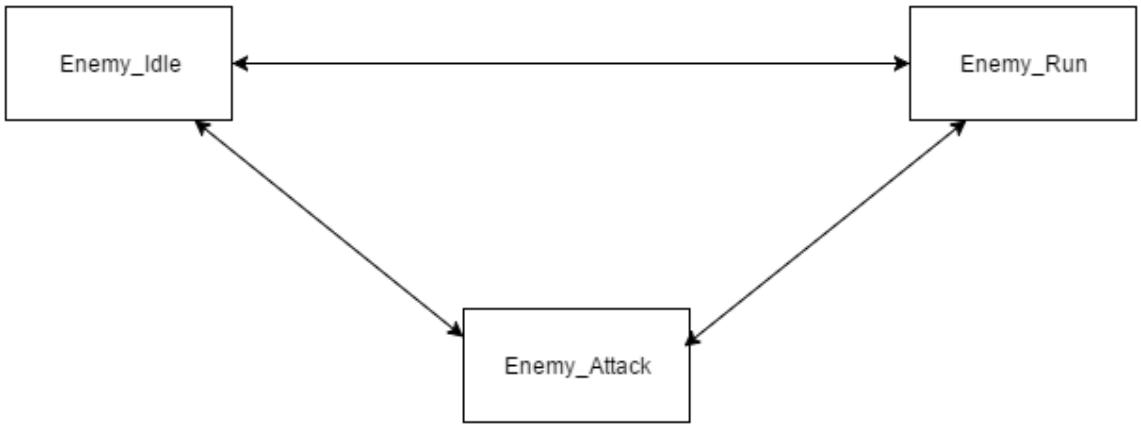
3.3.1 Enemigos cuerpo a cuerpo

Los enemigos cuerpo a cuerpo son los que se enfrentaran a Walkin en las distancias cortas. Cuando el jugador entre dentro del rango de alerta de un enemigo de cuerpo a cuerpo, lo alertará y este comenzará a correr hacia donde se encuentra y una vez lo haga, le atacará hasta que logre eliminar el jugador o morir en su intento.

Esta clase de enemigos trabaja con tres variables que irán variando dependiendo de lo que ocurra con el jugador dentro del nivel.

- **Awake**: indica si el jugador ha entrado en la zona de alerta o no
- **Run**: indica si el enemigo está corriendo hacia el jugador o no
- **Attack**: indica si el enemigo está atacando al jugador o no

Con el siguiente diagrama se muestran los estados de los enemigos cuerpo a cuerpo:



Las variables anteriores son las que modificaran el estado del enemigo para sus tres estados:

Enemy_Idle: significa que el enemigo se encuentra quieto y realizará la animación de cuando se encuentra parado.

Enemy_Run: significa que el enemigo se encuentra corriendo hacia el jugador y se encuentra haciendo la animación de correr.

Enemy_Attack: significa que el enemigo se encuentra atacando al jugador y realizando la animación de atacar.

Dentro de los tipos de enemigos cuerpo a cuerpo encontramos los siguientes, aumentando en dificultad de menor a mayor: espadachín, piquero, asesino, ninja y vikingo. Más adelante se explicará cada uno su implementación.

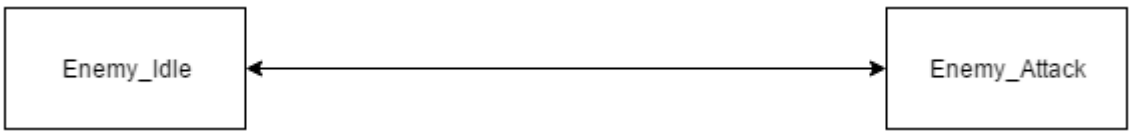
3.3.2 Enemigos a distancia

Los enemigos a distancia son los enemigos que van a atacar a Walkin con armas de largo alcance como arcos o pistolas. Al igual que los enemigos cuerpo a cuerpo, cuando el jugador entre dentro del área de alerta de estos, los despertará y comenzarán a dispararle hasta que consigan derrotar a Walkin o contrario.

Esta clase de enemigos trabaja únicamente con dos variables que iran cambiando conforme avance el jugador por el nivel.

- **Awake**: indica si el jugador ha entrado en la zona de alerta o no
- **Shot**: indica si el enemigo está disparando al jugador o no

Con el siguiente diagrama se muestran los estados de los enemigos de ataque a distancia:



Las variables anteriores son las que modificaran el estado del enemigo para sus tres estados:

Enemy_Idle: significa que el enemigo se encuentra quieto y realizará la animación de cuando se encuentra parado.

Enemy_Attack: significa que el enemigo se encuentra disparando al jugador y realizando la animación de disparar.

Dentro de los tipos de enemigos de ataque a distancia encontramos los siguientes, aumentando en dificultad de menor a mayor: arquero, mosquetero y pirata. Más adelante se explicará cada uno su implementación.

3.4 Menú

Cuando se inicia el juego, la primera pantalla que vamos a poder observar es la del Menú. En él vamos a encontrar el cartel del juego además de tres botones para interactuar:

- **New Game**: al darle comenzará el juego desde el principio eliminando todos los datos borrados de partidas anteriores.

- **Load Game**: ejecutará el juego con los datos guardados de anteriores partidas dejando al jugador seguir por donde se quedó antes de salir del juego.

- **Exit Game**: la aplicación se cerrará y dejara de ejecutarse.

3.5 Selectores de niveles

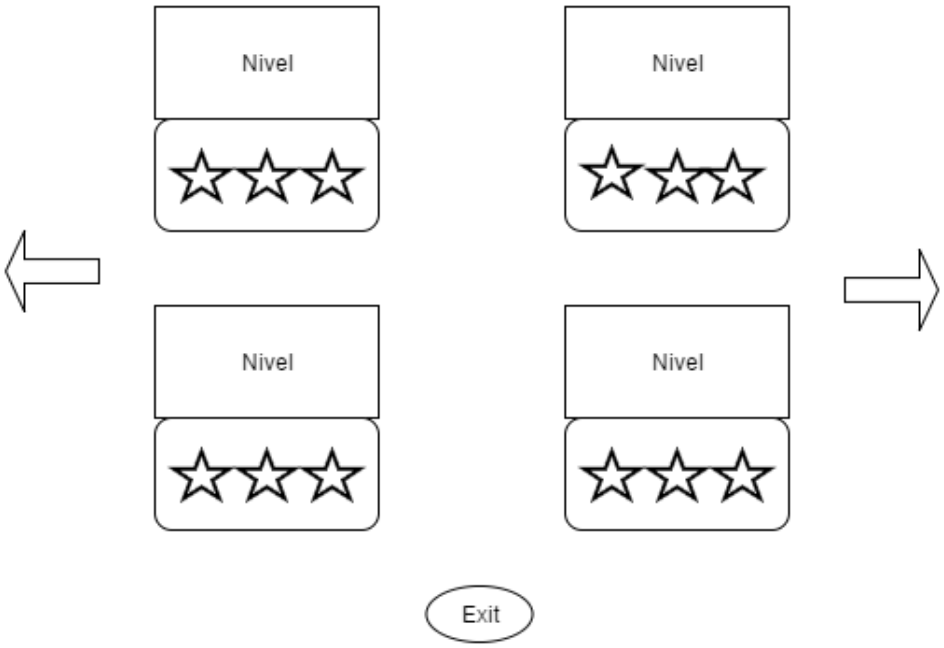
Al darle al botón de la pantalla de menú nos enviara a las pantallas de selectores de niveles, dependiendo de si hemos elegido “New Game” o “Load Game” se mostrarán de diferente forma.

En estas pantallas encontraremos que caben pon pantalla de selector cuatro niveles, los cuales tendrán un botón indicando el número de nivel que son para poder entrar dentro del nivel en específico.

Debajo de dichos botones, encontraremos las tablas en las que según el número de monedas recogidas durante el nivel, mostraran una, dos o tres estrellas.

A los laterales tendremos botones con flechas que nos permitirán transitar entre los selectores de niveles para avanzar o retroceder entre niveles.

Por la zona inferior de la pantalla se encontrará un botón que permitirá salir del juego y dejar de ejecutarlo.



3.6 Vista del juego

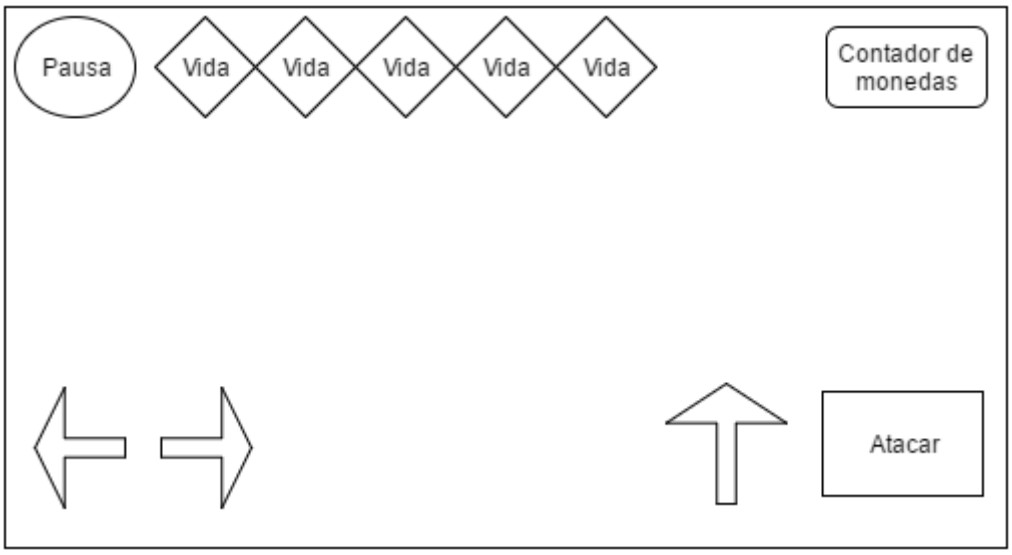
Una vez dentro de un nivel se mostrarán diferentes elementos que estarán presentes dentro de todos los niveles del juego.

Empezando por la parte superior encontraremos a la izquierda un botón de pausa que permitirá detener el juego además de abrir nuevas opciones, más adelante se explicará su funcionamiento.

Seguidamente del botón de pausa encontraremos las vidas del jugador. Estás representaran en número actual de vidas de Walkin durante el juego, si pierde vidas desaparecerán y si gana vidas, reaparecerán.

En la esquina superior derecha aparecerá el contador de monedas que indicará el número de monedas recogidas hasta el momento en el nivel.

En la zona inferior es donde encontraremos los cuatro botones previamente explicados en el apartado de jugador: flecha izquierda, flecha derecha, flecha arriba y espada.



3.7 Menú de pausa

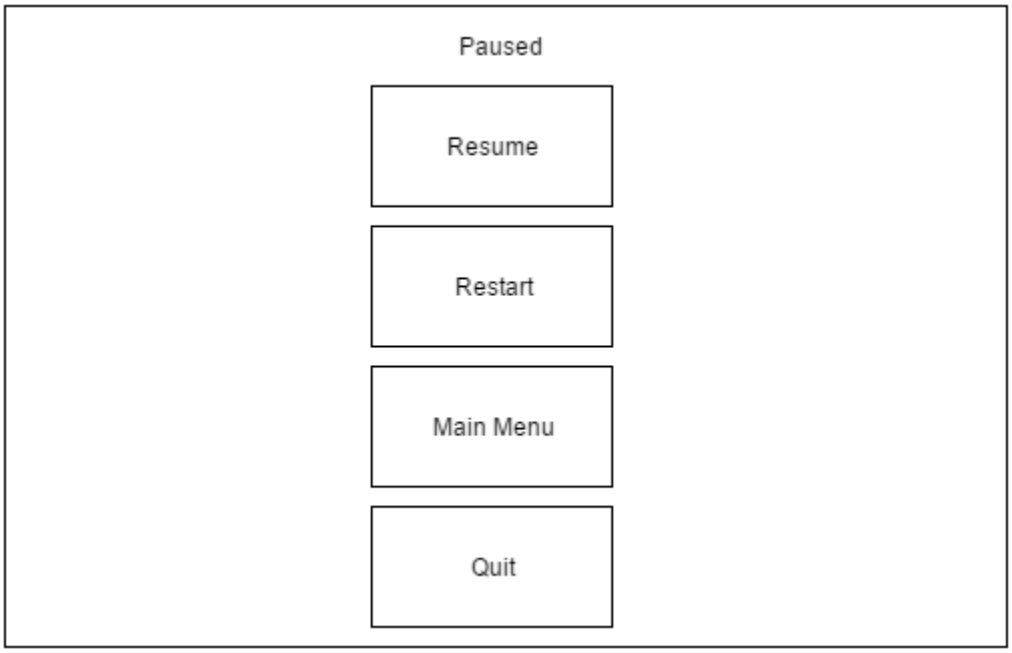
Cuando el jugador toque el botón de pausa de la vista del juego, el juego se detendrá pasando dicho visor a segundo plano y pasando a un primer plano un nuevo selector de botones con diferentes opciones. Serán las cuatro siguientes:

- **Resume**: al darle inmediatamente el juego se reanudará y podrá seguir jugando

- **Restart**: al darle, el nivel volverá a empezar desde el principio, se reseteará

- **Main Menu**: al darle saldrá del nivel en cuestión volviendo a la vista del selector de niveles

- **Quit**: detendrá la aplicación por completo, cerrándola y saliendo del juego



3.8 Niveles

Cada nivel dentro del juego es completamente distinto uno de otro. Para hacer esto posible se debe de tener en cuenta los siguientes factores:

- **Distancia**: unos niveles son más largos que otros, dependiendo de la dificultad que muestren

- **Enemigos**: conforme se avanza por los niveles aparecen más enemigos y diferentes tipos de estos

- **Obstáculos y trampas**: al avanzar por los niveles van apareciendo nuevos elementos que son trampas y nuevos obstáculos para detener al jugador

- **Música**: cada cuatro niveles, es decir, un selector de niveles, la música que se reproduce dentro de esos niveles cambia.

4 PLANIFICACIÓN

El proyecto se ha establecido para ser completamente operativo en la duración de siete meses. Los intervalos de tiempo son grandes debido a que al estar asistiendo también a clases y tener exámenes, el tiempo libre que sobra es lo que se dedica a la elaboración de este proyecto.

La planificación de este proyecto se ha dividido en cinco partes fundamentales.

4.1 Parte 1 – Creación del Primer Nivel

Al comenzar desde cero, es esencial la creación del primer nivel para la configuración del videojuego. Para ello, antes de comenzar a trabajar en el proyecto, previamente, se realizó una búsqueda intensiva por internet de recursos gratuitos para que conformaran los personajes, elementos de terreno, backgrounds…para tener una pequeña idea inicial de como comenzar a montar el primer nivel.

La duración de esta parte se ha previsto de un mes, en este caso, el mes de Febrero de 2016. A cada subtarea se le ha asignado unos días en específico para la finalización de esta y tener un objetivo marcado por días y semanas.

En esta parte se han realizado las siguientes subtareas:

- **Programación básica del Player**: lo primero de todo ha sido la creación del Player. Para ello se han elaborado las animaciones necesarias que va a realizar el personaje. A continuación se ha construido un script en el cual se ha programado el movimiento básico a través del teclado y un pequeño salto.

- **Montar los elementos que conforman el nivel**: una vez ya tenemos el Player montado, hay que montar un pequeño nivel inicial, a modo tutorial. Se han colocado todos los elementos necesarios: suelo, backgrounds, carteles…para conformar así el primer nivel.

- **Programación de los elementos y testeo del primer nivel**: al tener ya el nivel montado, hay que programar a partir de Unity los elementos del nivel para que interactúen adecuadamente con el personaje durante el juego. Unity tiene una ventana modo “Game” que permite una simulación de lo que vendría a ser la aplicación en marcha. Por ello se han realizado los testeos necesarios para el funcionamiento adecuado del primer nivel.

4.2 Parte 2 – Programación de Enemigos

Una vez se ha finalizado la construcción del primer nivel, que es a modo de tutorial, hay que pasar a la creación de los primeros enemigos, los más sencillos.

Durante esta parte se van a programar dos tipos diferentes de enemigos: enemigos de ataque a distancia y enemigos de ataque cuerpo a cuerpo. Ambos presentan programaciones completamente diferentes que deben de ser tratadas minuciosamente pues son la base para futuros enemigos más fuertes, los cuales su código estará basado en los códigos de estos dos primeros enemigos, uno de cada clase.

La duración prevista para esta parte se alarga hasta dos meses, exactamente el mes de Marzo y Abril del 2016. Al igual que en la parte anterior, a cada subtarea se le han asignado unos días o semanas en específico para una finalización a tiempo de cada una de las tareas correspondientes.

En esta parte se han realizado las siguientes subtareas:

- **Programación y creación de enemigo a distancia**: el primer enemigo a crear es el de ataque a distancia. Lo primero, al igual que con el Player, se han creado las animaciones correspondientes y una vez establecidas se ha pasado a la elaboración de sus códigos scripts que permiten el correcto funcionamiento de atacar con armas a distancia al personaje principal.

- **Montar niveles con enemigo a distancia**: una vez finalizado el enemigo a distancia, se ha procedido a la construcción de varios niveles en las que este enemigo aparece, dificultando poco a poco el transcurso por los niveles.

- **Programación y creación de enemigo cuerpo a cuerpo**: al igual que el enemigo de ataque a distancia, primero se ha pasado a la elaboración de las animaciones correspondientes para esta nueva clase de enemigos. Después se ha pasado a la construcción de los scripts necesarios para que estos personajes se muevan y vayan a atacar al personaje principal.

- **Montar niveles con enemigos de cuerpo a cuerpo**: con la incorporación de esta nueva clase de enemigos dentro del juego se ha pasado a la elaboración de nuevos niveles así como de la construcción de algún elemento o trampa nueva en los niveles para ir aumentado progresivamente la dificultad del juego. Si la dificultad no va aumentando conforme se avanza por el juego, al jugador puede llegar a resultarle aburrido.

4.3 Parte 3 – Creación de nuevos niveles

Al tener ya la forma de crear enemigos tanto de ataque a distancia como de cuerpo a cuerpo, mediante los recursos obtenidos previamente antes de comenzar con el proyecto, lo siguiente ya es ponerse a crear nuevos niveles con nuevos tipos de enemigos.

Además de crear los nuevos enemigos también es el crear nuevas trampas, nuevas tretas para dificultar el camino al jugador conforme avance por los niveles.

La duración de esta parte se ha extendido a dos meses, Mayo y Junio de 2016, debido a que es la creación de todos los niveles que van a conformar el videojuego además de la creación de todos los elementos que van a formar parte de él.

4.4 Parte 4 – Transformación a formato Android

Al crear todo lo necesario en la parte anterior, tenemos todo lo necesario para que el juego sea operativo para plataforma de escritorio, es decir, para ordenador. Pero el objetivo principal de este juego es llevarlo a plataforma Android y que acabe en el Play Store y sea accesible a todo el mundo.

Durante esta parte se va a cambiar el formato de Unity para que trabaje en plataforma Android. Esto conlleva el cambiar zonas de código dentro de los scripts debido a que hay que cambiarlo para que funcione en pantallas táctiles.

La duración para esta parte se ha establecido para la mayoría del mes de Julio.

En esta parte se han realizado las siguientes subtareas:

- **Introducir los botones de control del personaje**: tal y como se explica en la parte de Diseño, en Vista del juego, el juego de estar compuesto por los botones que van a controlar al player, que estarán situados en la zona inferior de la pantalla. Una vez introducidos hay que configurarlos para el correcto funcionamiento de estos.

- **Reprogramación del Player**: una vez tenemos los botones hay que cambiar el código del Player para que deje de funcionar mediante el teclado y pase a funcionar tras la pantalla táctil ya sea de móviles o tablets.

4.5 Parte 5 – Testeo y Solucionar Problemas

Una vez que el control por botones está configurado es hora de testear el juego al completo.

Durante esta parte, me he dedicado al testeo completo del juego, teniendo que jugar al videojuego para detectar posibles fallos que puedan haber surgido y arreglarlos.

Seguidamente, una vez comprobado que está todo correcto, lo último que queda es la redacción de este documento.

La duración prevista ha sido de un mes, de Agosto 2016.

En esta parte se han realizado las siguientes subtareas:

- Comprobar que todo el juego funciona correctamente y solucionar problemas: tal y como se ha comentado previamente, ha habido que testear al 100% todo el juego para comprobar el funcionamiento de este. Enseguida que se ha detectado un error, se ha arreglado y se ha tenido que crear de nuevo una apk del juego para volver a comprobar el funcionamiento del juego desde el principio.

- Redacción del Proyecto: una vez testado el juego, éste ya está finalizado, pero faltaba la redacción de este documento que explica todo el proyecto desarrollado.

A continuación se muestra una tabla explicativa con el reparto de tiempo de las tareas a realizar durante este proyecto:

Nombre de la tarea	Fecha de inicio	Fecha final
Sección 1 - Creación Primer Nivel		
Subtarea 1 - Programación básica del Player	02/02/16	10/02/16
Subtarea 2 - Montar los elementos que conforman el nivel	11/02/16	15/02/16
Subtarea 3 - Programación de los elementos y testeo del primer nivel	16/02/16	29/02/16
Sección 2 - Programación Enemigos		
Subtarea 1 - Programación y creación de enemigo a distancia	01/03/16	26/03/16
Subtarea 2 - Montar niveles con enemigos a distancia	27/03/16	02/04/16
Subtarea 3 - Programación y creación de enemigo a cuerpo a cuerpo	03/04/16	23/04/16
Subtarea 4 - Montar niveles con enemigos de cuerpo a cuerpo	24/04/16	30/04/16
Sección 3 - Creación de nuevos Niveles		
Subtarea 1 - Creación de todos los niveles que compondrán el juego. Además ir incorporando nuevos enemigos y nuevos elementos a los niveles	01/05/16	30/06/16
Sección 4 - Transformación a formato Android		
Subtarea 1 - Introducir los botones de control del personaje	01/07/16	09/07/16
Subtarea 2 - Reprogramación del Player	10/07/16	19/07/16
Sección 5 - Testeo y solucionar problemas		
Subtarea 1 - Comprobar que todo el juego funciona correctamente y solucionar errores	01/08/16	13/08/16
Subtarea 2 - Redacción del Proyecto	14/08/16	31/08/16

5 IMPLEMENTACIÓN

En este apartado se va a explicar la implementación de todo el proyecto así como de las partes y herramientas utilizadas en cada momento. No se va a explicar por orden de creación, sino por relevancia dentro del juego.

Dada la complejidad del proyecto se han obtenido más de 60 scripts para poder realizar todas las funcionalidades de este videojuego.

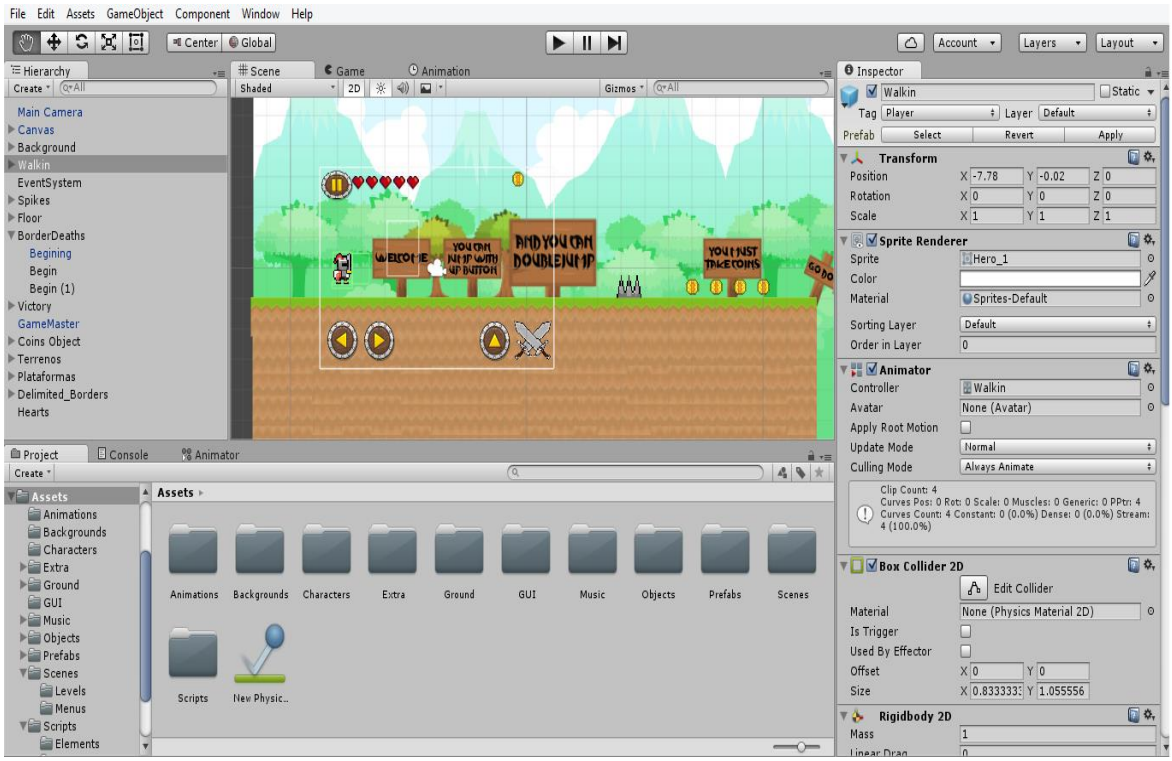
5.1 Entorno de Unity

En este apartado se va a hacer una pequeña introducción a cómo funciona el programa informático de Unity.

En la base de Unity se encuentran los *GameObjects*, la clase base para todas las entidades de una *escena* de Unity. Así, los personajes, la cámara, cualquier objeto del escenario y los elementos de los menús son *GameObjects*.

Una *escena* es simplemente una colección de objetos. Típicamente una *escena* representará un escenario independiente del juego, es decir, los niveles o los menús de este.

Esta es una captura de la pantalla principal de Unity:



5.2 El Player

Al primero que había que implementar dentro del juego es al mismísimo Walkin, el protagonista del juego.

Para comenzar, se deben de construir las animaciones que va a utilizar el personaje durante todo el juego. Esto se hace en la ventana de *Animation*, donde sprite por sprite se pone en un clip que generará la animación deseada.

Además, al personaje se le han añadido un par de *GameObjects* vacíos: uno llamado *groundCheck* y otro llamado *AttackTrigger*.

- ***groundCheck***: este objeto, junto a su script correspondiente sirve para comunicarle al juego si el personaje está tocando el suelo. Esto sirve principalmente para tener un control a la hora de realizar los saltos. Si el personaje está tocando el suelo, podrá saltar, si se encuentra en el aire, únicamente podrá volver a saltar una vez más para lograr un salto doble.

- ***AttackTrigger***: este objeto, junto a su script correspondiente, se activa cuando el personaje ataca. Es un objeto que cuando entra en contacto con un enemigo, detecta que es un enemigo por un Tag que tienen incorporados todos los enemigos y le causa daño al enemigo. Con esto es con lo que el personaje le quita vida a los enemigos hasta lograr eliminarlos.

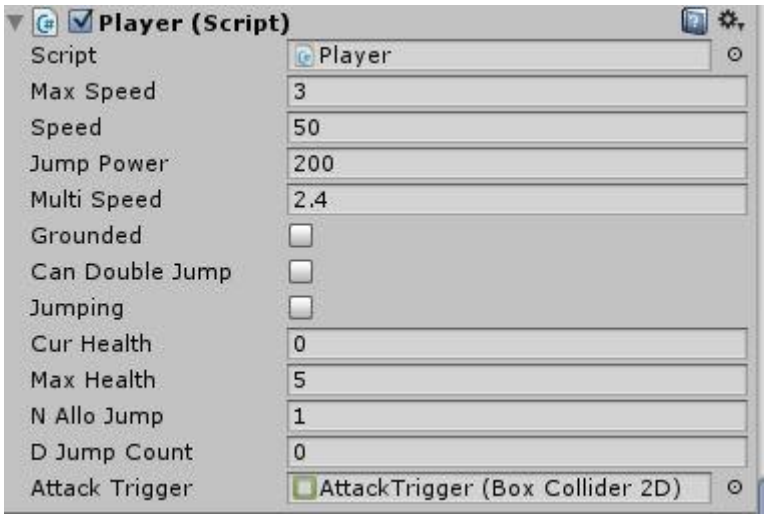
A parte, al objeto del personaje se ha puesto un Rigidbody2D, que esto lo que hace es ponerle masa al objeto, por lo que le afecta la gravedad. Esto sirve principalmente para que en los saltos, el personaje vuelva a caer al suelo.

Para que las animaciones funciones, se ha tenido que colocar el componente *Animator*, que es un gestor que controla todas las animaciones previamente realizadas en la ventana de *Animation* para que se activen cuando sean llamadas.

Además, hay un componente llamado Animation que lo que hace es que cuando el personaje sufra daño, lanzará una animación que volverá al personaje completamente rojo por apenas no llega a un segundo como señal a que ha sido dañado.

Luego, el objeto *Player*, está compuesto por uno de los scripts más importantes y más complejos, el script *Player*. También posee el script llamado *PlayerAttack*, que sirve para llamar a todo lo que hace referencia del jugador cuando realiza un ataque.

Con la siguiente captura de pantalla se explicará el script de Player:



Esta es la vista del script del Player dentro de Unity. Como se pueden observar hay varias variables que precisan ser explicadas para comprender el correcto funcionamiento del script.

- **Max Speed**: este es un valor que sirve para limitar la velocidad del player dentro del juego.

```
//Limitar la velocidad del Player
if (rb2d.velocity.x > maxSpeed)
{
    rb2d.velocity = new Vector2(maxSpeed, rb2d.velocity.y);
}

if (rb2d.velocity.x < -maxSpeed)
{
    rb2d.velocity = new Vector2(-maxSpeed, rb2d.velocity.y);
}
```

Rb2d es una forma de llamar al objeto del Player dentro del script.

Como se observa, tanto como si personaje va hacia la derecha (maxSpeed positiva) como si va a la izquierda (maxSpeed negativa), con la transformación del Vector2 verá reducida su velocidad para poder tener dentro del juego una velocidad apropiada de movimiento dentro del juego.

- **Speed y Multi Speed**: Estos valores se utilizan para que el personaje tenga movimiento.

```
if (moveLeft && !moveRight)
{
    rb2d.AddForce(Vector2.left * speed * multiSpeed);
    rb2d.transform.localScale = new Vector3(-1, 1, 1);
}

if (moveRight && !moveLeft)
{
    rb2d.AddForce(Vector2.right * speed * multiSpeed);
    rb2d.transform.localScale = new Vector3(1, 1, 1);
}
```

  Ambos valores sirven para multiplicar el Vector que los va a mover ya sea tanto como para la izquierda como para la derecha. La transformación del Vector3 sirve para que el personaje gire completamente el cuerpo y se encare en la dirección correspondiente.

- ***CurHealth*** y ***MaxHealth***: estos dos valores hacen referencia a la vida del personaje. *MaxHealth* es el valor de la máxima vida que tiene el personaje, y no puede ser superior en ningún momento. Por otro lado, *CurHealth*, hace referencia a la vida actual del personaje. Esta es la variable que irá cambiando dependiendo de lo que ocurra durante el juego.

- ***Grounded***, ***Can Double Jump***, ***Jumping***: estas variables son booleanas. Sirven para indicar en cada momento cuando el personaje se encuentra tocando el suelo, saltando o que puede saltar por segunda vez para realizar el salto doble.

  Si *Grounded* se encuentra activa, ni *Can Double Jump* ni *Jumping* podrán estarlo. Cuando este deje de estarlo, es decir, el personaje se encuentre saltando, las otras dos se activaran y cuando realice el segundo salto, *Can Double Jump* se desactivará y *Jumping* lo hará enseguida que el personaje toque el suelo.

- ***Jump Power***, ***N Allo Jump***, ***D Jump Count***: El primer valor hace referencia a la pontencia que gana el personaje al saltar, para que pueda ser impulsado hacia arriba.

  Las otras dos variables sirven controlar que únicamente el personaje pueda realizar un único doble salto y ninguno más. Sin esto, el jugador podría hacer que el personaje siguiera saltando llegando a provocar que el personaje se salga de la escena.

A continuación se muestra una captura de pantalla con la sección de código que muestra el comportamiento del Player cuando se aprieta el botón de saltar:

```
public void Jump()
{
    jumping = true;

    if (jumping)
    {
        if (grounded)
        {
            rb2d.AddForce(Vector2.up * jumpPower);
            canDoubleJump = true;
            grounded = false;
            dJumpCount = 0;
        }
        else
        {
            if (canDoubleJump && dJumpCount < nAlloJump) //Segundo salto
            {
                canDoubleJump = false;
                grounded = false;
                rb2d.velocity = new Vector2(rb2d.velocity.x, 0);
                rb2d.AddForce(Vector2.up * jumpPower);// / 1.25f);
                dJumpCount++;
            }
        }
    }

}
```

Dentro del código del script podemos encontrar también cómo funciona el player cuando sus vidas llegan a 0, es decir, que muere.

También encontramos la llamada a la animación *Player_Flash* que es la que se reproduce cuando el personaje sufre daño.

Dentro de los niveles, podemos encontrar tanto monedas como corazones. Cuando el personaje choca contra alguno de estos elementos ocurre algo: si es con monedas, incrementará en uno el número de monedas que posee y si toca un corazón, sanará una vida. Si posee todas las vidas, el corazón desaparecerá dado que no se puede superar el valor de *MaxHealth*.

Por último, el script está compuesto de todas las llamadas que ocurren cuando el personaje llega al final del nivel: un cartel que pone: *Victory*. Esta zona de código sirve para guardar los valores que se han obtenido dentro del nivel, guardándolas en sus respectivas variables, guardando también el juego y devolviendo al jugador a la escena de Selector de Niveles al cabo de unos pocos segundos.

Aquí tenemos una pequeña captura de código del primer nivel que representa esto último que se ha explicado acerca de lo que ocurre cuando el personaje llega al final del nivel:

```
public IEnumerator Victory_01(int seconds)
{
    yield return new WaitForSeconds(seconds);
    LevelManager.coinsL1 = GameMaster.coins;
    PlayerPrefs.SetInt("SavedCoinsL1", LevelManager.coinsL1);
    PlayerPrefs.Save();
    GameMaster.coins = 0;
    Application.LoadLevel("Menu"); //("Menu");
}
```

Esto es por lo que hace al script de Player, pero como se ha comentado previamente, el objeto Player posee dos scripts. El segundo script es el de *PlayerAttack*.

Aquí podemos observar que este script necesita tener controlado el *AttackTrigger*, que era el objeto que iba a dañar a los enemigos. Además tiene incorporado el sonido de *sword*, que hace referencia al sonido de una espada y la variable Volum con valor 1 indica que ese sonido de la espada sonará al máximo volumen, dado que 1 es el máximo valor.

Este script funciona de tal forma que cuando el jugador presiona el botón de las espadas, este script es llamado y ejecuta un ataque activando el *AttackTrigger* que infligirá daño al enemigo, además de la reproducción del sonido de la espada.

Estas son unas capturas del Player en cada una de las animaciones posibles que tiene:

Idle **Walking** **Jumping** **Attacking**

5.3 La Cámara

Todas las *escenas* de Unity poseen una cámara llamada *Main Camera* que se crea siempre automáticamente cuando se crea una nueva *escena*.

Estas cámaras son las responsables de proyectar lo que queremos que se vea en la pantalla del juego cada vez que es ejecutado.

El objetivo para este juego, como el personaje se va a mover a lo largo de todos los niveles, de izquierda a derecha, el objetivo de la cámara en todos los niveles es seguir al personaje, no perderlo de vista en ningún momento.

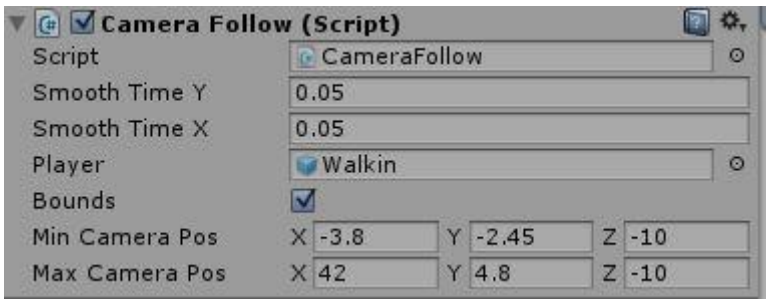
Para ello se ha programado un script que coge al personaje como referencia y se hace que siga todos sus movimientos.

Además, se ha incorporado una opción que proporciona una gran mejora para este estilo de juegos, los *bounds*, bordes, límites.

Estos límites sirven para que la cámara tenga unos puntos tanto del eje x como del eje y en los que no puede sobrepasarlos. Esto sirve para que siempre la camera se mantenga dentro del nivel y en ningún momento se salga.

Estos valores de límites son cambiables dependiendo del nivel, dado que cada nivel es un mundo nuevo y unos pueden ser más largos que otros.

Esta captura muestra el comportamiento del script. Los *Smooth Time* sirven para marcan una velocidad cuando la cámara se mueve, para mantenerla siempre nivelada y que no se mueva demasiado. Los valores que se observan abajo son para marcar los puntos tanto mínimos como máximos en los que se puede mover la cámara. Como se ha comentado, estos valores cambian dependiendo del nivel.



5.4 El Canvas

El Canvas es un objeto que aparece en el mundo del 2D. Es como un gran panel que se asocia con la cámara y con la pantalla de Game. Sirve principalmente para incorporar y coordinar los elementos que queremos que se vean en todo momento.

El Canvas en todos los niveles se divide en varios componentes: el botón de pausa, los elementos de la pausa, los corazones, el contador de monedas y los botones de control del personaje.

5.4.1 El Botón de Pausa y sus Elementos

Está situado en la esquina superior izquierda. Cuando el jugador aprieta dicho botón, Unity llama al script que tiene incorporado y realiza la siguiente acción:

- Lo primero es que detiene el tiempo de juego, paralizando todo lo que está ocurriendo dentro del juego, todo excepto la música.

- Nada más es llamado el script se activan cuatro botones que estaban ocultos, cada uno con una acción diferente:

    - **Resume**: Al presionar sobre este botón, el juego volverá a reanudará.

    - **Restart**: al presionar sobre este botón, la partida volverá a comenzar desde el principio, volviendo a poner el contador de monedas a 0.

    - **Main Menu**: al presionar sobre este botón, saldrá inmediatamente del nivel y devolverá al jugador a la pantalla de selector de nivel.

    - **Quit**: este botón detendrá por completo la aplicación, cerrándola.

5.4.2 Los Corazones

Al lado del botón de Pausa se encuentran los 5 corazones que posee el personaje dentro del nivel. Estos van desapareciendo y reapareciendo conforme avanza el jugador por el nivel.

Hay un script que los controla como un array dentro de la *Main Camera*, pero al estar asociada al Canvas, van a aparecer en cada momento.

5.4.3 Contador de Monedas

Este objeto interactúa con el *Canvas* y con un objeto que se titula *GameMaster*. Está situado en la esquina superior derecha.

Está compuesto por dos elementos: una imagen de las monedas, como indicación de que es el contador, y un texto.

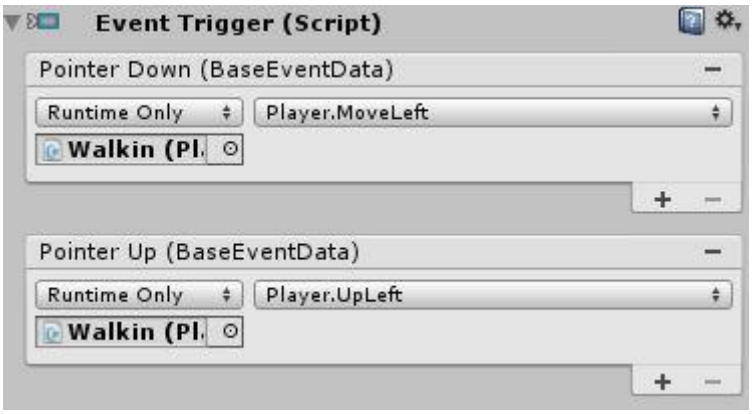
Este texto comienza siempre con el valor de 0 y conforme el jugador va recogiendo monedas este número se incrementa.

5.4.4 Botones de Control del Jugador

Son los objetos encargados del control del personaje dentro de los niveles. Cada uno es responsable de una acción para el personaje, y tal y como se ha comentado previamente en el apartado de Diseño, son cuatro botones: Flecha Izquierda, Flecha Derecha, Flecha Arriba y Espadas.

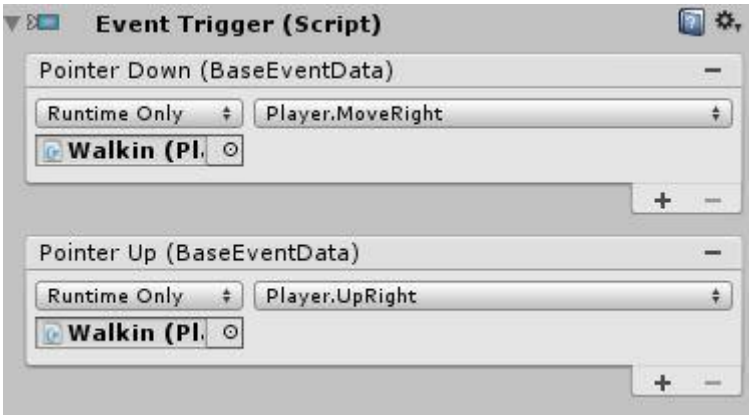
El comportamiento de cada botón llama a una parte en específico del script del Player.

- **Flecha Izquierda**: al tocar la pantalla el personaje se moverá hacia la izquierda. Si el contacto con la pantalla es permanente, que el dedo no se ha levantado, el personaje seguirá moviéndose. En el momento el dedo es levantado de la pantalla, el personaje deja de moverse.



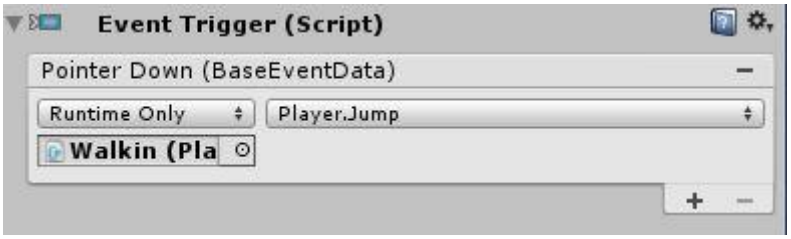
Tal y como se observa en la imagen, al poner el dedo sobre la pantalla será llamada la acción *MoveLeft* que hará mover al personaje. Cuando se levante el dedo, será llamada la acción *UpLeft*, que detendrá el movimiento del personaje en la dirección izquierda.

- **Flecha Derecha**: al tocar la pantalla el personaje se moverá hacia la derecha. Si el contacto con la pantalla es permanente, que el dedo no se ha levantado, el personaje seguirá moviéndose. En el momento el dedo es levantado de la pantalla, el personaje deja de moverse.



Tal y como se observa en la imagen, al poner el dedo sobre la pantalla será llamada la acción *MoveRight* que hará mover al personaje. Cuando se levante el dedo, será llamada la acción *UpRight*, que detendrá el movimiento del personaje en la dirección derecha.

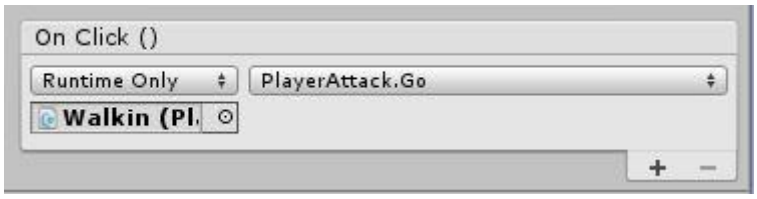
- **Flecha Arriba**: al tocar la pantalla el personaje realizará un salto. Si es apretado otra vez mientras el personaje está en el aire, volverá a realizar un segundo salto, más saltos no son posibles. Mantener el dedo sobre el botón no hará que el personaje este continuamente saltando.



Al tocar la pantalla se llama a la acción *Jump*, cuyo código ya ha sido previamente explicado en el apartado del *Player*.

- **Espadas**: al tocar este botón el personaje realizará un ataque. Esta acción en vez de ser llamada por el script del *Player*, corre a cargo del script *PlayerAttack*, tal y como se ha explicado antes. Mantener el dedo sobre el botón no hará que el personaje este continuamente atacando.

Hay que levantar completamente el dedo del botón para que el personaje realice la acción de atacar.



Tal y como se observa, en vez de llamar al script *Player*, llama a *PlayerAttack*.

5.4.5 Visualización del Canvas

A continuación se va a mostrar una captura de pantalla para una mejor visualización de todos los elementos explicados:

Como se observa el resultado es igual al esperado según el formato que deseaba en el apartado de Diseño.

5.5 Enemigos a Distancia

A continuación se va a proceder con la explicación de todos los enemigos que atacan al personaje principal con armas a distancia.

5.5.1 Arquero

El Arquero es el primer enemigo de todo el juego y va a servir como base para la creación de más enemigos de ataque a distancia. La base será la misma, lo que cambiará serán varios valores que harán más difíciles de eliminar a esta nueva clase de enemigos.

Al igual que como con el Player, lo primero a realizar son las animaciones en el *Animation*. Para este tipo de enemigos, únicamente nos hacen falta dos animaciones, cuando están quietos y cuando están disparando.

El objeto Arquero tiene tres sub-objetos que formaran parte de él:

- Uno es el *Collider* que servirá para darle un margen a su cuerpo, para que cuando colisione con el Player, este no lo atraviese.

- El segundo es el *Range* que sirve para marcar el rango de alerta del Arquero. En el momento el Player entra dentro de este *Range*, el Arquero ya puede pasar a dispararle.

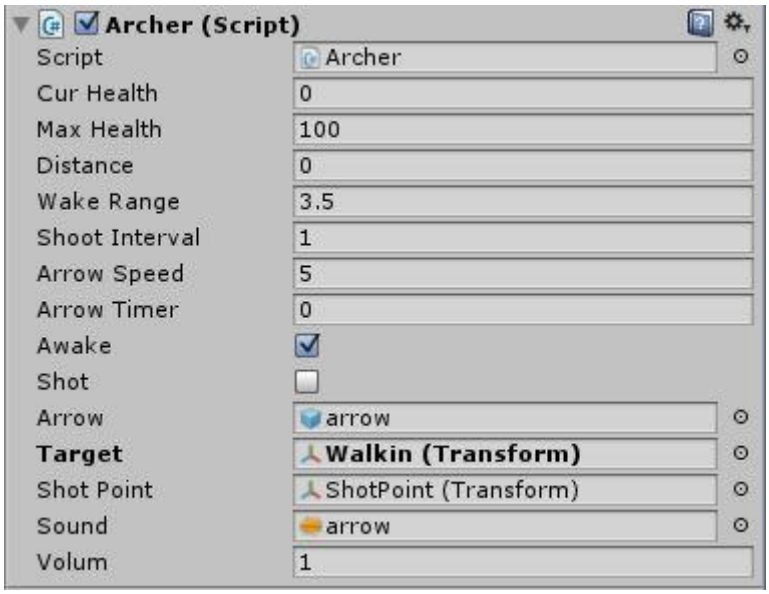
- El último es el *ShotPoint*. Este es un punto desde el cual se van a instanciar objetos Flecha, pues son estas que van a causar el daño a Player.

Tal y como posee el Player, a esta clase de objetos que tienen animaciones también hay que colocarles un componente *Animator* para que controle las animaciones y, también un componente *Animation* con la misma animación de poner al objeto de color rojo cuando este reciba daño.

Al igual que el Player, el Arquero posee más de un script. Por un lado tenemos el script *Archer*, que compone la inmensa mayoría del comportamiento del Arquero. Y por otro lado, dentro de *Range*, encontramos el otro script que posee: *AttackArrow*.

A continuación procederé a una pequeña explicación de ambos scripts.

Seguidamente podemos observar una captura de pantalla del motor de Unity acerca del script *Archer*:



Como podemos observar este script posee muchas variables y muchos elementos que serán explicados:

- **CurHealth** y **MaxHealth**: funcionan de la misma manera que con el Player, previamente explicada en el apartado del Player.

- **Distance**: este número indica la distancia a la que se encuentra el Player de él.

- **Wake Range**: cuando el *Distance* sea inferior a este número, el Arquero comenzará a disparar al jugador.

- **Shoot Interval**: indica el número de segundos entre disparo y disparo.

- **Arrow Speed**: indica la velocidad a la que irá la flecha hacia el jugador.

- **Arrow Timer**: es un cronometro del *Shoot Interval*.

- **Awake y Shot**: *Awake* llama a la animación de Idle y *Shot* a la de disparo. Además sirven para indicar que el objeto está realizando dicha acción.

El Arquero necesita tener una referencia del objeto que se va a instanciar, que este caso son los "*arrow*", que causan 1 de daño si impactan en el Player.

Al igual necesita tener una referencia a un target, un objetivo, que este caso será el *Player*. Lo mismo referenciar donde se encuentra el objeto *ShotPoint* y que cuando se dispare una flecha se reproduzca un sonido a Volumen 1 (máximo) de una flecha siendo disparada.

Aquí podemos ver una muestra de código de cómo se crean flechas para ser disparadas al Player:

```
GameObject arrowClone;
arrowClone = Instantiate(arrow, shotPoint.transform.position, shotPoint.transform.rotation) as GameObject;
arrowClone.GetComponent<Rigidbody2D>().velocity = direction * arrowSpeed;

arrowTimer = 0;

AudioSource.PlayClipAtPoint(sound, posicion.position, Volum);
```

El script *AttackArrow* se encuentra dentro del objeto *Range*. Este script simplemente es para llamar a la acción de Atacar cuando el Player entra dentro de este *Range*:

```
void OnTriggerStay2D(Collider2D col)
{
    if (col.CompareTag("Player"))
    {
        archer.Attack(true);
    }
}
```

- Matar a un Arquero va a otorgar 5 monedas al Player.

Aquí podemos ver unas capturas de las animaciones del Arquero:

Idle **Shot**

5.5.2 Mosquetero

Funciona exactamente que el Arquero, debido a que está creado a partir de él. Las pequeñas diferencias que podemos encontrar son las siguientes:

- **MaxHealth**: en vez de tener 100 como el Arquero, el Mosquetero tiene 200
- **Wake Range**: en vez de ser 3.5 como el Arquero, la del Mosquetero es de 4.5
- **Shot Interval**: en vez de ser 1'' como el Arquero, la del Mosquetero es de 3''

- **Bullet**: en vez de disparar Arrow que causan 1 de daño al Player, el Mosquetero dispara Bullets que si impactan, causarán 2 de daño al jugador.

- Matar a un Mosquetero va a otorgar 5 monedas al Player.

Aquí podemos ver unas capturas de las animaciones del Mosquetero:

Idle **Shot**

5.5.3 Pirata

Funciona exactamente que el Arquero y el Mosquetero, debido a que está creado a partir del Arquero. Las pequeñas diferencias que podemos encontrar son las siguientes:

- **MaxHealth**: en vez de tener 100 como el Arquero, el Pirata tiene 200

- **Wake Range**: en vez de ser 3.5 como el Arquero, la del Mosquetero es de 4.5

- **Shot Interval**: en vez de ser 1'' como el Arquero, la del Mosquetero es de 1.5''

- **Bullet Pirata**: en vez de disparar Arrow que causan 1 de daño al Player, el Pirata dispara unos Bullets que si impactan, causarán 1 de daño al jugador.

- Matar a un Pirata va a otorgar 15 monedas al Player.

Aquí podemos ver unas capturas de las animaciones del Pirata:

Idle **Shot**

5.6 Enemigos a Cuerpo a Cuerpo

A continuación se va a proceder con la explicación de los enemigos cuerpo a cuerpo que para que ataquen al Player se deben de encontrar a su lado.

5.6.1 Espadachín

El Espadachín es el primer enemigo de combate cuerpo a cuerpo de todo el juego y va a servir como base para la creación de más enemigos de combate cuerpo a cuerpo. La base será la misma, lo que cambiará serán varios valores que harán más difíciles de eliminar a esta nueva clase de enemigos.

Al igual que como con el Player, lo primero a realizar son las animaciones en el *Animation*. Para este tipo de enemigos, nos hacen falta tres animaciones, cuando están quietos, cuando están corriendo y cuando están atacando.

De forma muy parecida a los enemigos de ataque a distancia, también el objeto Espadachín tres sub-objetos que forman parte de él:

- Uno es el *Collider* que servirá para darle un margen a su cuerpo, para que cuando colisione con el Player, este no lo atraviese.

- El segundo es el *RangeFight* que sirve para marcar el rango de ataque del Espadachín. En el momento el Player entra dentro de este *RangeFight*, el Espadachín ya puede proceder a atacar.

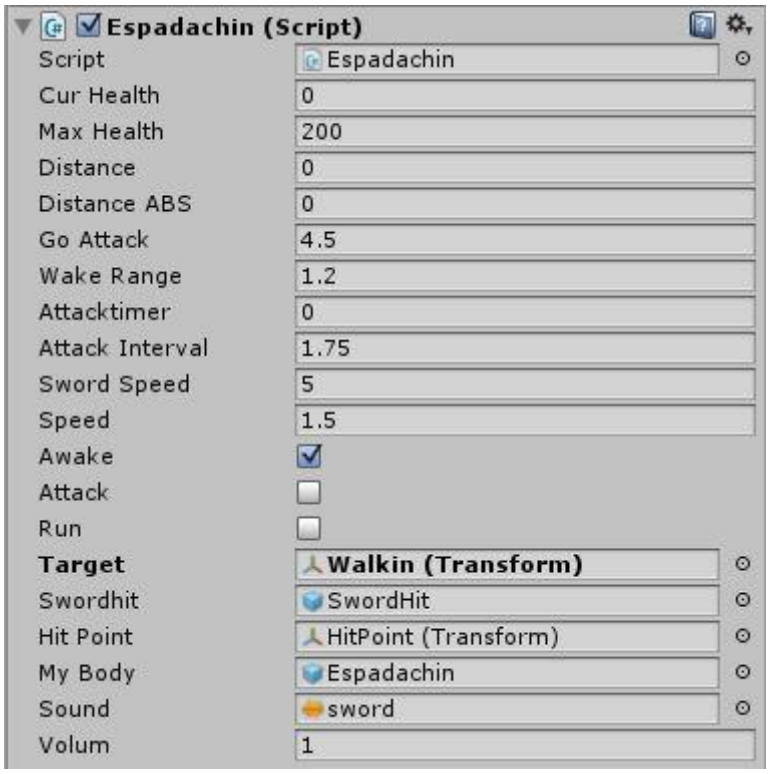
- El último es el *HiPoint*. Este es un punto desde el cual se van a instanciar objetos Sword, pues son objetos que van a causar el daño a Player.

Tal y como posee el Player, a esta clase de objetos que tienen animaciones también hay que colocarles un componente *Animator* para que controle las animaciones y, también un componente *Animation* con la misma animación de poner al objeto de color rojo cuando este reciba daño.

Al igual que el Player, el Espadachín posee más de un script. Por un lado tenemos el script *Espadachín*, que compone la inmensa mayoría del comportamiento del Espadachín. Y por otro lado, dentro de *RangeFight*, encontramos el otro script que posee: *EspadachinAttack*.

A continuación procederé a una pequeña explicación de ambos scripts.

Seguidamente podemos observar una captura de pantalla del motor de Unity acerca del script *Espadachín*:



Como podemos observar este script posee muchas variables y muchos elementos que serán explicados:

- **CurHealth** y **MaxHealth**: funcionan de la misma manera que con el Player, previamente explicada en el apartado del Player.

- **Distance**: este número indica la distancia a la que se encuentra el Player de él.

- **DistanceABS**: este número se asocia con Distancie y sirve para poner dicho número en valor absoluto. Esto sirve para que si el Player se encuentra detrás del enemigo, este se gire y vaya a por él.

- **GoAttack**: este valor sirve para que cuando el *Distance* sea inferior a este número, el Espadachín comenzará a correr hacia el Player.

- **Wake Range**: cuando el *Distance* sea inferior a este número, el Espadachín comenzará a atacar al jugador. Para que *WakeRange* este activo, lógicamente antes se ha activado el *GoAttack*.

- **AttackTimer**: es un cronometro del *Attack Interval*.

- **Attack Interval**: indica el número de segundos entre ataque y ataque.

- **Sword Speed**: indica la velocidad a la que atacar la espada al jugador.

- **Speed**: es la velocidad a la que va a correr el Espadachín hacia el Player.

- **Awake**, **Attack y Run**: *Awake* llama a la animación de Idle, *Run* llama a la animación de correr y *Attack* a la de atacar. Además sirven para indicar que el objeto está realizando dicha acción.

El Espadachín necesita tener una referencia del objeto que se va a instanciar, que este caso son los "*swordHit*", que causan 1 de daño si impactan en el Player.

Al igual necesita tener una referencia a un target, un objetivo, que este caso será el *Player*. Lo mismo referenciar donde se encuentra el objeto *HitPoint* y que cuando se ataque, una se reproduzca un sonido a Volumen 1 (máximo) de una espada siendo utilizada.

Aquí podemos ver el código de activación del Espadachín para cuando tiene que ir a atacar al Player:

```
void RangeGoAttack()
{
    if(distanceABS < goAttack && distanceABS > wakeRange)
    {
        run = true;
        if (run)
        {
            if(distance > 0)
            {
                myBody.transform.Translate(Vector2.left * Time.deltaTime * speed);
                myBody.transform.localScale = new Vector3(-1, 1, 1);
            }

            if (distance < 0)
            {
                myBody.transform.Translate(Vector2.right * Time.deltaTime * speed);
                myBody.transform.localScale = new Vector3(1, 1, 1);
            }
        }
    }
}
```

Dependiendo de si el jugador se encuentra a distancia negativa o positiva, el Espadachín se moverá a la izquierda o a la derecha según corresponda con la posición del Player.

El script *EspadachinAttack* se encuentra dentro del objeto *RangeFight*. Este script simplemente es para llamar a la acción de Atacar cuando el Player entra dentro de este *RangeFight*. El código funciona de la misma manera que como con el Arquero en *AttackArrow*, salvo que en vez de reconocer el objeto Arquero, reconoce al objeto Espadachín.

- Matar a un Espadachín va a otorgar 5 monedas al Player.

Aquí podemos ver unas capturas de las animaciones del Espadachín:

Idle **Run** **Attack**

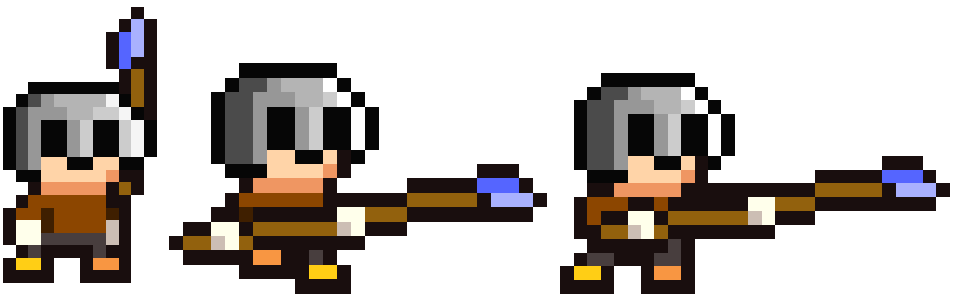
5.6.2 Piquero

Funciona exactamente que el Espadachín, debido a que está creado a partir de él. La diferenciación principal entre el Espadachín y el Piquero se basa en el sprite que los diferencia lógicamente y en las dos siguientes propiedades, por todo lo demás, son exactamente iguales.

- **GoAttack**: en vez de ser 4.5 como el Espadachín, la del Piquero es de 4.2
- **Attack Interval**: en vez de ser 1.75'' como el Espadachín, la del Piquero es de 1''
- Matar a un Piquero va a otorgar 5 monedas al Player.

Aquí podemos ver unas capturas de las animaciones del Piquero:

Idle **Run** **Attack**



5.6.3 Asesino

Funciona exactamente que el Espadachín y el Piquero. Las pequeñas diferencias que podemos encontrar son las siguientes:

- **KnifeHit**: el Asesino cuando impacta sobre el Player le causa 2 de daño
- **Attack Interval**: el del Asesino es de 1’’
- Matar a un Asesino va a otorgar 5 monedas al Player.

Aquí podemos ver unas capturas de las animaciones del Asesino:

Idle **Run** **Attack**

5.6.4 Ninja

Funciona exactamente que el Espadachín, el Piquero y el Asesino. Las pequeñas diferencias que podemos encontrar son las siguientes:

- **MaxHealth**: el Ninja posee 250 de vida, 50 más que los anteriores
- **KatanaHit**: el Asesino cuando impacta sobre el Player le causa 3 de daño
- **Attack Interval**: el del Ninja es de 1.25’’
- Matar a un Ninja va a otorgar 10 monedas al Player.

Aquí podemos ver unas capturas de las animaciones del Ninja:

Idle **Run** **Attack**

5.6.5 Vikingo

Funciona exactamente que el Espadachín, el Piquero, el Asesino y el Ninja. Las pequeñas diferencias que podemos encontrar son las siguientes:

- **MaxHealth**: el Vikingo posee 550 de vida, el que más vida de todo el juego
- **SwordHit**: el Vikingo causa 1 de daño, al igual que el Espadachín y el Piquero
- **Attack Interval**: el del Vikingo es de 2''

Matar a un Vikingo va a otorgar 15 monedas al Player.

Aquí podemos ver unas capturas de las animaciones del Vikingo:

5.7 Trampas y Elementos Dañinos

Además de los enemigos, mientras el jugador está jugando por un nivel, se va a encontrar varios elementos que pueden llegar a causarle daño e interponerse en su camino e incluso, llegar a eliminarlo.

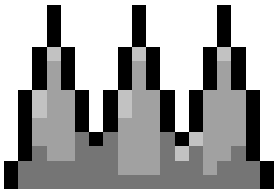
A continuación se van a explicar brevemente esta clase de objetos que pueden llegar a ser dañinos para el jugador.

5.7.1 Spikes

Son los elementos dañinos más comunes en todo el juego. Se basan en ser unos pinchos los cuales el jugador deberá de esquivarlos saltando por encima de ellos. Si por el contrario, cayera encima de estos, le causaran daño al Player.

Una vez el Player este situado encima de un spike, no seguirá recibiendo daño conforme vaya transcurriendo el tiempo. Únicamente será dañado una vez.

Aquí tenemos una pequeña visualización del aspecto que tienen los spikes:

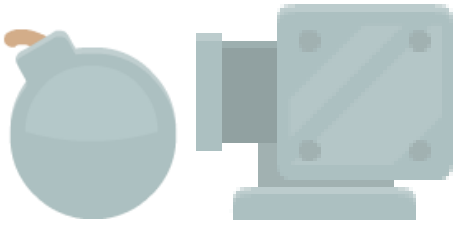


5.7.2 Cañones

Por otro lado, tenemos los Cañones. La forma de atacar es muy similar a la que tienen programa los enemigos de ataque a distancia dado que funciona de la misma manera: instancian un objeto que es el que causa daño, en este caso, una bomba, que si impacta sobre el Player, causará 1 de daño.

La diferencia es que los Cañones están en constante disparo: una vez empiece el nivel, los cañones disparan una bomba cada 5 segundos, así hasta que sean eliminados, pues eliminar un Cañón va a otorgar 5 monedas al Player.

Aquí tenemos una pequeña visualización del aspecto que tienen los cañones y las bombas:



5.7.3 Trampas para Esquivar

El juego se compone de tres mundos: uno de hierba, uno de desierto y otro de nieve.

Para cada uno de estos mundos hay unas trampas que van a estar en constante movimiento, de arriba hacia abajo y de abajo hacia arriba.

Cada objeto es independiente de otro igual, pues la configuración de la velocidad a la que van a moverse va a depender del nivel en cuestión. Al igual que cada uno se va a

mover hasta unos puntos completamente diferentes para así dificultar el trayecto del jugador dentro del nivel.

Estás trampas, si impactan con el jugador van a causarle 2 de daño.

Aquí tenemos una muestra del código que los tres objetos poseen:

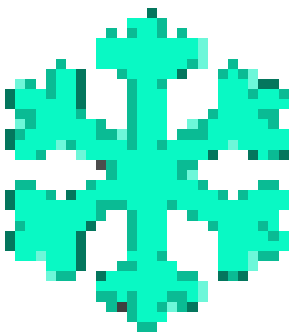
```
void Update ()
{
    if (dirUp)
    {
        transform.Translate(Vector2.up * speed * Time.deltaTime);
    }
    else
    {
        transform.Translate(-Vector2.up * speed * Time.deltaTime);
    }

    if(transform.position.y >= pos1)
    {
        dirUp = false;
    }
    if (transform.position.y <= pos2)
    {
        dirUp = true;
    }
}
```

Tal y como se observa, el objeto se moverá hacia arriba a una velocidad constante y cuando llegue al punto más alto, cambiará de dirección hacia abajo, y así constantemente.

A continuación una visualización de los tres objetos que van a estar asignados cada uno a un mundo en específico:

Mundo Hierba**: Fuego** Mundo Desierto**: Serpientes** Mundo Nieve**: Copos de Nieve**





*Cabe destacar que el objeto que va a contener a los **Copos de Nieve**, está compuesto por muchos de ellos.

5.7.4 Trampas que se caen

Las siguientes trampas se tratan en ser plataformas que sirven para el jugador avance por el nivel, pero con la contra de que en el momento el jugador entra en contacto con estas plataformas al cabo de un tiempo, caen al vacío y si el jugador no ha salto, este cae con la plataforma, perdiendo todas las vidas.

Al igual que las trampas para esquivar, hay un tipo asignado para cada mundo del juego.

Dependiendo del mundo en el que este el jugador, el tiempo que la plataforma va a poder aguantar, se disminuye:

- **Mundo Hierba**: el tiempo es de 0.8''
- **Mundo Desierto**: el tiempo es de 0.7''
- **Mundo Nieve**: el tiempo es de 0.6''

Esto es para aumentar la dificultad del juego conforme se va avanza por el juego.

A continuación una visualización de los tres objetos que van a estar asignados cada uno a un mundo en específico:

Mundo Hierba

Mundo Desierto

Mundo Nieve

5.8 Elementos de los Niveles

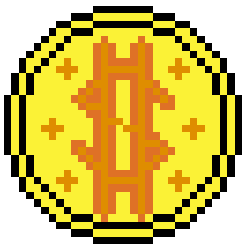
Al igual que por los niveles hay objetos que sirven para intentar acabar con el jugador, hay objetos que están para ayudarlo o simplemente obstaculizarlo pero sin causarle daño.

5.8.1 Monedas

Hay montones de ellas repartidas por todos los niveles. Son esenciales para el transcurso del videojuego, pues se necesita recoger al 100% el número de monedas que posee el nivel para poder desbloquear el nivel siguiente.

Tal y como se ha comentado antes, matar enemigos o cañones va a otorgar monedas al jugador, pero todas las monedas que se encuentre el Player por el nivel le va a otorgar 1 más por cada moneda recogida.

Aquí tenemos una visualización del aspecto que poseen las monedas del juego:

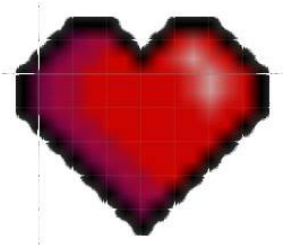


5.8.2 Corazones

Al igual que hay monedas, hay corazones repartidos por los niveles que sirven para que el jugador recupere vidas perdidas.

Tal y como se ha comentado, no se pueden tener más de 5 vidas, por lo que si se recogen corazones teniendo el máximo de vida, estos desaparecerán y no causaran ningún efecto.

Aquí tenemos una visualización del aspecto que poseen los corazones en el juego:



5.8.3 Elevadores

Esta clase de objetos sirven a modo de ascensor para que el jugador pueda llegar a zonas que se encuentran fuera de su alcance.

Funciona de forma similar a las trampas para esquivar, pero sin recibir daño. El objeto se mueve hasta un punto y cuando lo alcanza, se mueve en la dirección contraria y así constantemente durante todo el transcurso del nivel.

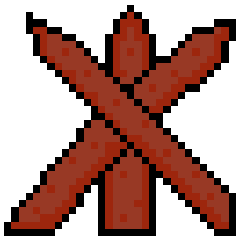
Aquí tenemos una visualización del aspecto que poseen los elevadores en el juego:

5.8.4 Pinchos

Estos objetos simplemente están en el terreno de los niveles para molestar al jugador y que el trayecto no sea todo el rato caminando.

Son objetos que no dañan al jugador. Únicamente deberá de saltarlos y continuar.

Aquí tenemos una visualización del aspecto que poseen los pinchos en el juego:



5.9 Edificios

Para adornar los niveles se han creado edificios. Estos están compuestos de muchos objetos que les dan forma.

El jugador puede caminar por encima de ellos en la mayoría de ellos sin problema.

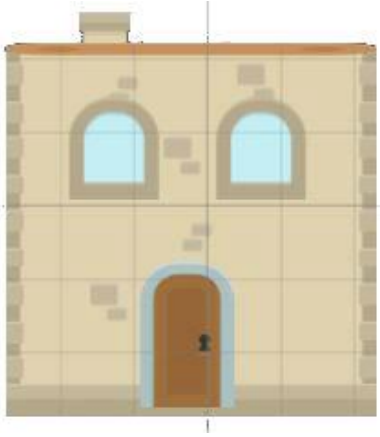
Podemos encontrar varios tipos: edificios, torres y puentes.

5.9.1 Trampas para Esquivar

Dentro del juego encontramos hasta 5 tipos de edificios: 2 tipos de casas, 2 tipos de tabernas y un edificio de barracones.

Aquí tenemos una visualización del aspecto que poseen estos edificios dentro del juego:

Casa 1



Casa 2

Taberna 1

Taberna 2

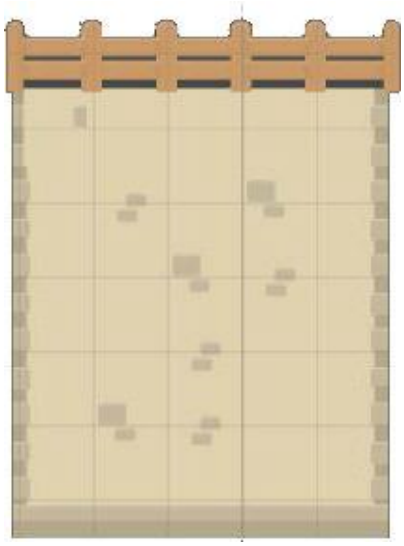
Barracones

5.9.2 Torres

Son los edificios principales a los que los elevadores van a llevar al jugador.

La parte superior es por donde el jugador puede andar. Normalmente esa parte estará llena de monedas o de enemigos por lo que el jugador deberá de estar atento para cuando deba de subir a una torre.

Aquí tenemos una visualización del aspecto que poseen las torres dentro del juego:



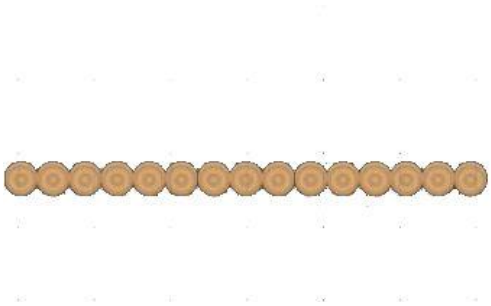
5.9.3 Puentes

Estos sirven para conectar terrenos o torres dentro de los niveles y podemos encontrar de dos tipos: puente de cadenas o sin cadenas:

Aquí tenemos una visualización del aspecto que poseen los puentes dentro del juego:

Puente con cadenas

Puente sin cadenas



5.10 Terrenos

Estos los objetos por los cuales el personaje va a estar en contacte contacto. Representan el suelo de todos los niveles y, tal y como se ha comentado, existen tres mundos, por lo que el terreno va cambiando dependiendo del mundo en el que se encuentre el jugador.

Para formar un sólido terreno hace falta la unión de muchísimos objetos que se han agrupado para lograr un resultado convincente dentro de los niveles.

A continuación se van a mostrar los elementos básicos, mundo por mundo, con los que se han montado todos los terrenos de todos los niveles.

En todos los mundos vamos a poder encontrar 5 tipos de elementos: el terreno central, el terreno superior, el terreno izquierda, el terreno derecha y la plataforma.

El jugador puede andar por todos menos por el terreno central, dado que estos son mera decoración dentro de los niveles.

5.10.1 Mundo Hierba

- **Hierba Superior**: es la parte superior de los terrenos en el mundo hierba. Son donde principalmente va a estar en contacto el jugador cuando este "Grounded".

- **Hierba Central**: es mera decoración. Se sitúan todos por debajo de los objetos de hierba superior.

- **Hierba Izquierda**: suelen utilizarse para elevar zonas de terreno. Son una pequeña rampa. Se complementan con más objetos como hierbas superiores y hierbas derechas para formas terrenos sobresalientes.

- **Hierba Derecha**: suelen utilizarse para elevar zonas de terreno. Son una pequeña rampa. Se complementan con más objetos como hierbas superiores y hierbas izquierdas para formas terrenos sobresalientes.

- **Plataforma Hierba**: están puestas como terrenos que se encuentran flotando en el aire. Sirven para que el jugador se desplace por el nivel.

5.10.2 Mundo Desierto

- **Arena Superior**: es la parte superior de los terrenos en el mundo desierto. Son donde principalmente va a estar en contacto el jugador cuando este "Grounded".

- **Arena Central**: es mera decoración. Se sitúan todos por debajo de los objetos de arena superior.

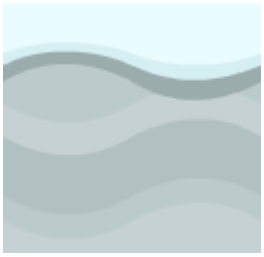
- **Arena Izquierda**: suelen utilizarse para elevar zonas de terreno. Son una pequeña rampa. Se complementan con más objetos como arenas superiores y arenas derechas para formas terrenos sobresalientes.

- **Arena Derecha**: suelen utilizarse para elevar zonas de terreno. Son una pequeña rampa. Se complementan con más objetos como arenas superiores y arenas izquierdas para formas terrenos sobresalientes.

- **Plataforma Arena**: están puestas como terrenos que se encuentran flotando en el aire. Sirven para que el jugador se desplace por el nivel.

5.10.3 Mundo Nieve

- **Nieve Superior**: es la parte superior de los terrenos en el mundo nieve. Son donde principalmente va a estar en contacto el jugador cuando este “Grounded”.



- **Nieve Central**: es mera decoración. Se sitúan todos por debajo de los objetos de arena superior.

- **Nieve Izquierda**: suelen utilizarse para elevar zonas de terreno. Son una pequeña rampa. Se complementan con más objetos como nieves superiores y nieves derechas para formas terrenos sobresalientes.

- **Nieve Derecha**: suelen utilizarse para elevar zonas de terreno. Son una pequeña rampa. Se complementan con más objetos como arenas superiores y arenas izquierdas para formas terrenos sobresalientes.

- **Plataforma Nieve**: están puestas como terrenos que se encuentran flotando en el aire. Sirven para que el jugador se desplace por el nivel.

5.11 Objetos Decorativos

Para que los niveles queden más atractivos para los jugadores, estos están repletos de pequeños objetos decorativos así como: árboles, arbustos, rocas, setas, cactus, muñecos de nieve…conforme al mundo del nivel correspondiente.

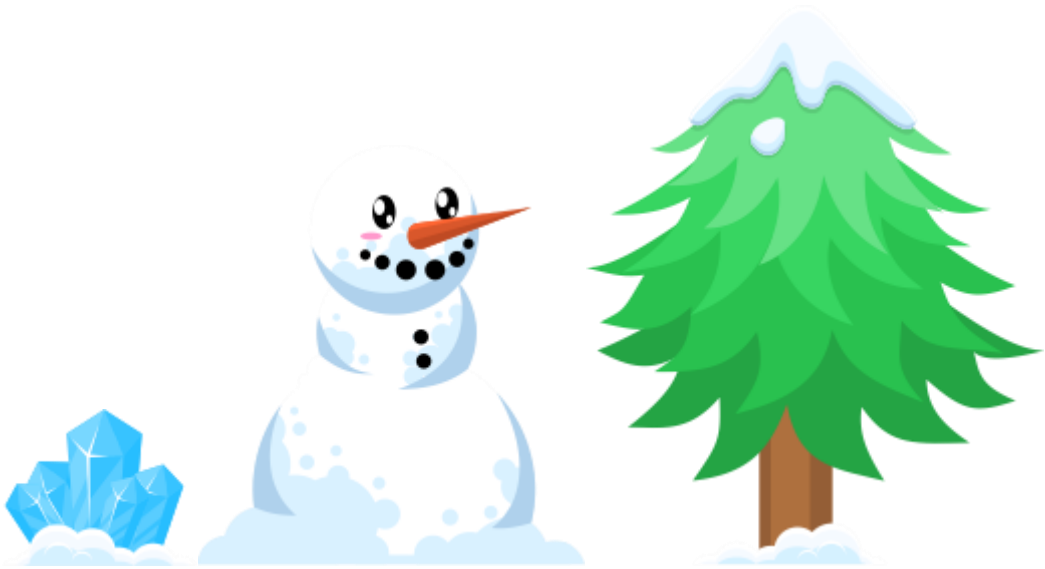
Con estos objetos, el Player no puede interactuar dentro del juego. Simplemente son meros objetos decorativos para atraer visualmente.

Aquí podemos ver una pequeña recopilación de estos objetos:

- **Mundo Hierba**

- **Mundo Desierto**

- **Mundo Nieve**



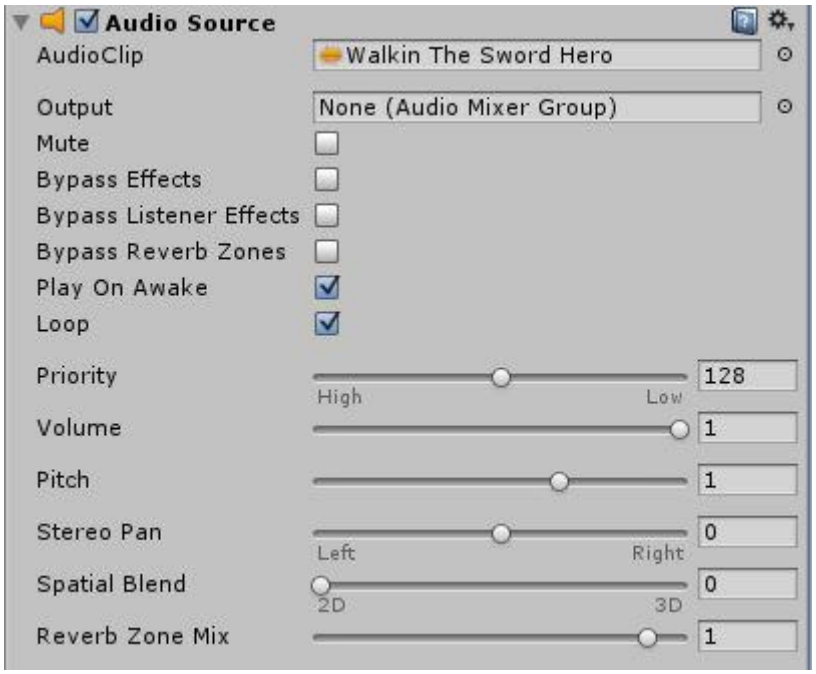
5.12 Música

Todo el videojuego está compuesto por varias canciones con un toque de 8 bits, gracias a la ayuda del programa GX SCC comentado previamente.

El juego consta de 7 canciones en total: una canción para el Menú principal y para los Selectores de Niveles y luego hay 6 canciones para los niveles en sí.

Como el juego se compone de 24 niveles, cada 4 niveles suena una canción diferente, que es lo mismo, que cada pantalla de Selector de Niveles.

El componente de sonido dentro de Unity está colocado dentro de la Main Camera.



Es un componente *Audio Source* en los que hay que colocar en la sección de *AudioClip* la canción correspondiente a la *escena* en la que estemos.

- **Play On Awake**: sirve para que, nada más se inicialice la escena, comience a reproducir la canción

- **Loop**: lo que hace esta opción en poner en bucle la canción, de tal forma que si la canción terminase de reproducirse dentro del nivel, volvería a comenzar de nuevo y así, sucesivamente hasta completar el nivel.

Estas dos opciones están seleccionadas en todas las *escenas* de todo el videojuego.

5.13 Menú

Es la primera escena de todo el videojuego. Tal y como se diseñó en la parte de Diseño, el resultado final ha sido el esperado:

- Encontramos el cartel con el título del videojuego en grande y que llame a la atención, ocupando gran parte de la escena.

- Debajo del cartel, se sitúan los tres botones con los que va a poder interactuar el jugador:

    - **New Game**: al darle comenzará el juego desde el principio eliminando todos los datos borrados de partidas anteriores.

- **Load Game**: ejecutará el juego con los datos guardados de anteriores partidas dejando al jugador seguir por donde se quedó antes de salir del juego.

- **Exit Game**: la aplicación se cerrará y dejara de ejecutarse.

*Es la misma explicación del apartado de Diseño: Menú

Aquí se puede observar una visualización del resultado final del Menú:

5.14 Selector de Niveles

Cuando el jugador selecciona en la pantalla de Menú tanto el botón de New Game o el de Load Game es enviado a las escenas de Selectores de Niveles.

Si se dio a New Game, todos los datos guardados serán eliminados y se resetearán todos los datos del juego. Si por el contrario se dio a Load Game, la escena aparecerá tal y como el jugador la dejo antes de salir del juego.

En las pantallas de selectores de niveles podemos encontrar varios objetos de interacción para los jugadores:

*El resultado final es igual al diseñado en el apartado de Diseño.

- **Botones de Selector de Nivel**: son los objetos grandes con el número en su interior. El número indica el nivel del juego. Cuando se tocan estos botones, cargan el nivel correspondiente y envían al jugador a jugar dicho nivel.

- **Botones de Transición de Niveles**: son los objetos con una flecha en su interior. Sirven para avanzar o retroceder entre pantallas de selectores de niveles, pues como se observa, solo se muestran 4 niveles por pantalla y el juego posee 24 en total.

- **Botón de Salida**: es el botón que se encuentra en la parte inferior que posee una puerta en su interior. Al darle, el juego se cerrará, abandonando la partida.

- **Tablas de Estrellas**: debajo de cada nivel se puede observar que hay placas en las que hay estrellas. Dependiendo del número de monedas que haya recogido el jugador dentro del nivel, aparecerán una, dos o tres estrellas. Es esencial recoger el 100% de monedas de un nivel, pues es la única forma de desbloquear el nivel siguiente.

Para una mejor comprensión del funcionamiento de las estrellas de cada nivel, se va a proceder una breve explicación del código, en este caso, del nivel 5 del juego:

```
/// LEVEL 5 ///
if (LevelManager.coinsL5 == 0)
{
    Star1L5.SetActive(false);
    Star2L5.SetActive(false);
    Star3L5.SetActive(false);
}
if (LevelManager.coinsL5 >= 1 && LevelManager.coinsL5 <= 59)
{
    LevelManager.nivel5_estrellas++;
    Star1L5.SetActive(true);
    Star2L5.SetActive(false);
    Star3L5.SetActive(false);
}
if (LevelManager.coinsL5 >= 60 && LevelManager.coinsL5 <= 100)
{
    LevelManager.nivel5_estrellas += 2;
    Star1L5.SetActive(true);
    Star2L5.SetActive(true);
    Star3L5.SetActive(false);
}
if (LevelManager.coinsL5 == 101)
{
    LevelManager.nivel5_estrellas += 3;
    Star1L5.SetActive(true);
    Star2L5.SetActive(true);
    Star3L5.SetActive(true);
    Mission_6.SetActive(true);
}
```

Este código es una pequeña zona del script del menú_2 que contiene al nivel 5. En este código se hace referencia a un script denominado *LevelManager*, que es un script que contiene todas las variables estáticas del juego. Gracias a ellas se va a poder tener un control del juego y de las partidas guardadas.

Como se observa, tenemos cuatro posibilidades: que no aparezca ninguna estrella, una, dos o tres. Todo se define mediante las monedas recogidas. Cada nivel posee su rango específico para cada estrella, pues cada nivel posee un número total de estrellas diferente.

Se marcan los cuatro rangos de aparición de estrellas, y en el último, el de tres estrellas, se activa el siguiente nivel, en este caso, el nivel 6.

Cuando se devuelva a la pantalla de selector de niveles, viniendo desde un nivel que se ha completado al 100%, el siguiente nivel aparecerá según al primer caso del código, aparecerá una tabla completamente vacía pues el valor de las monedas aún es 0.

5.15 Menú de Pausa

El menú de pausa se activa cuando desde dentro de un nivel, el jugador toca el botón situado en la esquina superior izquierda.

Cuando el jugador toca dicho botón, el juego se detiene mostrando cuatro botones con las siguientes funciones, las mimas descritas en el apartado de Diseño: menú de pausa.

- **Resume**: al darle inmediatamente el juego se reanudará y podrá seguir jugando
- **Restart**: al darle, el nivel volverá a empezar desde el principio, se reseteará
- **Main Menu**: al darle saldrá del nivel en cuestión volviendo a la vista del selector de niveles
- **Quit**: detendrá la aplicación por completo, cerrándola y saliendo del juego

Cuando el botón de pausa es tocado, desaparece hasta que juego vuelva a ser de nuevo reanudado.

Aquí podemos contemplar una visualización del resultado de la pantalla de menú de pausa dentro del juego:

6 RESULTADOS

Con toda la implementación realizada, el juego está finalizado.

Posteriormente se van a exponer unas capturas de pantalla, una por mundo, para una mejor visualización de la implementación.

También se va a tratar el tema de la comercialización, de cómo el proyecto se ha acabado subiendo a la plataforma Play Store, para que así todo el mundo pueda descargarse el juego y disfrutar de un rato de entretenimiento, además del estudio de mercado realizado para ver a qué público está dirigido este videojuego.

6.1 Capturas de pantalla

A continuación se van a presentar varias capturas de pantalla del juego, a poder ser una captura por mundo, para una mejor visualización de estos.

En estas capturas se van a poder observar los estados de los objetos, los personajes…

- **MUNDO HIERBA**

- **MUNDO DESIERTO**

- **MUNDO NIEVE**

6.2 Comercialización y Mercado: Play Store

El objetivo principal del videojuego es subirlo a la plataforma de Play Store.

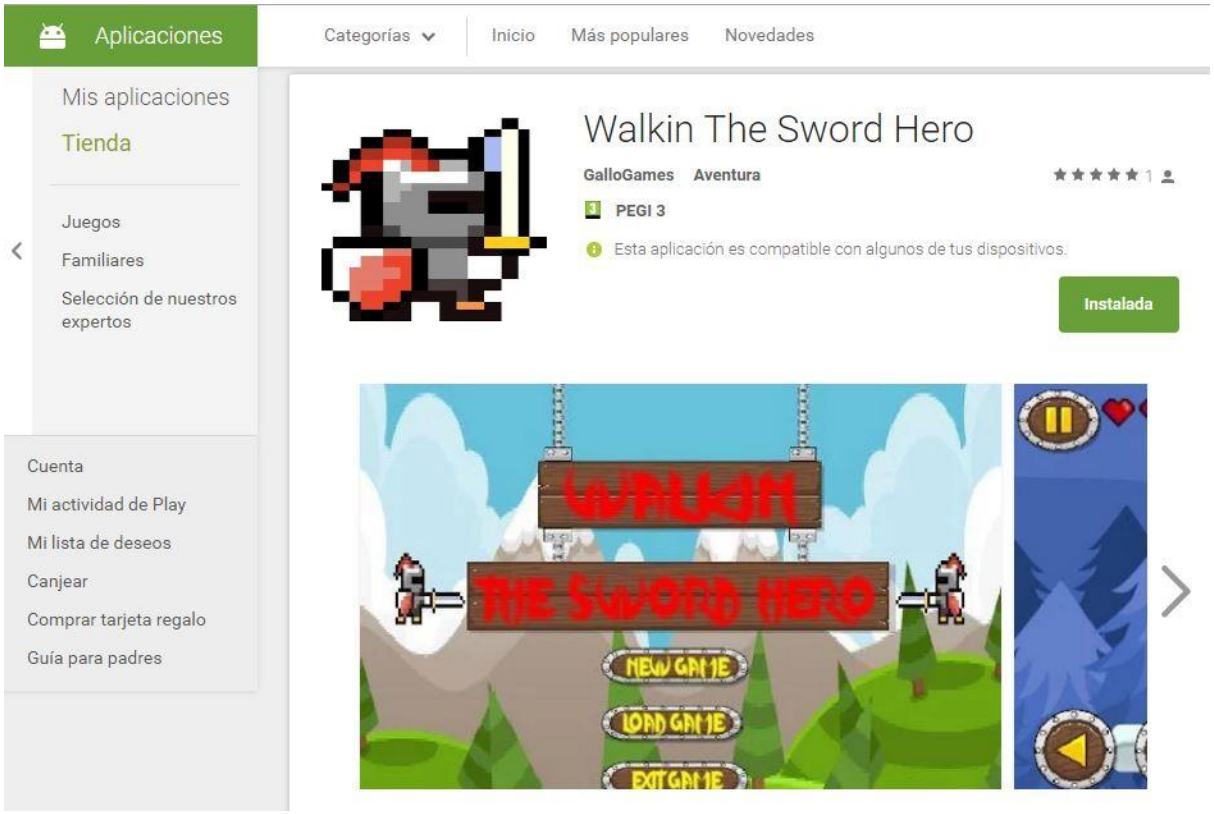
Para poder subir una aplicación o un juego a Play Store hay que crearse una cuenta de Desarrollador en la que hay que pagar 25$.

Una vez registrado hay que seguir toda una serie de pasos específicos para que la aplicación pueda ser subida con éxito.

El juego se ha catalogado como juego de aventuras y apto para todos. La clasificación del contenido del juego, Google lo ha acabado catalogando de la siguiente forma:

Todas estas imágenes representan que en todo el mundo, este juego es apto para todo el público.

Aquí podemos ver una visualización del resultado final de la aplicación subida a Play Store:



7 TRABAJO FUTURO

Los posibles trabajos futuros para la aplicación son los siguientes:

- Ampliación de nuevos mundos, lo que implica la creación de nuevos niveles y de nuevos enemigos

- Incorporación de las herramientas de Google Play Services: esto sería vincular el juego a cuentas de redes sociales como Facebook o Twitter, dado que el juego está teniendo éxito entre los jugadores.

8 AGRADECIMIENTOS

Principalmente a Jordi Joan Linares Pellicer porque gracias a los conocimientos adquiridos durante el curso de Diploma de Extensión Universitaria en Creación de Videojuegos he podido realizar este proyecto, además de los conocimientos adquiridos durante mi transcurso como alumno Erasmus.

A todos los testeadores del juego que me han ayudado a encontrar fallos e incorporar nuevos elementos que ayuden al jugador durante el transcurso del videojuego.

Gracias a las páginas web de OpenGameArt y GameArt2D, dado que de ellas he logrado sacar la inmensa mayoría del material gráfico del videojuego.

También agradecer a todo el foro de Unity, a toda esa gente que ayuda a todos los que con este gran programa en algún momento encontramos algún contratiempo o error.

9 BIBLIOGRAFÍA

Open Game Art: personajes del videojuego
http://opengameart.org/

Game Art 2D: terrenos y botones del videojuego
http://www.gameart2d.com/