
**Universidad de Valladolid**

**ESCUELA DE INGENIERÍA INFORMÁTICA (SEGOVIA)**

**Grado en Ingeniería Informática de Servicios y Aplicaciones**

# Desarrollo de un videojuego en 2D para Android con Unity 3D

***Alumno*: Joaquín Casas Albertos**

**Tutor: Luis María Fuentes García**


# DESARROLLO DE UN VIDEOJUEGO EN 2D PARA ANDROID CON UNITY 3D

Trabajo de Fin de Grado
Ingeniería Informática de Servicios y Aplicaciones
Julio 2015

*Autor*: Joaquín Casas Albertos
Tutor: Luis María Fuentes García

# ÍNDICE

# ÍNDICE DE TABLAS

# ÍNDICE DE FIGURAS



# ÍNDICE DE DIAGRAMAS

# Resumen

Este proyecto consiste en desarrollar un videojuego del género plataformas desde cero para dispositivos Android, que disponga de distintos niveles que presenten retos a superar. Para ello se utilizará el motor gráfico Unity3D, uno de los más utilizados a la hora del desarrollo de videojuegos, debido a su gran potencial.

El jugador controlará a un lémur y le ayudará a recuperar los mangos que le han sido robados a través de los niveles que componen el juego, derrotando a los enemigos que encuentre en su camino.

**Palabras clave**: videojuego, 2D, Unity3D, Android, dispositivo móvil.

# Abstract

This Project is about developing from scratch a platforms videogame for Android devices, who has different levels that present challenges to overcome. To do it, I have used Unity3D graphics engine, one of the most used in game development, due to it has a great potential.

The player controls a lemur, helping him to recover the Mangos that have been stolen, through the levels that make up the game, defeating the enemies on his way.

**Keywords**: videogame, 2D, Unity3D, Android, mobile device.

# Capítulo 1
# Introducción

Este documento contiene lo relativo al proyecto "**CocoPunch: desarrollo de un videojuego en 2D para Android con Unity3D**", basado en la creación de un videojuego para dispositivos Android en formato 2D con la ayuda del motor gráfico Unity3D.

El juego es del género Plataformas, caracterizado por basarse en el desplazamiento horizontal y vertical sobre una serie de plataformas con enemigos, a la vez que se recogen una serie de objetos que permiten completar el juego.

En este TFG se implementará un videojuego en 2D que permitirá disfrutar de distintos niveles en los que se deberá recoger Mangos, derrotar a los enemigos que estén en el camino, y llegar al final de cada nivel. Para el desarrollo del juego se utilizará la plataforma Unity3D, que dispone de la opción de desarrollo en 2D.

Para el desarrollo de un videojuego, es necesaria la participación de diseñadores, programadores, músicos, etc.. En este caso, al no formar parte de ningún equipo de trabajo, todo lo realizaré yo.

En este capítulo se presenta una breve introducción del problema a tratar y cuál es la solución. La motivación que lleva a la elección de este tema y el por qué es interesante. Los objetivos y sub-objetivos planteados. Las herramientas utilizadas para la solución del problema. La organización del documento.

## 1.1. Motivación

Los videojuegos llevan con nosotros desde hace décadas, y siempre han sido utilizados tanto por jóvenes como por mayores. Ya sea en videoconsolas portátiles, no portátiles o dispositivos móviles, siempre han estado y siguen estando en nuestras vidas.

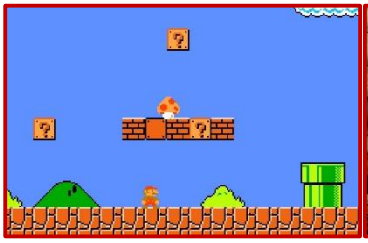
El mundo de los videojuegos sigue con su imparable avance, y, actualmente, es una de las mayores industrias del mundo, tanto en cuanto al dinero que maneja como por la cantidad de personas que hacen uso de sus productos.

Con la aparición de los juegos *independientes*, el mercado ha crecido exponencialmente, debido a que ya no es necesario el disponer de un gran equipo ni un gran presupuesto para desarrollar un videojuego. Este tipo de juegos predominan en los *smartphones* por la cantidad de documentación y plataformas que permiten su desarrollo, y la posibilidad de cualquier persona de poseer un dispositivo móvil de estas características en la actualidad. Esto permite que cada cierto tiempo aparezca un videojuego que revolucione las redes sociales y el mundo, como el *Flappy Bird*, videojuego en 2D para Android. Con ejemplos como este se puede ver el potencial e importancia que tienen en la actualidad los videojuegos para dispositivos móviles.

Flappy Bird

El desarrollo de videojuegos ha aumentado, pero se ha perdido la esencia de la aventura y de enfrentarse a retos en distintos niveles, llegando a la monotonía. Actualmente la mayoría son juegos a los que juegas dos veces, y lo desinstalas y buscas otro porque no te presenta ningún reto.

Esta situación se puede comprobar en los juegos denominados *EndlessRun*[1], que tratan sobre un único nivel en el que no controlamos el movimiento del personaje, sino que corre hacia delante, teniendo que controlar acciones como saltar o golpear para seguir jugando. Pero el haber tantos juegos de este tipo, y el ser muy repetitivos, hace que no tardes mucho en cambiar de entretenimiento. Por ello he pensado sea una gran idea desarrollar un videojuego que recupere los juegos míticos como Super Mario Bros o Crash Bandicoot de las consolas que tanto entretenimiento nos dieron, que permitían controlar a un personaje por distintos niveles, recogiendo objetos y con un objetivo definido.



Super Mario Bros


Crash Bandicoot

[1] http://appadvice.com/appguides/show/endless-running

## 1.2. Objetivos

El objetivo principal del TFG es el desarrollo de la "demo" de un videojuego en 2D con Unity 3D, construyendo la estructura de un videojuego, sus componentes y características que permitan la creación de múltiples niveles diferentes. Para conseguir este objetivo, deben cumplirse sub-objetivos:

- **Diseño del videojuego**: personaje principal del juego, enemigos, objetos a recoger y escenario por el que mover al personaje.
- **Creación de una Interfaz de Usuario Gráfica** sencilla y llamativa.
- **Creación del sonido del juego**.
- **Programación de las funcionalidades del videojuego**.
- **Creación de Inteligencia Artificial** para los enemigos.
- **Creación de distintos niveles** que demuestren un reto para el jugador.

Así se quiere **aprender a utilizar Unity3D**, y adquirir la capacidad de desarrollar un videojuego desde cero, comprendiendo su proceso y requerimientos.

Otros sub-objetivos secundarios que no son prioritarios, pero que querría poder implementar son:

- **Permitir al usuario guardar el progreso del juego** para seguir en el punto donde lo había dejado.
- **Desarrollar un modo historia** para dar un fin a la aventura.

## 1.3. Alcance

El juego va dirigido a cualquier persona que disponga de un dispositivo Android, la dificultad para pasar cada nivel es apta para cualquier persona que, con experiencia o no con los videojuegos, desee entretenerse; no así el completarlo al 100%, que hará necesario el observar y buscar los secretos del nivel, requiriendo un mayor conocimiento sobre el mundo de los videojuegos.

## 1.4. Herramientas utilizadas

Para el desarrollo del TFG se va a hacer uso de distintas herramientas:

- **Unity3D**: motor gráfico. Unity3D es una plataforma de desarrollo flexible y potente para la creación de juegos interactivos multiplataforma, tanto en 3D como en 2D, y multiplataforma. Unity nos permite desarrollar un contenido de gran calidad, y dispone de una documentación extensa y completa.

- **UnityRemote**: aplicación Android que permite ejecutar el Editor de Unity desde el dispositivo móvil, pudiendo ver el funcionamiento del juego sin necesidad de exportarlo e instalarlo.

- **MonoDevelop**: IDE utilizado por Unity3D.

- **C#**: lenguaje de programación.Se ha elegido este en vez de Javascript debido a que hay un mayor número de usuarios de Unity que programan con este lenguaje, por lo que a la hora de resolver dudas es más fácil encontrar respuesta. A parte, el haber aprendido C y Java en la carrera, aun no siendo lo mismo que C#, facilita la comprensión y aprendizaje.

- **Spriter**: plataforma de animación 2D. Se empezó realizando las animaciones "frame by frame", lo que resulta un trabajo tedioso, por lo que se aprovechó la existencia de softwares como Spine o Spriter que permiten animar tus personajes en 2D de forma sencilla, ahorrándome muchísimo tiempo. Se eligió Spriter en vez de Spine por ser gratuito.

- **Adobe Photoshop CS3***:* software de diseño.

- **Wacom Intuos Pen small Tableta Gráfica***:* tableta gráfica que permite dibujar digitalmente, de esta forma los gráficos son más sencillos de crear y con mayor calidad.

- **Android SDK**: Kit de Desarrollo Software de Android.

- **Bfxr**: página web online que permite la creación de efectos de sonido mezclando distintos sonidos de una manera no muy compleja y muy libre.

- **Microsoft Office Professional Plus 2013, Gedit**: procesador de texto.

- **StarUML**, **Dia**: software para la realización de diagramas.

- **Draw.io**: herramienta online de creación de diagramas.

- **Google Drive**: plataforma de almacenamiento.

- **Mobizen**: programa software que permite visualizar la pantalla del dispositivo móvil a la vez en el ordenador al que se conecte, ya sea mediante USB o Wifi.

### 1.4.1. ¿Por qué Unity3D?

Se ha decidido hacer uso de la plataforma de desarrollo Unity3D, debido a su gran potencial para la creación de videojuegos y su integración multiplataforma, tanto para PC como para consolas (Xbox, PlayStation) como para dispositivos móviles (IOS, Android), permitiendo desarrollar y exportar los juegos a distintas plataformas a gran velocidad sin necesidad de demasiados cambios, excepto los específicos para cada una.

Inicialmente se trataba de un entorno de desarrollo de juegos 3D, por lo que los desarrolladores de juegos en 2D optaban por otras plataformas como Cocos2D, pero Unity introduce el modo 2D en su versión 4.6, que simplifica el desarrollo en 2D ofreciendo *físicas* 2D, cámaras 2D, etc..


Otra de sus ventajas es el ser gratuito, tiene sus limitaciones como la aparición obligatoria del logo de Unity en el *SplashScreen* (pantalla previa al juego), y su versión Pro que es muy cara (1500€), pero para el desarrollo de un juego sencillo no es necesario.

La mayor de sus ventajas es su gran documentación (tanto en inglés como en español *http://docs.unity3d.com/Manual/index.html*), para distintos lenguajes de programación (C#, Javascript y Boo), hay gran cantidad de tutoriales y el foro de ayuda de Unity te resuelve cualquiera de tus dudas. Y los *assets*, que son paquetes ya desarrollados que realizan distintas funcionalidades, desde juegos completos hasta el controlador del jugador, que permiten utilizarlos sin necesidad de programarlos uno mismo, ahorrando tiempo (*https://www.assetstore.unity3d.com/*).

La posibilidad de programar con distintos lenguajes de programación (a la vez), permitiendo usar C#, Javascript o Boo, lenguajes más fáciles que C++, y más eficientes para el desarrollo en Unity, pues permite centrarse sólo en las funcionalidades del juego y no en la gestión de memoria, punteros, etc.. ahorrando gran cantidad de tiempo. Es una gran ventaja de este motor gráfico. Su editor gráfico es intuitivo y de fácil uso, con distintas partes que te permiten trabajar de forma eficiente.

Todas estas ventajas y características, hace a Unity3D una de las plataformas más utilizadas y potentes en la actualidad a la hora de desarrollar videojuegos, y cada vez más en el ámbito específico de Android.

### 1.4.2. ¿Por qué Android?

Android es un sistema operativo de código abierto basado en Linux para dispositivos móviles, tanto smartphones, como tablets, PCs, etc.. El lenguaje de Android es Java y la extensión de las aplicaciones Android se denomina Android Package (.apk).

En su inicio fue desarrollado por Android Inc., pero en 2005 Google la adquirió, y en 2008 lo sacó al mercado, hasta llegar actualmente a ser el mayor sistema operativo en dispositivos móviles.

El funcionamiento de Android se basa en la máquina virtual Dalvik, que es la encargada de compilar el código Java el tiempo de ejecución. La estructura del sistema operativo se basa en un sistema de capas ejecutado sobre un framework de Java orientado a objetos en la máquina virtual Dalvik:

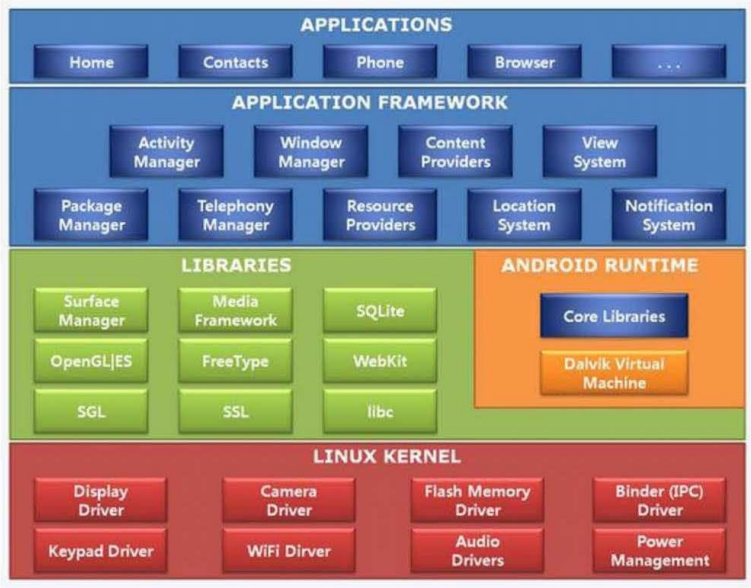



**Figura 1. Estructura de Android**

Junto a las librerías, el núcleo Kernel de Linux constituyen el corazón de Android, entre las librerías se pueden encontrar OpenGL que se encarga de los gráficos o SQLite que gestiona las bases de datos.

Las ventajas de Android, aparte de ser de **código libre**, pudiendo modificarlo y compartirlo libremente, son:

- **El más utilizado actualmente**: si se quiere adquirir un dispositivo móvil, la mayor parte de las posibilidades serán Android, debido a su gran potencial y ser el más utilizado hoy en día.

- **Precio**: comparado con otros sistemas operativos, debido al amplio rango de compañías que disponen de móviles con dicho sistema operativo, se pueden encontrar dispositivos muy baratos con gran relación precio/calidad.

- **Google**: el ser hijo de Google permite disponer de cantidad de servicios de esta compañía, como GoogleDrive, Gmail, Youtube, etc.. garantizando la compatibilidad con ellas.

- **Multitarea**: permite ejecutar distintas aplicaciones a la vez, sin necesidad de cerrarlas, permaneciendo en background.

- **GooglePlay**: a diferencia de la AppleStore de Apple, las aplicaciones de la tienda de Android no tienen por qué ser de pago, al contrario, la mayor parte de las aplicaciones que encontramos se tratan de aplicaciones gratuitas. Además, los desarrolladores deberán pagar solamente una licencia a la hora de registrarse en la tienda, a diferencia de en la AppleStore que deberían pagar cada vez que deseen publicar una aplicación en la tienda.

- **Exportación**: Unity3D permite exportar los proyectos a Android rápidamente y sin tener que pagar, al contrario que con iOS, que debes disponer de la versión Pro del motor gráfico.

No todo son ventajas, Android también tiene varias desventajas:

- **Seguridad**: el poder publicar cualquier tipo de aplicación sin pagar hace que se cuelen aplicaciones ofensivas o perjudiciales. Aunque Android dispone de varios controles, siempre hay alguna forma de ignorarlos y conseguir introducirse en otros dispositivos.

- **Gasto de batería**: el sistema operativo Android consume mucha batería debido a su característica multitarea.

Pese a ello, sigue siendo el más utilizado en la actualidad, y un gran mercado para los iniciados en el desarrollo de aplicaciones, o videojuegos. Es por ello por lo que se ha elegido para el desarrollo del proyecto, a parte del no encontrar ningún juego que cumpla con los objetivos planteados en él, teniendo una buena visión de negocio.

## 1.5. Organización del documento

Esta sección define la estructura y el contenido de cada parte del documento. Se dividirá en Capítulos, cada capítulo con distintas secciones:

- **Capítulo 2. Estado del Arte**: análisis del entorno de desarrollo y juegos similares de referencia.
- **Capítulo 3. Introducción a Unity3D**: introducción al motor gráfico Unity3D y explicación breve de su estructura y capacidades.
- **Capítulo 4. Planificación y Presupuesto**: en este capítulo se establece la metodología a seguir. La planificación y el presupuesto del proyecto.
- **Capítulo 5. Análisis**: identificación de los requisitos. Casos de Uso. Diagramas de secuencia. Árbol de Características.
- **Capítulo 6. Diseño**: en Diseño se encuentra el diagrama de clases de diseño. Arquitectura software. Diagramas de Estados. Diseño de Interfaz. Diseño del juego.
- **Capítulo 7. Implementación**: explica la parte del desarrollo del proyecto en su fase de implementación.
- **Capítulo 8. Pruebas**: pruebas realizadas durante el desarrollo del videojuego.
- **Capítulo 9. Manual de Instalación**: en este capítulo se explica cómo instalar el juego en el dispositivo móvil.
- **Capítulo 10. Manual de Usuario**: en este capítulo se explica el modo de uso del videojuego.
- **Capítulo 11. Conclusiones**: conclusiones tras el desarrollo del proyecto. Futuras mejoras.
- **Capítulo 12. Bibliografía**: libros, artículos, tutoriales y páginas web consultados como ayuda para la realización del proyecto.
- **Capítulo 13. Glosario**: vocabulario técnico, argot, siglas, contenidas en el proyecto.
- **Capítulo 14. Contenido del CD-ROM**.

# Capítulo 2
# Estado del Arte

En este capítulo se analiza el entorno en el que se desenvuelve el TFG, comparándolo con las opciones actuales y las ventajas del desarrollo de un juego para Android en la actualidad.

La industria del videojuego está cada vez más en auge, sobre todo en el ámbito de los dispositivos móviles, así es que tanto en la AppStore como en GooglePlay, el 20% de aplicaciones corresponde a juegos.

Según un estudio realizado por la firma Digi-Capital[2], la industria de los videojuegos alcanzará un volumen de negocio de 100.000 millones de dólares en 2017, lo que muestra el gran potencial que ofrece.

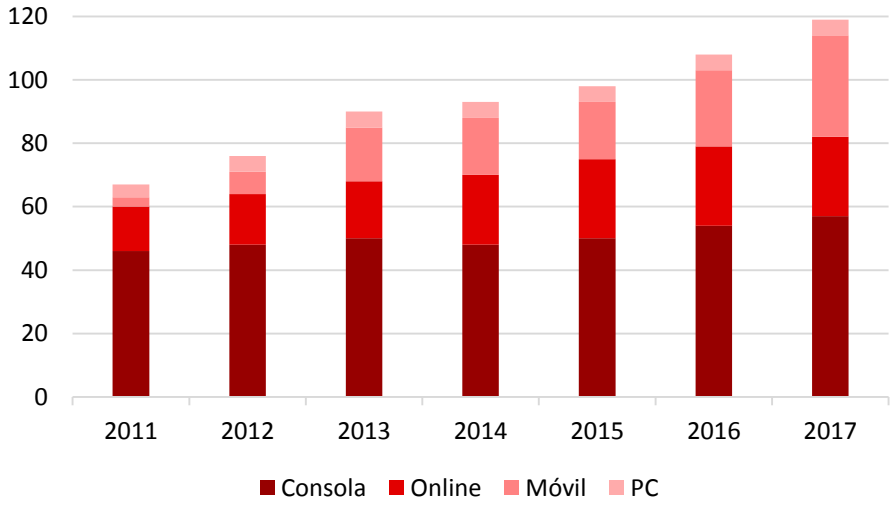



**Figura 2. Evolución del volumen de negocio en millones de $ de la industria del videojuego**

[2] http://www.digi-capital.com/news/2014/01/mobile-driving-games-revenue-to-100b-by-2017-and-5-6b-ma-in-2013/#.VYPQU_ntmko

Los juegos son y seguirán siendo el principal tipo de aplicación de descarga en dispositivos móviles, como muestra la **Figura3**[3]:

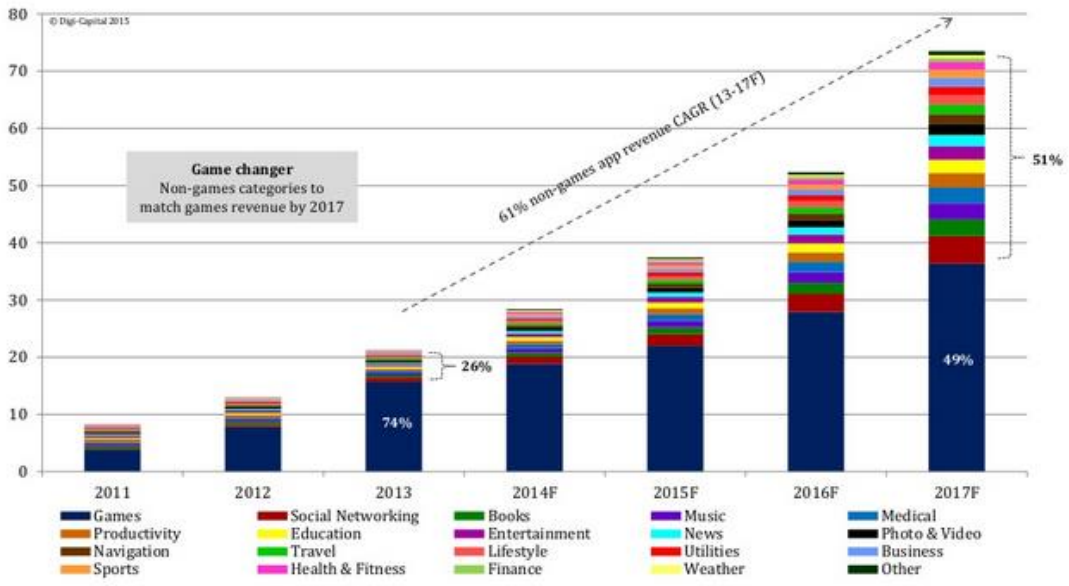



**Figura 3. Evolución de los tipos de aplicación de descarga en dispositivos móviles**

Comparando entre entornos móviles, IOS, Android, WindowsPhone, como podemos observar en la **Figura4**, a diferencia de hasta ahora, Android supera a IOS en la cabeza de la lista, aumentando de un 35% a un 48%, mientras que IOS baja de un 54% a 41%. El problema que tenía Android era la fragmentación de sus versiones, pero debido a las medidas tomadas por Google para solucionar este tema, se ve cómo le gana la primera posición a IOS. Symbian y Windows Phone parecen no ser una gran amenaza por ahora.

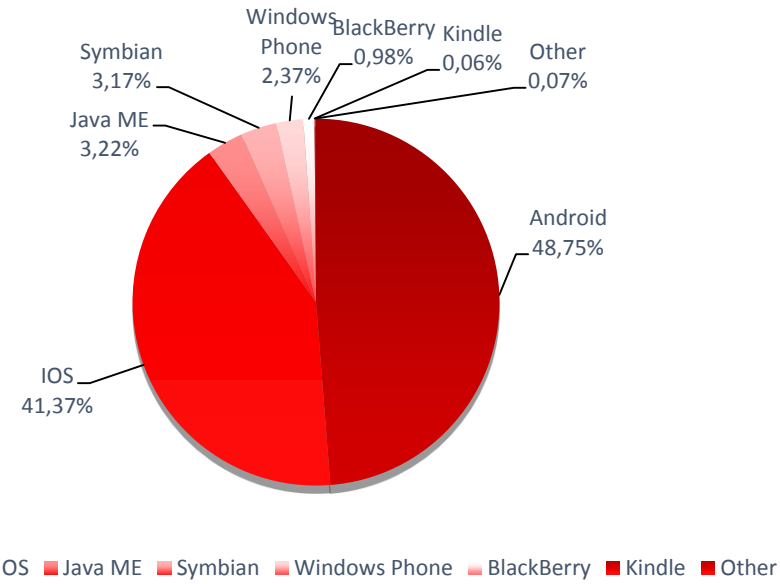



**Figura 4. Comparativa del uso de los distintos SO en dispositivos móviles 2015**

[3] http://es.slideshare.net/appbackr/digi-capital-mobile-internet-investment-review-q2-2014-summary?related=3

Analizando los datos, se ha decidido desarrollar el videojuego para Android, aunque también podría ejecutarse en PC, IOS y Xbox debido a la característica multiplataforma de Unity3D. El desarrollo de videojuegos para consolas como Xbox o PlayStation no es tan abierto ante juegos de grupos muy reducidos de personas y con poco presupuesto, ya que no contienen ninguna tienda específica para descargar aplicaciones, lo más parecido son los *juegos Arcade*, pero suelen ser de pago y difícilmente dados a conocer.

Se ha elegido la versión mínima *Android 4.0 'Ice Cream Sandwich' API 14* para poder llegar a un mayor número de usuarios, sin llegar a versiones demasiado antiguas y desactualizadas, pero sin dirigirse a la última versión, pues, aunque haya menos fragmentación, sigue existiendo, y no se trata de un videojuego que requiera mucha potencia en el dispositivo.

El desarrollo de videojuegos ha evolucionado, pasando de equipos reducidos de menos de 10 personas trabajando durante 1 año, con un presupuesto muy bajo, a desarrollarse con más de 100 durante varios años y presupuestos de millones. Esta evolución se identifica en los *juegos AAA*, los cuales destacan en todos sus apartados (jugabilidad, gráficos, sonido).

En el caso de juegos para móviles, es distinto, ya que es un mercado muy grande en el que predominan los *juegos Free-To-Play* (gratuitos), por lo que las compañías más poderosas del mundo de los videojuegos no se atreven a sumergirse en él (por ahora), hay casos como *Rayman Jungle Run* de Ubisoft o *Hearthstone* de Blizzard que han dado el paso y han recibido una muy buena acogida.

Hearthstone Rayman Jungle Run

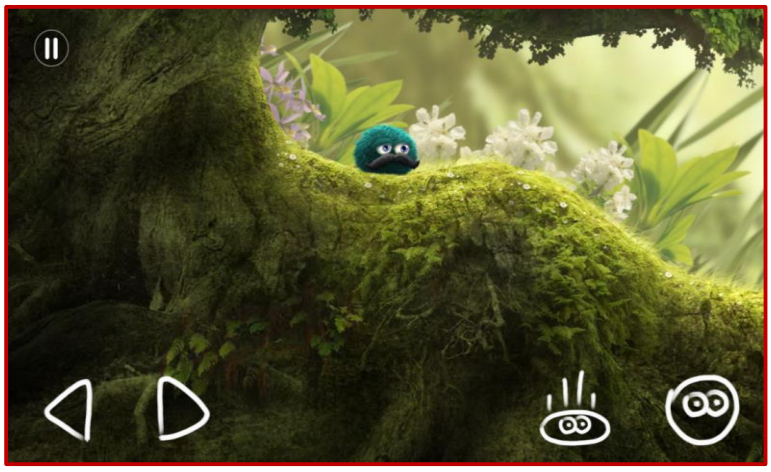
En contraposición a los videojuegos AAA, están los denominados indies (independientes), producidos por pequeños equipos, que sin el apoyo económico de las grandes compañías, desarrollan videojuegos que prescinden de grandes gráficos e historia, centrándose en la esencia del juego, haciéndolo divertido y no repetitivo.

Como vemos, esto define este proyecto, ya que es una única persona la encargada de los gráficos, programación, música y argumento del videojuego, con un presupuesto bajo.

## 2.1. Juegos referencia


**Leo's Fortune**

Juego que empezó en iOS y recibió muy buenas críticas, llegando a ganar el premio Apple Design en 2014, por lo que fue desarrollado en Android, donde también tiene muy buena nota. El protagonista es Leopold Fortunate (Leo), un ingeniero que posee una gran fortuna de oro, al cual tendremos que ayudar a recuperar su oro robado a lo largo de 24 distintos niveles de plataformas y puzles. El fuerte de este juego es su apartado gráfico, detallado y bonito, que, junto con un control fluido y sencillo, te hace no querer parar de jugar.



✓ **Pros**: gráficos, controles, fluidez, chekpoints.

✗ **Contras**: duración. Para ser un videojuego de corta duración, el coste de $4.99 es algo excesivo, comparando con otros de características similares.


**Lep's World**

Juego que ya dispone de 3 ediciones y miles de descargas. Quiere parecerse al famoso Super Mario Bros, pero se queda en el camino. Los gráficos no son muy buenos, la interfaz algo caótica y no siempre responde correctamente al usuario. Para mi gusto es lento, sientes que el personaje se mueve con lentitud. Los niveles son muy parecidos todos, y no hay ninguna relación entre los objetos que te encuentras, tanto enemigos aleatorios como ¿piñas? que lanzas para acabar con ellos no tienen ningún tema en común. Es uno de los pocos juegos gratuitos que dispone de checkpoints, aunque escasean en los niveles.


✓ **Pros**: chekpoints, sencillez, duración.

✗ **Contras**: gráficos, interfaz, lento, publicidad.

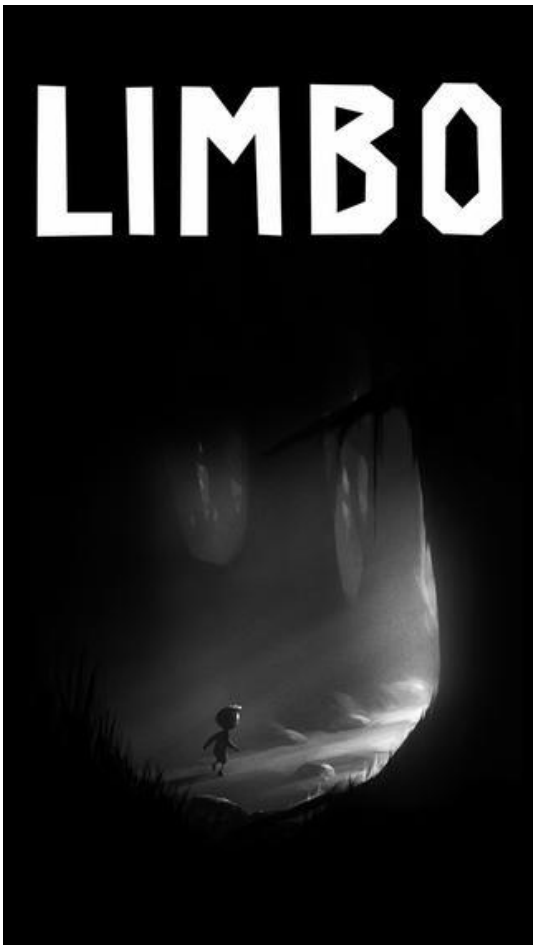

**Super Oscar**

Al igual que el Lep's World, intenta imitar al querido Super Mario Bros, copiando el nombre Super Mario – Super Oscar, las nubes y bloques de monedas, etc.. Pero no tiene nada que ofrecer en comparación, interfaz muy caótica, publicidad en cada paso que das, no hay checkpoints, y no siendo los niveles muy cortos, molesta mucho el tener que volver al inicio cada vez que mueres. Tiene bastantes bugs y, al igual que en Lep's World, pero en este caso más exagerado, los enemigos no tienen un tema en común, hay mucha variedad, pero sin ningún sentido.


✓ **Pros**: gráficos, sencillez, duración.

✗ **Contras**: interfaz, publicidad, checkpoints, bugs.

## Limbo




Uno de los referentes en cuando a juegos indies. Destacado por su apartado gráfico y sonoro minimalistas. Sin color ni diálogos, ni historia, pero con algo que te lleva a no querer dejar de jugar. Trata sobre un niño a quien debemos guiar por un mapa en blanco y negro, sin saber a dónde ni por qué, podemos decir que estamos tan perdidos como el protagonista del juego.

Se desarrolló para Xbox, y posteriormente para PC. No hay versión para dispositivos móviles, pero he creído necesario el comentar uno de los mejores juegos indies de plataformas.

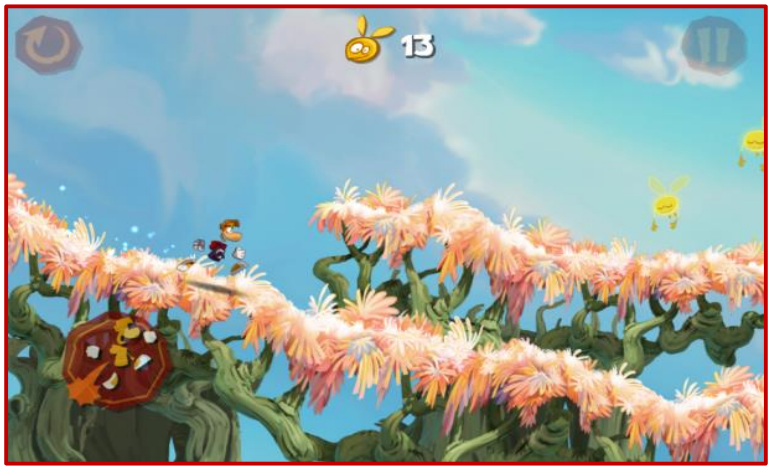
Limbo es, en sí, una obra de arte convertida en videojuego que consigue superar a algunos otros de grandes compañías.

✓ **Pros**: gráficos, sonido, jugabilidad, sencillez.

✗ **Contras**: duración. Ha recibido muchas críticas por la corta duración del juego.


**Rayman Jungle Run**

Un claro ejemplo del auge de los juegos en Android y su potencia, ya que Rayman es uno de los iconos de la gran compañía Ubisoft, y han decidido a dar el salto a esta plataforma, recibiendo una gran acogida. El que un juego cueste dinero, no siempre es malo, pues con ese dinero se mantiene y actualiza el juego constantemente, permitiendo disfrutar de un gran juego como este. En este caso no se trata de un juego de plataformas, sino uno del género conocido como *endlessrun*, en el que no controlamos el movimiento del personaje, sino sólo acciones como saltar, golpear. Los gráficos, música y control le hacen ser digno de ser un videojuego AAA.




✓ **Pros**: gráficos, sencillez, banda sonora, duración.

✗ **Contras**: el ser endlessrun hace repetitivos los niveles, y la música que son pocas pistas repetidas durante muchos niveles.

# Capítulo 3
# Introducción a Unity3D

En este capítulo se introduce el motor gráfico Unity3D, sus características y modo de uso. El aprendizaje sobre Unity3D ha sido autónomo por medio de tutoriales, la comunidad de Unity, y mucha práctica.

Los proyectos en Unity3D se basan en la utilización de Assets (paquetes de recursos), los cuales contienen materiales, efectos, scripts, sprites, etc... Unity dispone de la Asset Store, una tienda donde pueden adquirirse recursos ya creados por otros desarrolladores, gratuitos o de pago, que permiten reducir el tiempo de desarrollo, y evitar "reinventar la rueda". Para los iniciados en este motor gráfico, como en este caso, Unity te facilita ayuda mediante una extensa comunidad en la que puede encontrarse cualquier tipo de problema resuelto, una gran ayuda a la hora de empezar a utilizar un software desconocido y tan potente.

La Interfaz de Unity es intuitiva y sencilla, no ha sido difícil familiarizarse con ella en un periodo corto de tiempo. Esta se divide en varias secciones (*Interfaz de Unity3D*):

1. **Project**: en esta sección se muestra la estructura del proyecto en Unity, en ella se muestran los Assets que incluyen los recursos a usar en las escenas del videojuego.

2. **Hierarchy**: objetos de la escena actual.

3. **Inspector**: una de las partes más importantes de Unity3D, aquí se muestran las características del objeto seleccionado, aquí se añaden los atributos y scripts que dan vida a los objetos.

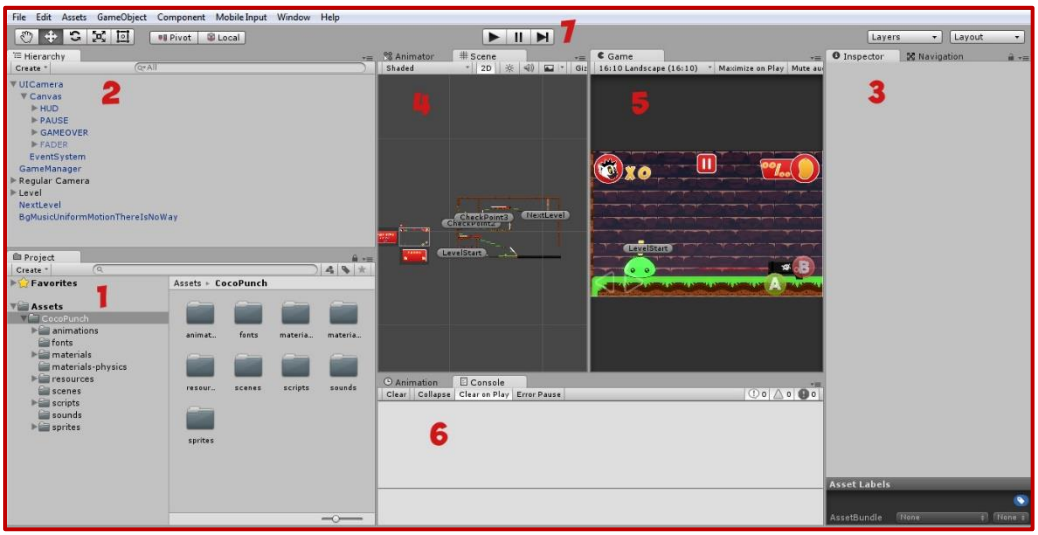
4. **Scene**: al igual que en Android se tienen Activities, o pantallas que el usuario visualizará, Unity tiene las Scenes, donde se pueden añadir los recursos para construir los niveles o pantallas del videojuego.

5. **Game**: sección que muestra cómo se verá el juego al compilarse, hay que tener cuidado con el AspectRatio o resolución de la pantalla elegida, dependiendo de la plataforma sobre la que se desarrolle el proyecto, se mostrarán unas u otras. En el proyecto se utilizará una resolución para dispositivos Android en posición horizontal (16:10 Landscape).

**6**. **Console**: muestra los mensajes de error/debug.

**7**. **Game controller**:

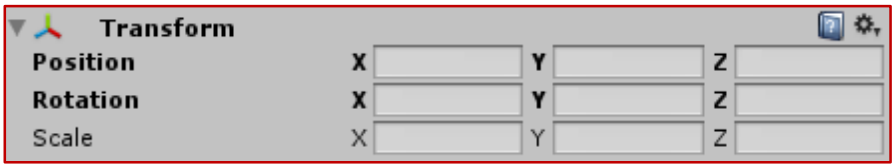
- **Play**: ejecuta/detiene el juego.
- **Pause**: pausa el juego, permitiendo observar la escena sin necesidad de detener el juego por completo.




Interfaz de Unity3D

Al tratarse de un videojuego en 2D, no van a usarse materiales ni físicas en 3D, sino en 2D. Para materiales, en Unity se utilizan los denominados *Sprites*, mapa de bits 2D, y los *SpriteSheet*, que son conjuntos de sprites. El empaquetar sprites en spritesheets reduce la memoria utilizada; Unity permite dividir los sprites de un conjunto.

Un videojuego en Unity se basa en ***GameObjects*** (objetos), que siempre tienen una característica en común:




*Transform* indica la posición, rotación y tamaño del objeto en las 3 dimensiones (X,Y,Z), al trabajar en 2, solo se hace caso a X e Y.

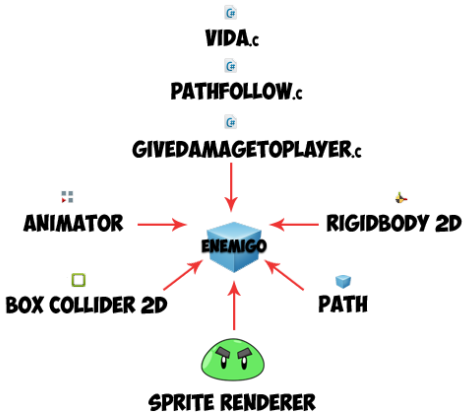
A estos GameObjects se les puede añadir todo tipo de ***Componentes*** (efectos, audio, físicas, render, scripts, UI, etc..) y es con lo que compone los niveles del juego.

Unity facilita las cosas permitiendo crear ***Prefabs***, que son objetos que predefines con unos componentes, y puedes añadirlos las veces que quieras con solo arrastrarlos hacia tu Scene (¡cuidado con modificar un Prefab sin querer!).

Componentes necesarios para conocer son, entre otros:

- **Sprite Renderer**: convierte un objeto en un sprite 2D.
- **Animator**: permite a un objeto tener distintos estados de animación.
- **Physics 2D**: permite a un objeto colisionar con otros.
- **RigidBody**: permite a un sprite interactuar con las físicas del juego.
- **Camera**: convierte un objeto en una cámara.
- **Particle System**: convierte un objeto en generador de partículas.
- **Script**: permite añadir funcionalidades y comportamientos a los objetos.

Unity se compone de GameObjects que tienen distintos componentes, pero ¿cómo hacer que un objeto tenga vida, o que un objeto te ataque? De eso se encargan los scripts, donde se programan los comportamientos de los objetos, y el funcionamiento del videojuego. Por ejemplo, si se tiene un objeto Enemigo, y se quiere que tenga vida, que patrulle un camino y que pueda atacarnos al entrar en su visión, se le deberán añadir distintos componentes:




En cada script deberá programarse el código que realice la función requerida. Los scripts en Unity deben heredar de la clase *MonoBehaviour*, que es la clase principal de Unity, disponiendo de distintas funciones que siguen un ciclo de vida (**Figura 5**) como *Awake()* – inicializa el script, *Start()* – se ejecuta una sola vez al activarse el script o *Update()* – se ejecuta una vez cada frame.

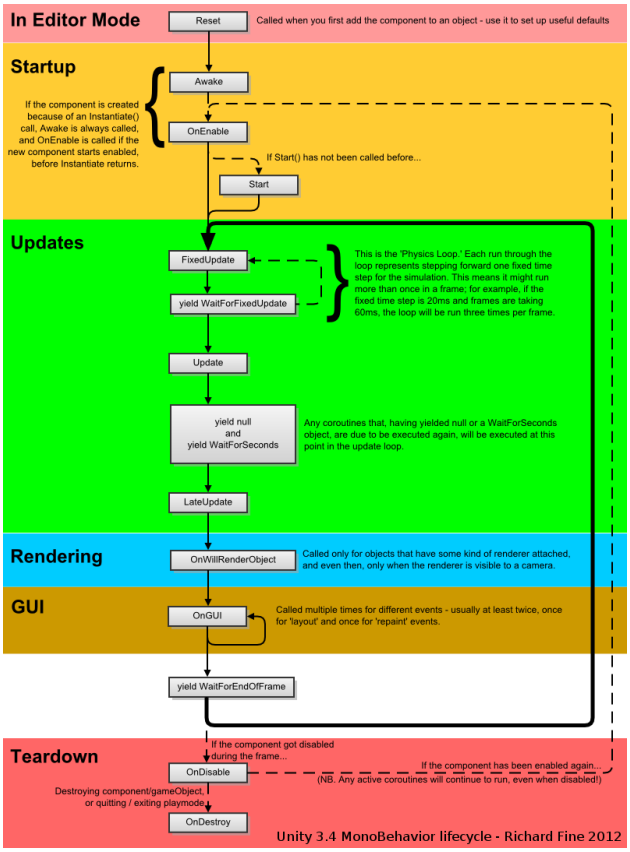



**Figura 5. Ciclo de vida de Monobehavior**

Unity3D dispone de funciones del tipo *Couroutine* (corrutina) que permiten pausar la ejecución del código y continuar con él en el siguiente frame. Se declaran:

```
función(){yield return;}
IEnumerator función(){yield return;}
```

La instrucción de retorno `yield return` es la encargada de pausar la ejecución y reanudarla en el siguiente frame. Para iniciar una Couroutine se debe llamar a la función `StartCouroutine(función)`. Este tipo de funciones será utilizado cuando no se necesite ejecutar cada frame como haría el método *Update()*, de esta forma se reduce notablemente el número de comprobaciones, mejorando la experiencia de juego.

La arquitectura de Unity3D (**Figura 6**) se basa en componentes, permitiéndola ser modular y extensible. Esto quiere decir que es a través de los Componentes de los GameObjects que se encuentran en las Escenas como se comunica Unity con la lógica del juego controlada por Mono. El flujo de ejecución de los scripts es controlado por el motor gráfico, el cual es controlado por el Runtime de Mono. Unity3D compila los assets y sus componentes a través del SDK de Android.

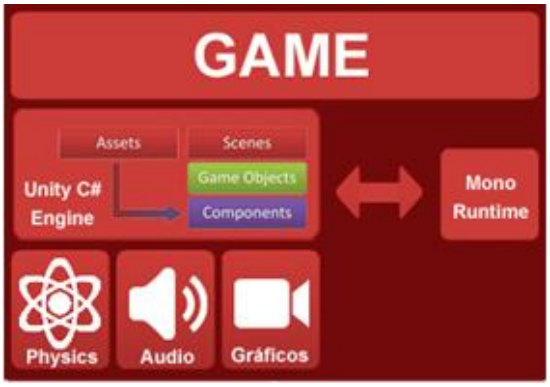



**Figura 6. Arquitectura de Unity3D**

La versión 4.6 de Unity introduce un nuevo sistema de Interfaz de Usuario que permite, de forma rápida y sencilla crear los menús y componentes de la IU, adaptándose a los distintos dispositivos en los que puede ser ejecutado.

Todas estas características convierten a Unity3D en una de las mejores opciones a la hora de desarrollar un videojuego, siendo usado tanto por grandes empresas como por pequeños equipos independientes.

# Capítulo 4
# Planificación y presupuesto

En este capítulo se describe la metodología elegida para el desarrollo del proyecto, la planificación y el presupuesto del proyecto.

## 4.1. Metodología

Como metodología se ha escogido el **modelo Incremental**, ya que los requisitos no están establecidos inicialmente, pudiendo ir añadiendo funcionalidades a la vez que se dispone de una versión funcional del producto, pudiendo adaptarme ante cambios. Cada iteración añade objetivos o mejora los ya completados.

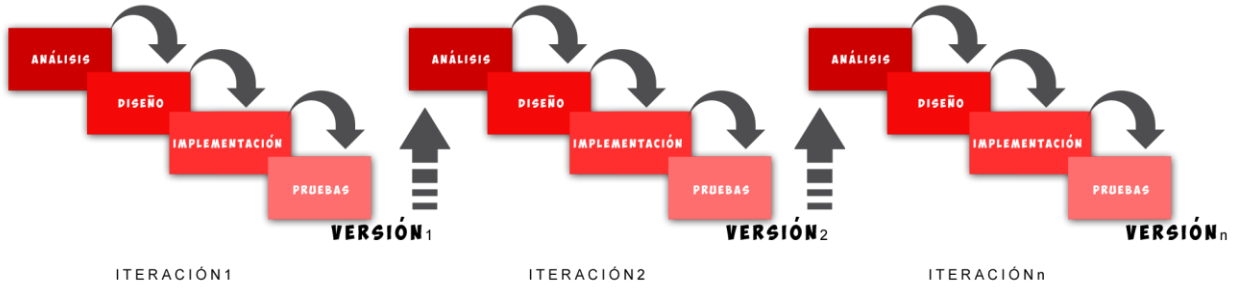



**Figura 7. Metodología Incremental**

## 4.2. Planificación

Durante el desarrollo del proyecto, va a haber varias fases:

- **Análisis**: fase de análisis sobre qué hacer, cómo hacerlo, con qué herramientas, limitaciones y objetivos a conseguir con este proyecto. Esta fase será realizada por el analista del proyecto.

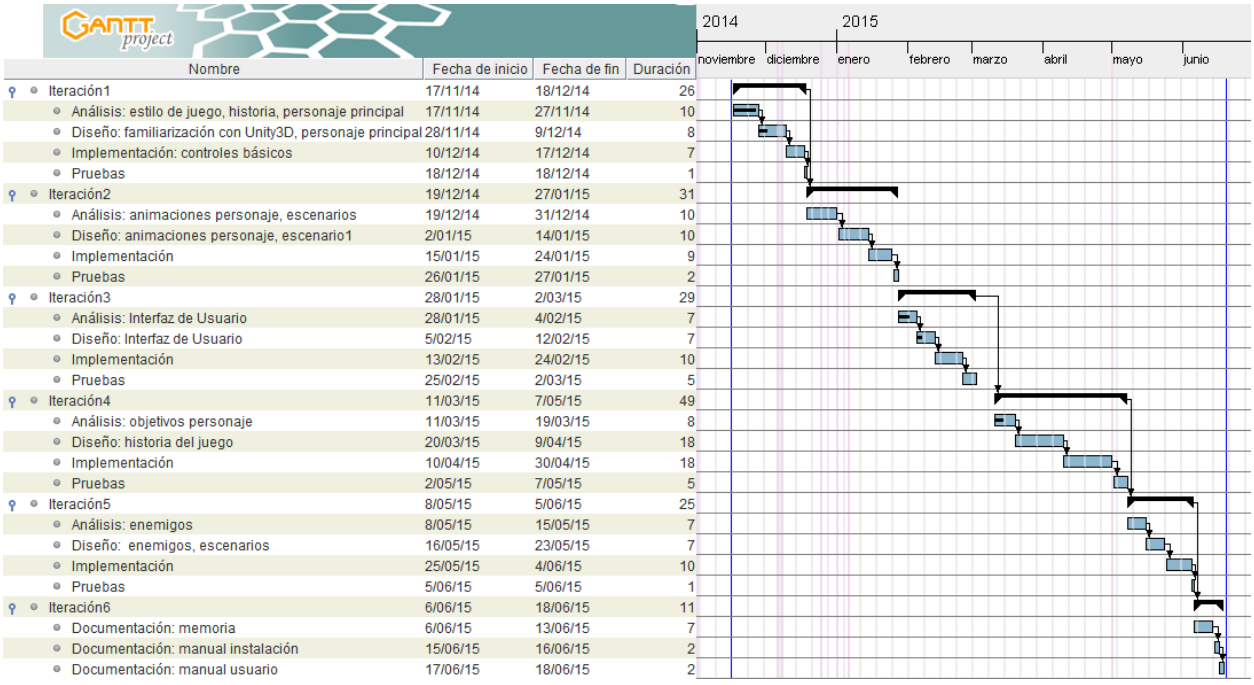
- **Diseño**: diseño del juego, tanto el personaje principal como los enemigos, objetos y plataformas del juego. Esta fase será realizada por el diseñador del proyecto.

- **Implementación**: desarrollo del videojuego CocoPunch. Programación de las funciones y niveles del juego. Esta fase será realizada por el programador del proyecto.

- **Pruebas**: fase en la que se comprueba las funcionalidades implementadas. Esta fase será realizada por el programador del proyecto.

- **Documentación**: fase en la que se recoge lo realizado, cómo se ha realizado y demás información necesaria para entender el objetivo del TFG y el trabajo que ha requerido. Esta fase será realizada por el documentador del proyecto.

Cada iteración comprende las fases indicadas, se planificó dividir las iteraciones por meses, mostrado en el *Diagrama de Gantt* (**Diagrama 1**).




| Nombre | Fecha de inicio | Fecha de fin | Duración |
|---|---|---|---|
| Iteración1 | 17/11/14 | 18/12/14 | 26 |
| Análisis: estilo de juego, historia, personaje principal | 17/11/14 | 27/11/14 | 10 |
| Diseño: familiarización con Unity3D, personaje principal | 28/11/14 | 9/12/14 | 8 |
| Implementación: controles básicos | 10/12/14 | 17/12/14 | 7 |
| Pruebas | 18/12/14 | 18/12/14 | 1 |
| Iteración2 | 19/12/14 | 27/01/15 | 31 |
| Análisis: animaciones personaje, escenarios | 19/12/14 | 31/12/14 | 10 |
| Diseño: animaciones personaje, escenario1 | 2/01/15 | 14/01/15 | 10 |
| Implementación | 15/01/15 | 24/01/15 | 9 |
| Pruebas | 26/01/15 | 27/01/15 | 2 |
| Iteración3 | 28/01/15 | 2/03/15 | 29 |
| Análisis: Interfaz de Usuario | 28/01/15 | 4/02/15 | 7 |
| Diseño: Interfaz de Usuario | 5/02/15 | 12/02/15 | 7 |
| Implementación | 13/02/15 | 24/02/15 | 10 |
| Pruebas | 25/02/15 | 2/03/15 | 5 |
| Iteración4 | 11/03/15 | 7/05/15 | 49 |
| Análisis: objetivos personaje | 11/03/15 | 19/03/15 | 8 |
| Diseño: historia del juego | 20/03/15 | 9/04/15 | 18 |
| Implementación | 10/04/15 | 30/04/15 | 18 |
| Pruebas | 2/05/15 | 7/05/15 | 5 |
| Iteración5 | 8/05/15 | 5/06/15 | 25 |
| Análisis: enemigos | 8/05/15 | 15/05/15 | 7 |
| Diseño: enemigos, escenarios | 16/05/15 | 23/05/15 | 7 |
| Implementación | 25/05/15 | 4/06/15 | 10 |
| Pruebas | 5/06/15 | 5/06/15 | 1 |
| Iteración6 | 6/06/15 | 18/06/15 | 11 |
| Documentación: memoria | 6/06/15 | 13/06/15 | 7 |
| Documentación: manual instalación | 15/06/15 | 16/06/15 | 2 |
| Documentación: manual usuario | 17/06/15 | 18/06/15 | 2 |

**Diagrama 1. Diagrama de Gantt**

- **Iteración1**:
  - **Análisis**: definir el estilo del juego, estética e historia, así como el boceto del personaje principal.
  - **Diseño**: familiarizarse con Unity3D y el diseño del personaje principal.
  - **Implementación**: importar a Unity3D el personaje principal e implementar los controles básicos de mover al personaje.
  - **Pruebas**: las pruebas en este caso no son del juego sino del diseño de éste.

- **Iteración2**:
  - **Análisis**: análisis de las animaciones del personaje y los componentes de los niveles.
  - **Diseño**: animaciones del personaje principal y diseño de los bloques con compondrán los niveles.
  - **Implementación**: implementación de las animaciones del personaje principal y creación de la primera escena del juego con los bloques diseñados.
  - **Pruebas**: pruebas de las animaciones del personaje y las físicas con los bloques del nivel.

- **Iteración3**:
  - **Análisis**: analizar los componentes de la Interfaz de Usuario del juego.
  - **Diseño**: diseño de los componentes de la GUI.
  - **Implementación**: implementación de la GUI del juego.
  - **Pruebas**: probar las animaciones de la GUI del juego y su correcta respuesta.

- **Iteración4**:
  - **Análisis**: analizar el objetivo e historia del juego.
  - **Diseño**: diseño de las imágenes de la historia del juego, y del mapa de niveles.
  - **Implementación**: implementación de la secuencia de imágenes que componen la historia del juego, y añadir el mapa de niveles a la Interfaz de Usuario de la pantalla de inicio.
  - **Pruebas**: probar el funcionamiento de las imágenes de la historia, y el cargar niveles desde el mapa.

- **Iteración5**:
  - **Análisis**: analizar los enemigos a diseñar, sus características y modo de ataque.
  - **Diseño**: diseñar los enemigos del juego, y sus animaciones y diseñar nuevos niveles.
  - **Implementación**: implementar las funcionalidades de los enemigos, como atacar, y añadir los nuevos niveles diseñados al juego.
  - **Pruebas**: probar las funcionalidades de los enemigos y los nuevos niveles.

- **Iteración6**:
  - **Análisis**: establecer la estructura del documento y cómo realizarlo.
  - **Diseño**: diseñar el Índice del documento, el estilo y composición.
  - **Implementación**: corrección de fallos que pueda tener el juego.
  - **Pruebas**: pruebas del juego para comprobar su perfecto funcionamiento.

## 4.3. Presupuesto inicial

Siguiendo la planificación definida en el *Diagrama de Gantt* (**Diagrama1**), se estima el presupuesto del proyecto. Se divide entre presupuesto hardware que contiene los elementos hardware que van a ser necesarios para el desarrollo del proyecto; presupuesto software que contiene el software necesario, y presupuesto de desarrollo que trata de los recursos humanos necesarios.

### 4.3.1. Presupuesto hardware

Hardware necesario para el desarrollo del proyecto:

- **Ordenador personal**: se va a requerir de un ordenador para la realización del proyecto, dicho ordenador deberá contener el sistema operativo Windows. Contando que un ordenador portátil tiene una vida media de 4 años, y que se va a hacer uso de él durante 28 semanas, su uso se calcula:

  100% ---------- 192 semanas
  **x**% ---------- 28 semanas
  **x** = (28*100)/192 = **14,6%** de uso.

- **Dispositivo móvil**: también será necesario un dispositivo móvil con el sistema operativo Android para instalar y probar el videojuego. Calculamos su uso del mismo modo, contando que la vida media de un móvil hoy en día es de 3 años, y va a hacerse uso durante 28 semanas:

  100% ---------- 144 semanas
  **x**% ---------- 28 semanas
  **x** = (28*100)/144 = **19,4%** de uso.

- **Cable microUSB**: para conectar el dispositivo móvil al ordenador y transferir la aplicación va a requerirse un cable para ello, el tipo de cable que lo permite se llama microUSB, debido al bajo coste, se establece el porcentaje de uso al **50%**.

- **Wacom Intuos Pen Tableta Gráfica CTL-480S**: tableta gráfica necesaria para el diseño del videojuego de manera más sencilla y con un mejor resultado. Su vida media es de 3 años, estableciendo que se usará sólo durante las fases de Diseño, se usará durante 4 semanas. El porcentaje de uso de la tableta según el tiempo de desarrollo es:

  100% ---------- 144 semanas
  **x**% ---------- 4 semanas
  **x** = (4*100)/144 = **2,8%** de uso.

- **Conexión a Internet**: depende del precio de la compañía con quien esté contratado, suponiendo que se trata de Orange, el precio de la banda ancha es de 42€, al ser pago por mes, se considera un **40%** su uso mensual.

- **Impresora HP Deskjet F2280 All-in-One**: para la documentación se requerirá de una impresora, el gasto de tinta se estima de 50€, haciendo uso del **100%**.

| Hardware | Uso (%) | Coste (€) | (Uso * Coste) / 100 |
|---|---|---|---|
| **Ordenador personal portátil: Sony VAIO VPCEH** | 14,6% | 520 | 75,92 |
| **Dispositivo móvil: THL W100** | 19,4% | 159 | 30,85 |
| **Cable microUSB** | 50% | 1,45 | 0,725 |
| **Wacom Intuos Pen Tableta Gráfica CTL-480S** | 2,8% | 60,97 | 1,7 |
| **Conexión a Internet** | 40% 7 meses | 42*7 | 117,6 |
| **Impresora HP Deskjet F2280** | 100% | 50 | 14 |
| | | **TOTAL (€):** | **240,795** |

**Tabla 1. Presupuesto inicial - Presupuesto hardware**

### 4.3.2. **Presupuesto software**

Software necesario para el desarrollo del proyecto:

- **Unity3D, UnityRemote, Android SDK, Monodevelop, Spriter, bfxr, Dia, StarUML, Draw.io y Google Drive** son gratuitos, por lo que el coste es de 0€.

- **Adobe Photoshop CS3**: la licencia mensual de este software es de 19,99€, estimando su uso de 4 semanas, al igual que la tableta gráfica, se tiene el coste de una licencia mensual, es decir, el **100%**.

- **Microsoft Office Professional Plus 2013**: el coste de la licencia de este paquete es de 539€ y se usa el mismo porcentaje que el ordenador donde se instala, **14,6%.**

| Software | Uso (%) | Coste (€) | (Uso * Coste) / 100 |
|---|---|---|---|
| **Unity3D** | - | - | 0 |
| **Android SDK** | - | - | 0 |
| **Unity Remote** | - | - | 0 |
| **Monodevelop** | - | - | 0 |
| **Adobe Photoshop CS3** | 100% | 19,99 | 19,99 |
| **Spriter** | - | - | 0 |
| **Bfxr** | - | - | 0 |
| **Dia** | - | - | 0 |
| **StarUML** | - | - | 0 |
| **Draw.io** | - | - | 0 |
| **Microsoft Office Professional Plus 2013** | 14,6% | 539 | 78,694 |
| **Google Drive** | - | - | 0 |
| | | **TOTAL (€):** | **98,684** |

**Tabla 2. Presupuesto inicial – Presupuesto software**

### 4.3.3. **Presupuesto de desarrollo**

Costes referidos al desarrollo del proyecto por parte de un Ingeniero Informático de Servicios y Aplicaciones.

Se van a requerir distintas categorías de trabajo:

- **Analista**: encargado de la fase de *Análisis* del proyecto.
- **Diseñador**: encargado de la fase de *Diseño* del proyecto.
- **Programador**: encargado de la fase de *Implementación* y *Pruebas* del proyecto.
- **Documentador**: encargado de la fase de *Documentación* del proyecto.

Contando que se ha planificado la duración del proyecto de algo más de 7 meses, trabajando todos los días de la semana, exceptuando los sábados y domingos, 8 horas al día:

171 días*8h/día = **1368h**.

Al dividir el proyecto en iteraciones, cada fase será realizada por un especialista, permitiendo conocer mediante el Diagrama de Gantt las horas realizadas por cada uno.

| | Tiempo | Coste (€) |
|---|---|---|
| **Analista** | 336h | 10€/h |
| **Diseñador** | 400h | 10€/h |
| **Programador** | 544h | 12€/h |
| **Documentador** | 88h | 8€/h |
| **TOTAL (€):** | **14592** | |

**Tabla 3. Presupuesto inicial – Presupuesto de desarrollo**

### 4.3.4. **Presupuesto total**

El presupuesto total es la suma de los presupuestos de hardware, software y desarrollo y otros gastos referentes a material de oficina, DVDs, etc..:

| | Coste (€) |
|---|---|
| **Hardware** | 240,795 |
| **Software** | 98,684 |
| **Desarrollo** | 14592 |
| **Otros** | 50 |
| **TOTAL (€):** | **14981,479** |

**Tabla 4. Presupuesto inicial – Presupuesto total**

## 4.4. Estimación de costes mediante COCOMO

Para realizar una mejor estimación del presupuesto, se utiliza el **MO**delo **CO**nstructivo de **CO**stes (**COCOMO**), el cual se basa en la cantidad de líneas de código del proyecto, y una serie de factores de complejidad. Se elige el submodelo **intermedio** debido a que se dispone de una especificación de requisitos, este submodelo requerirá aplicar 15 factores de coste.

Para el desarrollo de la aplicación se estima una cantidad de **4 KLDC** (miles de líneas de código), esto define el tipo de COCOMO a seguir como **Orgánico**, ya que el tamaño del proyecto se encuentra en el rango entre 1KLDC < **4KLDC** < 50KLDC.

La siguiente tabla (**Tabla 5**), contiene los *15 factores de coste*, necesarios para la obtención de los factores de complejidad:

| FACTORES DE COSTE | VALOR | | | | | |
|---|---|---|---|---|---|---|
| | Muy bajo | Bajo | Medio | Alto | Muy alto | Extra |
| **Factores software** | | | | | | |
| **Fiabilidad requerida** | 0,75 | 0,88 | 1,00 | 1,15 | **1,4** | |
| **Tamaño de la Base de Datos** | | 0,94 | 1,00 | 1,08 | 1,16 | |
| **Complejidad del software** | 0,70 | 0,85 | 1,00 | **1,15** | 1,30 | 1,65 |
| **Factores hardware** | | | | | | |
| **Restricciones de rendimiento en tiempo de ejecución** | | | 1,00 | 1,11 | 1,30 | **1,66** |
| **Restricciones de memoria** | | | 1,00 | **1,06** | 1,21 | 1,56 |
| **Volatilidad del entorno de la máquina virtual** | | 0,87 | **1,00** | 1,15 | 1,30 | |
| **Tiempo de respuesta** | | 0,87 | 1,00 | **1,07** | | |
| **Factores de personal** | | | | | | |
| **Capacidad de los analistas** | 1,46 | 1,19 | **1,00** | 0,86 | 0,71 | |
| **Experiencia con el tipo de aplicación** | 1,29 | 1,13 | 1,00 | **0,91** | 0,82 | |
| **Experiencia con el hardware** | 1,21 | 1,10 | 1,00 | **0,90** | | |
| **Experiencia con el lenguaje de programación** | 1,14 | 1,07 | 1,00 | **0,95** | | |
| **Capacidad de los programadores** | 1,42 | 1,17 | 1,00 | **0,86** | 0,70 | |
| **Factores del proyecto** | | | | | | |
| **Técnicas modernas de programación** | 1,24 | 1,10 | 1,00 | **0,91** | 0,82 | |
| **Utilización de herramientas software** | 1,24 | 1,10 | 1,00 | **0,91** | 0,83 | |
| **Restricciones en la planifica-ción temporal del desarrollo** | 1,23 | 1,08 | 1,00 | **1,04** | 1,10 | |

**Tabla 5. Factores de coste COCOMO**

Una vez definidos los factores de ajuste, se calcula el **factor de ajuste del esfuerzo (m(X))** que es el productorio de los 15 factores:

$$\mathbf{m(X)} = \prod_{k=1}^{n} factork = 1{,}4 * 1{,}15 * 1{,}66 * 1{,}06 * 1{,}00 * 1{,}07 * 1{,}00 * 0{,}91 * 0{,}90 * 0{,}95 * 0{,}86 * 0{,}91 * 0{,}91 * 1{,}04 = 1{,}74681$$

Debido a que el proyecto es de tipo Orgánico, se siguen los *factores de complejidad* de la tabla que especifica este tipo:

| | a | b | c | d |
|---|---|---|---|---|
| **Orgánico** | **3,2** | **1,05** | **2,5** | **0,38** |
| **Semi-acoplado** | 3,0 | 1,12 | 2,5 | 0,35 |
| **Empotrado** | 2,8 | 1,20 | 2,5 | 0,32 |

**Tabla 6. Factores de complejidad COCOMO**

Estos factores se utilizan para calcular el **esfuerzo (E)**:

$$\mathbf{E} = a * KLDC^{b} * \mathrm{m(X)} = 3{,}2 * 4^{1{,}05} * 1{,}74681 = 23{,}96\,\boldsymbol{\frac{personas}{mes}}.$$

El **tiempo de desarrollo (TD)** equivale a:

$$\mathbf{TD} = \mathrm{c} * E^{d} = 2{,}5 * 23{,}96^{0{,}38} = 8{,}36\,\boldsymbol{meses}.$$

Finalmente se calcula el **número medio de personas** necesarias para realizar el proyecto:

$$\mathbf{Nº\ medio\ de\ personas} = \frac{\mathrm{E}}{\mathrm{TD}} = \frac{\frac{23{,}96 personas}{mes}}{8{,}36 meses} \approx 2{,}86\,\boldsymbol{personas}.$$

Según la estimación mediante COCOMO, para la realización del proyecto serán necesarias 3 personas trabajando durante 8 meses. Atendiendo a que el proyecto va a ser desarrollado por solamente una persona, el tiempo se extendería hasta los 8*3 = 24 meses.

Para estimar los costes del proyecto con los resultados del COCOMO, se necesita conocer el total de horas que se va a trabajar, sabiendo que serían 24 meses, 4 semanas por mes, trabajando 5 días a la semana 8h al día:

(24meses*4semanas*5días)*8h = 480días*8h = **3840h**

Se estima el esfuerzo de las distintas categorías de trabajo en la siguiente tabla:

| Categoría | Esfuerzo (%) |
|---|---|
| **Analista** | 18% |
| **Diseñador** | 43% |
| **Programador** | 36% |
| **Documentador** | 3% |
| **TOTAL:** | 100% |

**Tabla 7. Esfuerzo categorías de trabajo**

### 4.4.1. **Estimación costes hardware**

Hardware necesario para el desarrollo del proyecto, se utilizan los datos de la vida media usados en el **Presupuesto inicial**:

- **Ordenador personal**: según el COCOMO se va a hacer uso de él durante 24 meses que son 96 semanas, su uso se calcula:

  100% ---------- 192 semanas
  **x**% ---------- 96 semanas
  **x** = (96*100)/192 = **50%** de uso.

- **Dispositivo móvil**: también va a hacerse uso durante 96 semanas:

  100% ---------- 144 semanas
  **x**% ---------- 96 semanas
  **x** = (96*100)/144 = **66,6%** de uso.

- **Cable microUSB**: el porcentaje de uso sería el **100%**.

- **Wacom Intuos Pen Tableta Gráfica CTL-480S**: se usará durante 15 semanas. El porcentaje de uso de la tableta según el tiempo de desarrollo es:

  100% ---------- 144 semanas
  **x**% ---------- 15 semanas
  **x** = (15*100)/144 = **10,42%** de uso.

- **Conexión a Internet**: se considera un **40%** su uso mensual durante los 24 meses.

- **Impresora HP Deskjet F2280 All-in-One**: para la documentación se requerirá de una impresora, el gasto de tinta se estima de 50€, haciendo uso del **100%**.

| Hardware | Uso (%) | Coste (€) | (Uso * Coste) / 100 |
|---|---|---|---|
| **Ordenador personal portátil: Sony VAIO VPCEH** | 50% | 520 | 260 |
| **Dispositivo móvil: THL W100** | 66,6% | 159 | 105,89 |
| **Cable microUSB** | 100% | 1,45 | 1,45 |
| **Wacom Intuos Pen Tableta Gráfica CTL-480S** | 10,42% | 60,97 | 6,35 |
| **Conexión a Internet** | 40% 24 meses | 42*24 | 1008 |
| **Impresora HP Deskjet F2280** | 100% | 50 | 50 |
| | | **TOTAL (€):** | **1431,69** |

**Tabla 8. Estimación costes hardware**

### 4.4.2. **Estimación costes software**

Software necesario para el desarrollo del proyecto:

- **Adobe Photoshop CS3**: la licencia mensual de este software es de 19,99€, estimando su uso de 12 semanas, que son 3 meses, se tendría que adquirir la licencia 3 veces, es decir, el **100%**.

- **Microsoft Office Professional Plus 2013**: el coste de la licencia de este paquete es de 539€ y se usa el mismo porcentaje que el ordenador donde se instala, **50%**.

| Software | Uso (%) | Coste (€) | (Uso * Coste) / 100 |
|---|---|---|---|
| **Unity3D** | - | - | 0 |
| **Android SDK** | - | - | 0 |
| **Unity Remote** | - | - | 0 |
| **Monodevelop** | - | - | 0 |
| **Adobe Photoshop CS3** | 100% | 19,99*3 | 59,97 |
| **Spriter** | - | - | 0 |
| **Bfxr** | - | - | 0 |
| **Dia** | - | - | 0 |
| **StarUML** | - | - | 0 |
| **Draw.io** | - | - | 0 |
| **Microsoft Office Professional Plus 2013** | 50% | 539 | 269,5 |
| **Google Drive** | - | - | 0 |
| | | **TOTAL (€):** | **329,47** |

**Tabla 9. Estimación costes software**

### 4.4.3. **Presupuesto de desarrollo**

Contando que se ha estimado la duración del proyecto 24 meses, trabajando todos los días de la semana, exceptuando los sábados y domingos, 8 horas al día:

480 días*8h/día = **3840h**.

Aprovechando los cálculos realizados en la **Tabla7** sobre el esfuerzo de cada categoría de trabajo, se puede conocer las horas estimadas de cada uno:

- **Analista**: 18% de 3840h = 691,2h.
- **Diseñador**: 43% de 3840h = 1651,2h.
- **Programador**: 36% de 3840h = 1382,4h.
- **Documentador**: 3% de 3840h = 115,2h.

| | Tiempo | Coste (€) |
|---|---|---|
| **Analista** | 691,2h | 10€/h |
| **Diseñador** | 1651,2h | 10€/h |
| **Programador** | 1382,4h | 12€/h |
| **Documentador** | 115,2h | 8€/h |
| **TOTAL (€):** | **40934,4** | |

**Tabla 10. Estimación costes de desarrollo**

### 4.4.4. Presupuesto total

El presupuesto total es la suma de los presupuestos de hardware, software y desarrollo y otros gastos referentes a material de oficina, DVDs, etc..:

| | Coste (€) |
|---|---|
| **Hardware** | 1431,69 |
| **Software** | 329,47 |
| **Desarrollo** | 40934,4 |
| **Otros** | 50 |
| **TOTAL (€):** | **42745.56** |

**Tabla 11. Estimación costes total**

# 4.5. Coste real

Durante el desarrollo del proyecto, ha habido varias fases:

- **Análisis**: fase de análisis sobre qué hacer, cómo hacerlo, con qué herramientas, limitaciones y objetivos a conseguir con este proyecto. Esta fase es realizada por el analista del proyecto.

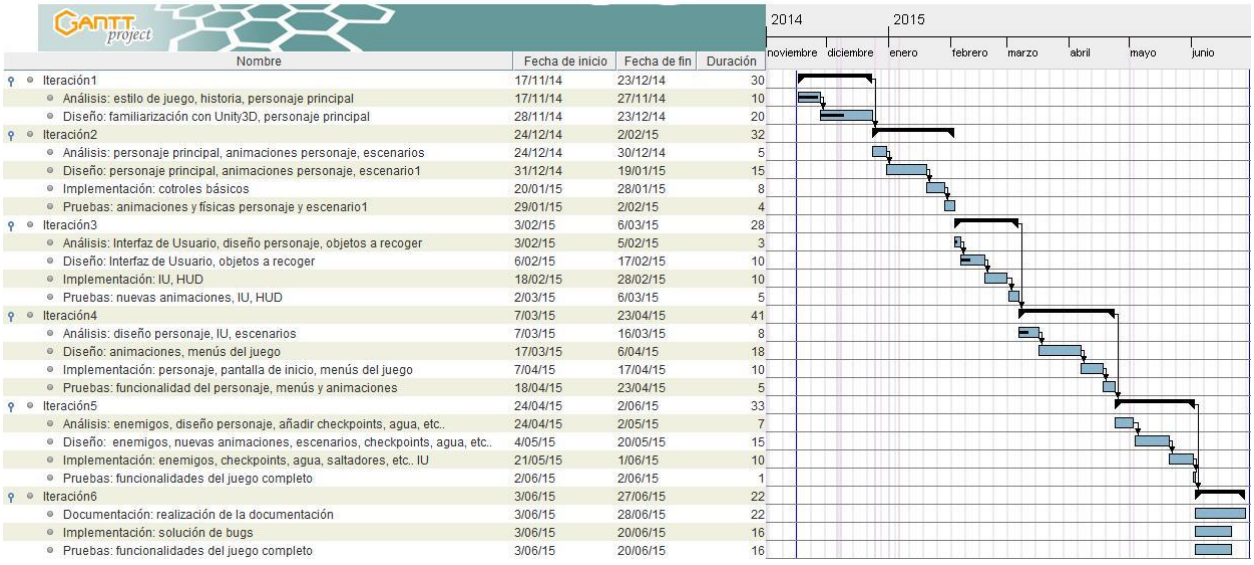
- **Diseño**: a la vez que aprendía Unity3D, diseñaba lo necesario para el juego, tanto el personaje principal como los enemigos, objetos y plataformas del juego. Esta fase es realizada por el diseñador del proyecto.

- **Implementación**: desarrollo del videojuego CocoPunch. Programación de las funciones y niveles del juego. Esta fase es realizada por el programador del proyecto.

- **Pruebas**: fase en la que se comprueba las funcionalidades implementadas. Esta fase es realizada por el programador del proyecto.

- **Documentación**: fase en la que se recoge lo realizado, cómo se ha realizado y demás información necesaria para entender el objetivo del TFG y el trabajo que ha requerido. Esta fase es realizada por el documentador del proyecto.

Cada iteración comprende las fases indicadas, como se muestra en el *Diagrama de Gantt* (**Diagrama2**).



| Nombre | Fecha de inicio | Fecha de fin | Duración |
|---|---|---|---|
| Iteración1 | 17/11/14 | 23/12/14 | 30 |
| Análisis: estilo de juego, historia, personaje principal | 17/11/14 | 27/11/14 | 10 |
| Diseño: familiarización con Unity3D, personaje principal | 28/11/14 | 23/12/14 | 20 |
| Iteración2 | 24/12/14 | 2/02/15 | 32 |
| Análisis: personaje principal, animaciones personaje, escenarios | 24/12/14 | 30/12/14 | 5 |
| Diseño: personaje principal, animaciones personaje, escenario1 | 31/12/14 | 19/01/15 | 15 |
| Implementación: cotroles básicos | 20/01/15 | 28/01/15 | 8 |
| Pruebas: animaciones y físicas personaje y escenario1 | 29/01/15 | 2/02/15 | 4 |
| Iteración3 | 3/02/15 | 6/03/15 | 28 |
| Análisis: Interfaz de Usuario, diseño personaje, objetos a recoger | 3/02/15 | 5/02/15 | 3 |
| Diseño: Interfaz de Usuario, objetos a recoger | 6/02/15 | 17/02/15 | 10 |
| Implementación: IU, HUD | 18/02/15 | 28/02/15 | 10 |
| Pruebas: nuevas animaciones, IU, HUD | 2/03/15 | 6/03/15 | 5 |
| Iteración4 | 7/03/15 | 23/04/15 | 41 |
| Análisis: diseño personaje, IU, escenarios | 7/03/15 | 16/03/15 | 8 |
| Diseño: animaciones, menús del juego | 17/03/15 | 6/04/15 | 18 |
| Implementación: personaje, pantalla de inicio, menús del juego | 7/04/15 | 17/04/15 | 10 |
| Pruebas: funcionalidad del personaje, menús y animaciones | 18/04/15 | 23/04/15 | 5 |
| Iteración5 | 24/04/15 | 2/06/15 | 33 |
| Análisis: enemigos, diseño personaje, añadir checkpoints, agua, etc.. | 24/04/15 | 2/05/15 | 7 |
| Diseño: enemigos, nuevas animaciones, escenarios, checkpoints, agua, etc.. | 4/05/15 | 20/05/15 | 15 |
| Implementación: enemigos, checkpoints, agua, saltadores, etc.. IU | 21/05/15 | 1/06/15 | 10 |
| Pruebas: funcionalidades del juego completo | 2/06/15 | 2/06/15 | 1 |
| Iteración6 | 3/06/15 | 27/06/15 | 22 |
| Documentación: realización de la documentación | 3/06/15 | 28/06/15 | 22 |
| Implementación: solución de bugs | 3/06/15 | 20/06/15 | 16 |
| Pruebas: funcionalidades del juego completo | 3/06/15 | 20/06/15 | 16 |



**Diagrama 2. Diagrama de Gantt coste real**

- **Iteración1**:
  - **Análisis**: definir el estilo del juego, estética e historia, así como el boceto del personaje principal.
  - **Diseño**: se había planificado familiarizarse con Unity3D y el diseño del personaje principal, pero se extendió el tiempo de diseño del personaje.
  - **Implementación**: se había planificado importar a Unity3D el personaje principal e implementar los controles básicos de mover al personaje, pero en esta iteración la fase de implementación no se realizó debido al retraso del diseño del personaje y las complicaciones que se encontraron para ello.
  - **Pruebas**: las pruebas en este caso no son del juego sino del diseño de éste.

- **Iteración2**:
  - **Análisis**: análisis de las animaciones del personaje y los componentes de los niveles. Debido a no haber acabado el diseño del personaje, se decide terminar con ello, y se analiza el coste de tiempo de realizar las animaciones, en un principio se decide realizarlas frame-by-frame, luego veremos que no fue una buena decisión.
  - **Diseño**: finalmente se termina el diseño del personaje y se continúa con las animaciones dibujándolas frame-by-frame, lo que lleva mucho tiempo y mucha dificultad al ser la primera vez que se hace y no tener conocimiento sobre la forma de hacerlo. También se diseña el suelo de los niveles, dividiéndolo en hierba (grass) y tierra (dirt).
  - **Implementación**: aprovechando los Standard Assets de Unity3D, añadí al personaje un controlador básico para poder probar las animaciones sobre el terreno.
  - **Pruebas**: pruebas de las animaciones del personaje y las físicas con los bloques del nivel.

- **Iteración3**:
  - **Análisis**: en esta iteración se planificó el realizar la Interfaz de Usuario del juego, debido a que este tema no requería mucho tiempo, se analizó el diseño del personaje y se decidió modificarlo. A la vez se analiza los objetos a diseñar, se decide crear Mangos para recoger y Vidas para aumentar las vidas en el juego. Debido a la poca ayuda que hay programando en Javascript, se decide programar en C#.
  - **Diseño**: el diseño de la Interfaz de Usuario fue sencillo debido a la idea clara de los colores y estilo de esta. Los mangos y vidas no tuvieron complicación tampoco.
  - **Implementación**: hubo problemas al implementar la Interfaz de Usuario, pues Unity3D no disponía de ningún sistema para ello, y al modificar la resolución de la pantalla, la IU no se escalaba. Este tema se dejó para más adelante. Se

implementó la funcionalidad de Mango y Vida, pero hubo problemas a la hora de actualizar el HUD debido a la decisión de utilizar una fuente personalizada.

- **Pruebas**: se comprobó que funcionaba la funcionalidad de recoger Mango y recoger Vida, aunque el problema de actualizar el HUD persistió.

- **Iteración4**:
  - **Análisis**: se analizó la situación del proyecto y se decidió que la historia se plantearía al final, o en futuras mejoras. Debido a este cambio, se aprovechó este mes para mejorar el diseño del personaje, la IU y el escenario.
  - **Diseño**: el diseño de Coco cambió bastante, y debido a la dificultad de dibujar las animaciones frame-by-frame, se buscó y encontró el software Spriter que permite animar objetos 2D. Para ello diseñé el personaje por partes, y la creación de animaciones se hizo más amena. Diseñé los menús de HUD y Pausa, aprovechando la salida de Unity 4.6 que incluía un nuevo sistema de GUI, sencillo y muy potente, que solucionaba el problema de la escalabilidad de resoluciones.
  - **Implementación**: se sigue el tutorial de 3DBuzz . Se implementó la pantalla de inicio del juego y los menús de Ajustes, Salir y Pausa.
  - **Pruebas**: se prueba la funcionalidad del personaje, los menús y sus animaciones.


- **Iteración5**:
  - **Análisis**: como el diseño de los enemigos está claro y debido a la experiencia adquirida no va a llevar mucho tiempo, se decide modificar nuevamente el diseño del personaje y sus animaciones, y mejorar el juego en general, para poder realizar la documentación en el siguiente paso.
  - **Diseño**: se diseña el nuevo aspecto del personaje y de los enemigos, se tenía previsto realizar zonas de diálogo para ayudar al jugador en su experiencia, pero se decidió crear unos pequeños “amigos” que fuesen quienes nos dieran consejos. Se diseñan distintos componentes de los niveles como bloques, saltadores, checkpoints, etc.. Se diseñan varios niveles para probar el juego.
  - **Implementación**: se añaden funcionalidades al personaje y se crean checkpoints, así como se mejora la IU añadiendo el menú de Selección de niveles. Se

implementan 3 niveles, el primero que introduce al jugador al juego explicando qué se va a encontrar; el segundo ya es un nivel jugable donde encontraremos enemigos y objetos que recoger con el objetivo de llegar al final del nivel, que cargará el siguiente nivel, que ofrecerá distintos retos. Con esto se consigue el objetivo deseado de crear la estructura de un juego, pudiendo añadir tantos niveles como quiera, necesitando sólo añadir los objetos a la escena y añadir un nivel más al menú de selección de nivel.

- **Pruebas**: se prueba el juego, y comprueba que se ha conseguido con éxito el desarrollo del proyecto. Se descubren algunos bugs (errores) que se corrigen o corregirán en el futuro.


- **Iteración6**:
  - **Documentación**: una vez acabada la estructura del juego e implementadas las funcionalidades deseadas, así como el diseño de varios niveles, se pasa a realizar la documentación necesaria del proyecto. Siguiendo otros proyectos y la estructura que debe tener la documentación, se define el Índice y divide en capítulos con diferentes secciones. realización de la documentación del proyecto.
  - **Implementación**: a la vez que se realiza la documentación, se solucionan varios bugs encontrados en el juego.
  - **Pruebas**: mientras se documenta el proyecto se realizan pruebas del juego para comprobar su correcto funcionamiento.

Como se puede comprobar, la planificación no fue perfecta, pero en gran medida se ha seguido, exceptuando al principio por las dificultades encontradas, debido al desconocimiento del tema, y la iteración sobre la historia que se canceló. Estos cambios y atrasos han hecho que el final del proyecto no fuese el 18/06 sino el 28/06.

### 4.5.1. **Coste hardware**

Hardware necesario para el desarrollo del proyecto:

- **Ordenador personal**: se ha hecho uso de él durante 30 semanas, su porcentaje de uso se calcula:

  100% ---------- 192 semanas
  **x**% ---------- 30 semanas
  **x** = (30*100)/192 = **15,6%** de uso.

- **Dispositivo móvil**: se ha hecho uso de él durante 30 semanas:

  100% ---------- 144 semanas
  **x**% ---------- 30 semanas
  **x** = (30*100)/144 = **20,8%** de uso.

- **Cable microUSB**: el porcentaje de uso se mantiene al **50%**.

- **Wacom Intuos Pen Tableta Gráfica CTL-480S**: ha sido usada durante la fase de Diseño, en las partes de uso de Adobe Photoshop, en total 6 semanas. El porcentaje de uso de la tableta según el tiempo de desarrollo es:

  100% ---------- 144 semanas
  **x**% ---------- 6 semanas
  **x** = (6*100)/144 = **4,16%** de uso.

- **Conexión a Internet**: se confirma el **40%** su uso mensual.

- **Impresora HP Deskjet F2280 All-in-One**: debido a que la impresión va a ser porterior a la elaboración del documento, se sigue estimando el gasto de tinta se de 50€, haciendo uso del **100%**.

| Hardware | Uso (%) | Coste (€) | (Uso * Coste) / 100 |
|---|---|---|---|
| **Ordenador personal portátil: Sony VAIO VPCEH** | 15,6% | 520 | 81,12 |
| **Dispositivo móvil: THL W100** | 20,8% | 159 | 33,072 |
| **Cable microUSB** | 50% | 1,45 | 0,725 |
| **Wacom Intuos Pen Tableta Gráfica CTL-480S** | 4,16% | 60,97 | 2,54 |
| **Conexión a Internet** | 40% 7 meses | 42*7 | 117,6 |
| **Impresora HP Deskjet F2280** | 100% | 50 | 14 |
| | | **TOTAL (€):** | **249,06** |

**Tabla 12. Coste real – Coste hardware**

### 4.5.2. **Coste software**

Software necesario para el desarrollo del proyecto:

| Software | Uso (%) | Coste (€) | (Uso * Coste) / 100 |
|---|---|---|---|
| **Unity3D** | - | - | 0 |
| **Android SDK** | - | - | 0 |
| **Unity Remote** | - | - | 0 |
| **Monodevelop** | - | - | 0 |
| **Adobe Photoshop CS3** | 100% | 19,99 | 19,99 |
| **Spriter** | - | - | 0 |
| **Dia** | - | - | 0 |
| **StarUML** | - | - | 0 |
| **Draw.io** | - | - | 0 |
| **Microsoft Office Professional Plus 2013** | 539% | 14,6 | 78,694 |
| **Google Drive** | - | - | 0 |
| | | **TOTAL (€):** | **98,684** |

**Tabla 13. Coste real – Coste software**

### 4.5.3. **Coste de desarrollo**

Costes referidos al desarrollo del proyecto por parte de un Ingeniero Informático de Servicios y Aplicaciones.

Se han requerido distintas categorías de trabajo:

- **Analista**: encargado de la fase de *Análisis* del proyecto.
- **Diseñador**: encargado de la fase de *Diseño* del proyecto.
- **Programador**: encargado de la fase de *Implementación* y *Pruebas* del proyecto.
- **Documentador**: encargado de la fase de *Documentación* del proyecto.

Contando que se ha trabajado durante 7 meses y 10 días, que son 30 semanas, 5 días a la semana, 8 horas al día, excepto en la última iteración, que, a la vez que se realizaba la documentación, se corregían errores del juego y se probaba su funcionamiento, dividiendo el tiempo 4h documentación, 3h implementación y pruebas:

(164días*8h)+[(22días*4h)+(16días*3h)+(16días*3h)] = **1452h**.

Al dividir el proyecto en iteraciones, cada fase ha sido realizada por un especialista, permitiendo conocer mediante el Diagrama de Gantt (**Diagrama2**) las horas realizadas por cada uno.

| | Tiempo | Coste (€) |
|---|---|---|
| **Analista** | 264h | 10€/h |
| **Diseñador** | 624h | 10€/h |
| **Programador** | 520h | 12€/h |
| **Documentador** | 44h | 8€/h |
| **TOTAL (€):** | **15472**€. | |

**Tabla 14. Coste real – Coste de desarrollo**

### 4.5.4. Coste total

| | Coste (€) |
|---|---|
| **Hardware** | 249,06 |
| **Software** | 98,684 |
| **Desarrollo** | 15472 |
| **TOTAL (€):** | **15819,744** |

**Tabla 15. Coste real – Coste total**

## 4.6. Conclusiones

En el presupuesto inicial se planificó la duración del proyecto de 7 meses, trabajando 5 días a la semana, 8 horas al día. Finalmente debido a la magnitud del proyecto tuvo que atrasarse 10 días la entrega. Es por ello la diferencia entre el presupuesto inicial y el coste real, ya que el sueldo a pagar y el uso del software aumenta. El número total de LDC es de 5480 en vez de 4K como se estimó en el COCOMO.

Respecto a la gran diferencia entre la estimación del presupuesto mediante COCOMO, y los otros dos resultados, se debe al hecho de que la estimación resultó la necesidad de 3 personas trabajando durante 8 meses, casi lo que ha llevado en la realidad, pero al tratarse de una sola persona la encargada, se dispara a los 24 meses, plazo imposible de seguir, aumentando el uso del hardware y software, y el sueldo del personal, hasta multiplicar casi por 3 al coste real.

| TIPO | COSTE TOTAL (€) |
|---|---|
| **Presupuesto inicial:** | 14981,479 |
| **Estimación de costes mediante COCOMO:** | 42745.56 |
| **Coste real:** | **15819,744** |

**Tabla 16. Comparación resultados del coste total**

# Capítulo 5
# Análisis

En este capítulo se realiza la especificación del comportamiento y funcionalidades del sistema, mediante distintos modelos. Se ha escogido el modelado UML debido a ser con el que se ha aprendido y familiarizado a describir lo que debe hacer el sistema, y por ser el más utilizado actualmente.

Las tablas podrán contener distintos apartados:

- **ID**: identificador. Se nombra según *secuencias numeradas* ID[S]-XY donde ID es el nombre del identificador (ACT, RQ, CU), S es el campo opcional (RQF, RQNF, etc..), y un valor ordinal asignado incrementalmente con cada número de tabla de dicho identificador.
- **Nombre**: identifica el contenido.
- **Versión**: número de versión.
- **Actor**: usuario que utiliza el contenido.
- **Disparador**: evento que inicia el contenido.
- **Dependencias**: relaciones con otras tablas.
- **Precondiciones**: condiciones previas que deben cumplirse para poder utilizarse.
- **Prioridad**: indica la importancia, se especifica como Baja/Media/Alta/Muy Alta.
- **Estado**: estado en el que se encuentra el contenido, se especifica como Implementado/No implementado.
- **Frecuencia de uso**: indica la utilización, se especifica como Baja/Media/Alta/Muy Alta.
- **Flujo normal**: secuencia de pasos que sigue el usuario al utilizar el contenido, se especifica como Paso (número)/Acción.
- **Flujo alternativo**: secuencia alternativa de pasos que sigue el usuario al utilizar el contenido, se especifica como Paso (número)/Acción.
- **Postcondiciones**: condiciones que modifica el uso del contenido.
- **Excepciones**: excepciones que pueden aparecer durante el uso del contenido.
- **Descripción**: identifica el contenido de forma detallada.
- **Otra información**: otra información necesaria.

## 5.1. Identificación de actores

En esta sección se identifica a los actores a los que va dirigido el producto.

Debido a la que el juego no va a contener perfiles, sino que todos los usuarios serán jugadores, existe un único actor.

| ID y Nombre: | ACT-01 Jugador |
|---|---|
| Versión: | v1 |
| Descripción: | Usuario principal. Jugador de la aplicación. |
| Otra información: | |

**Tabla 17. ACT-01 Jugador**

## 5.2. Objetivos

En esta sección se identifican los objetivos a conseguir. Para definir los objetivos se utilizará la siguiente tabla:

| ID y Nombre: | OBJ-XY |
|---|---|
| Versión: | *Número de versión.* |
| Descripción: | *Descripción detallada del contenido.* |
| Prioridad: | *Importancia de implementación.* |
| Estado: | *Implementado/No implementado.* |
| Otra información: | *Comentarios.* |

**Tabla 18. Ejemplo de tabla Objetivos**

| ID y Nombre: | OBJ-01 Diseño del videojuego |
|---|---|
| Versión: | v1 |
| Descripción: | Implementación de la estructura básica de un videojuego, permitiendo crear distintos niveles con enemigos, objetos que recoger, etc.. |
| Prioridad: | Muy Alta. |
| Estado: | Implementado. |
| Otra información: | |

**Tabla 19. OBJ-01 Diseño del videojuego**

| ID y Nombre: | OBJ-02 Creación de la Interfaz de Usuario Gráfica |
|---|---|
| Versión: | v1 |
| Descripción: | Implementación de menús para que el jugador pueda interactuar con la interfaz del juego, permitiendo pausar el juego, gestionar la música y sonido del juego, y salir de la aplicación correctamente. |
| Prioridad: | Muy Alta. |
| Estado: | Implementado. |
| Otra información: | |

**Tabla 20. OBJ-02 Creación de la GUI**

| ID y Nombre: | OBJ-03 Creación del sonido del juego |
|---|---|
| Versión: | v1 |
| Descripción: | Implementación de la música y efectos de sonido del videojuego. |
| Prioridad: | Media. |
| Estado: | Implementado. |
| Otra información: | La música será obtenida de otra fuente, manteniendo siempre los derechos de autor. |

**Tabla 21. OBJ-03 Creación del sonido del juego**

| ID y Nombre: | OBJ-04 Programación de las funcionalidades del videojuego |
|---|---|
| Versión: | v1 |
| Descripción: | Implementación de la funcionalidad del videojuego, permitiendo al jugador mover al personaje por los niveles, actualizar el HUD que muestre las vidas y mangos, etc.. |
| Prioridad: | Muy Alta. |
| Estado: | Implementado. |
| Otra información: | |

**Tabla 22. OBJ-04 Programación de las funcionalidades del videojuego**

| ID y Nombre: | OBJ-05 Desarrollo Interfaz Artificial (IA) |
|---|---|
| Versión: | v1 |
| Descripción: | Implementación de IA para controlar el comportamiento de los personajes no controlados por el jugador (enemigos). |
| Prioridad: | Alta. |
| Estado: | Implementado. |
| Otra información: | |

**Tabla 23. OBJ-05 Desarrollo IA**

| ID y Nombre: | OBJ-06 Creación de distintos niveles |
|---|---|
| Versión: | v1 |
| Descripción: | Diseño e implementación de distintos niveles que permitan mostrar lo desarrollado. |
| Prioridad: | Media. |
| Estado: | Implementado. |
| Otra información: | |

**Tabla 24. OBJ-06 Creación de distintos niveles**

| ID y Nombre: | OBJ-07 Guardar progreso |
|---|---|
| Versión: | v1 |
| Descripción: | Implementación del guardado/cargado de datos relativos al progreso del juego, permitiendo seguir jugando desde donde se estaba antes de cerrar la aplicación, no perdiendo tu progreso. |
| Prioridad: | Media. |
| Estado: | No implementado. |
| Otra información: | |

**Tabla 25. OBJ-07 Guardar progreso**

| ID y Nombre: | OBJ-08 Creación modo historia |
|---|---|
| Versión: | v1 |
| Descripción: | Implementación de un modo historia con imágenes ilustrativas de lo que ha pasado y que indique el objetivo del juego (recuperar El Mango de Oro). |
| Prioridad: | Media. |
| Estado: | No implementado. |
| Otra información: | |

**Tabla 26. OBJ-08 Creación modo historia**

## 5.3. Especificación de Requisitos

En esta sección se describen los requisitos de la aplicación, dando a conocer las funcionalidades a desarrollar y la información necesaria para el conocimiento del videojuego.

Para definir los requisitos se utilizará la siguiente tabla:

| ID y Nombre: | RQ[S]-XY |
|---|---|
| Versión: | *Número de versión.* |
| Descripción: | *Descripción detallada del contenido.* |
| Dependencias: | *Requisitos necesarios para su funcionamiento.* |
| Prioridad: | *Importancia de implementación.* |
| Frecuencia de uso: | *Uso del contenido.* |
| Otra información: | *Comentarios.* |

**Tabla 27. Ejemplo de tabla de Requisitos**

### 5.3.1. **Requisitos Funcionales**

Funcionalidades que debe tener la aplicación.

| ID y Nombre: | RQF-01 Seguir al jugador con la Cámara |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá seguir al jugador con la cámara principal del juego. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 28. RQF-01 Seguir al jugador con la Cámara**

| ID y Nombre: | RQF-02 Realizar Parallax Scrolling |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder mover los distintos componentes del fondo del juego a diferentes velocidades, añadiendo profundidad. |
| Dependencias: | RQF-01 |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 29. RQF-02 Realizar Parallax Scrolling**

| ID y Nombre: | RQF-03 Tener vidas |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá añadir vidas al jugador. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | El jugador comenzará con 2 vidas. |

**Tabla 30. RQF-03 Tener vidas**

| ID y Nombre: | RQF-04 Recibir daño |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá añadir la posibilidad de recibir daño al jugador. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | Al recibir daño, se restará el daño ocasionado, en caso de ser mayor que la vida del jugador, morirá perdiendo una vida del juego. |

**Tabla 31. RQF-04 Recibir daño**

| ID y Nombre: | RQF-05 Mover jugador a la izquierda |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder mover al jugador a la izquierda. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 32. RQF-05 Mover jugador a la izquierda**

| ID y Nombre: | RQF-06 Mover jugador a la derecha |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder mover al jugador a la derecha. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 33. RQF-06 Mover jugador a la derecha**

| ID y Nombre: | RQF-07 Saltar |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer saltar al jugador. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 34. RQF-07 Saltar**

| ID y Nombre: | RQF-08 Golpear |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer golpear al jugador. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 35. RQF-08 Golpear**

| ID y Nombre: | RQF-09 Saltar en la pared |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer saltar al jugador, cuando este se encuentre pegado a una pared. |
| Dependencias: | RQF.07 |
| Prioridad: | Baja. |
| Frecuencia de uso: | Baja. |
| Otra información: | El jugador podrá dar un salto cada vez que toque la pared. |

**Tabla 36. RQF-09 Saltar en la pared**

| ID y Nombre: | RQF-10 Morir |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer morir al jugador. |
| Dependencias: | RQF-03 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | En caso de no disponer de vidas, el sistema deberá mostrar el menú de GameOver. |

**Tabla 37. RQF-10 Morir**

| ID y Nombre: | RQF-11 Deslizarse por la pared |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que el jugador se deslice por la pared cuando éste esté pegado a ella y siga presionando el botón de moverse a la izquierda. |
| Dependencias: | RQF-05, RQF-06 |
| Prioridad: | Baja. |
| Frecuencia de uso: | Baja. |
| Otra información: | |

**Tabla 38. RQF-11 Deslizarse por la pared**

| ID y Nombre: | RQF-12 Colisionar con objetos |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que determinados objetos colisionen entre sí, generando eventos. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | Determinados objetos son el jugador, los enemigos, las plataformas, mangos, etc.. |

**Tabla 39. RQF-12 Colisionar con objetos**

| ID y Nombre: | RQF-13 Gestionar animaciones |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder gestionar las animaciones de los distintos objetos, actualizando sus estados cuando se cumpla la condición que requiera. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 40. RQF-13 Gestionar animaciones**

| ID y Nombre: | RQF-14 Generar daño |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que un objeto genere daño al colisionar con el jugador. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 41. RQF-14 Generar daño**

| ID y Nombre: | RQF-15 Matar al tocar |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que un objeto mate al jugador al tocarlo. |
| Dependencias: | |
| Prioridad: | Media. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 42. RQF-15 Matar al tocar**

| ID y Nombre: | RQF-16 Atacar |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que determinados objetos ataquen al jugador. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | Determinados objetos son los enemigos. |

**Tabla 43. RQF-16 Atacar**

| ID y Nombre: | RQF-17 Tener vida |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que un objeto tenga vida. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | Objetos que tienen vida son los enemigos o las cajas destructibles. |

**Tabla 44. RQF-17 Tener vida**

| ID y Nombre: | RQF-18 Renacer |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer renacer a un objeto en una determinada posición. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 45. RQF-18 Renacer**

| ID y Nombre: | RQF-19 Cargar nivel |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder cargar un nivel del juego. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 46. RQF-19 Cargar nivel**

| ID y Nombre: | RQF-20 Reproducir música de fondo |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder reproducir música de fondo en el juego. |
| Dependencias: | |
| Prioridad: | Media. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 47. RQF-20 Reproducir música de fondo**

| ID y Nombre: | RQF-21 Reproducir efecto de sonido |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder reproducir un efecto de sonido. |
| Dependencias: | |
| Prioridad: | Media. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 48. RQF-21 Reproducir efecto de sonido**

| ID y Nombre: | RQF-22 Gestionar música |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder activar/desactivar la música del juego. |
| Dependencias: | RQF-20 |
| Prioridad: | Media. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 49. RQF-22 Gestionar música**

| ID y Nombre: | RQF-23 Gestionar efectos de sonido |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder activar/desactivar la reproducción de efectos de sonido del juego. |
| Dependencias: | RQF-21 |
| Prioridad: | Media. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 50. RQF-23 Gestionar efectos de sonido**

| ID y Nombre: | RQF-24 Controlar los límites del nivel |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder establecer los límites del nivel, y gestionar el comportamiento del jugador al colisionar con ellos. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 51. RQF-24 Controlar los límites del nivel**

| ID y Nombre: | RQF-25 Recoger vida |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder añadir vidas al controlador del juego. |
| Dependencias: | RQF-27 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | Al recoger una vida, el HUD se deberá actualizar. |

**Tabla 52. RQF-25 Recoger vida**

| ID y Nombre: | RQF-26 Recoger mango |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder añadir mangos al controlador del juego. |
| Dependencias: | RQF-27 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | Al recoger un mango, el HUD se deberá actualizar. |

**Tabla 53. RQF-26 Recoger mango**

| ID y Nombre: | RQF-27 Actualizar HUD |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder actualizar las vidas y mangos del HUD del juego. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 54. RQF-27 Actualizar HUD**

| ID y Nombre: | RQF-28 Renacer en checkpoint |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer renacer al jugador en un punto determinado del nivel llamado Checkpoint, si este ha pasado por él previamente y dispone de vidas. |
| Dependencias: | RQF-18 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 55. RQF-28 Renacer en checkpoint**

| ID y Nombre: | RQF-29 Abrir menú |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder abrir un menú de IU. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 56. RQF-29 Abrir menú**

| ID y Nombre: | RQF-30 Cerrar menú |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder cerrar un menú de IU. |
| Dependencias: | RQF-29 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | Al cerrar un menú se mostrará el que previamente estaba activado. |

**Tabla 57. RQF-30 Cerrar menú**

| ID y Nombre: | RQF-31 Pausar juego |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder pausar el juego, no permitiendo moverse al jugador, y pausando el tiempo del juego. |
| Dependencias: | RQF-32, RQF-33 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 58. RQF-31 Pausar juego**

| ID y Nombre: | RQF-32 Congelar al jugador |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder congelar al jugador, impidiendo su movimiento. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 59. RQF-32 Congelar al jugador**

| ID y Nombre: | RQF-33 Modificar tiempo de juego |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder modificar el tiempo del juego, pudiendo ralentizarlo, pararlo o acelerarlo. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 60. RQF-33 Modificar tiempo de juego**

| ID y Nombre: | RQF-34 Reanudar juego |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder reanudar el juego saliendo de la Pausa. |
| Dependencias: | RQF-30, RQF-31 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 61. RQF-34 Reanudar juego**

| ID y Nombre: | RQF-35 Reiniciar nivel |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder reiniciar el nivel en el que se está jugando, reiniciando el progreso conseguido. |
| Dependencias: | RQF-30, RQF-31 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 62. RQF-35 Reiniciar nivel**

| ID y Nombre: | RQF-36 Salir del juego |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder salir del juego cerrando correctamente la aplicación. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 63. RQF-36 Salir del juego**

| ID y Nombre: | RQF-37 Generar raycasts |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder generar raycasts, para comprobar si un objeto colisiona con otro. |
| Dependencias: | RQF-12 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 64. RQF-37 Generar raycasts**

| ID y Nombre: | RQF-38 Generar partículas |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder generar partículas. |
| Dependencias: | |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | Partículas son los efectos generados al tocar el suelo, al salir del agua, al recoger un mango, al destruir un enemigo, etc.. |

**Tabla 65. RQF-38 Generar partículas**

| ID y Nombre: | RQF-39 Destruir partículas |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder destruir partículas generadas. |
| Dependencias: | RQF-38 |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 66. RQF-39 Destruir partículas**

| ID y Nombre: | RQF-40 Activar zona de diálogo |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder activar zonas de diálogo donde se mostrarán mensajes de ayuda al jugador. |
| Dependencias: | |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 67. RQF-40 Activar zona de diálogo**

| ID y Nombre: | RQF-41 Mostrar mensaje en zona de diálogo |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder mostrar mensajes en las zonas de diálogo cuando se activen. |
| Dependencias: | RQF-39 |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 68. RQF-41 Mostrar mensaje en zona de diálogo**

| ID y Nombre: | RQF-42 Añadir transición entre escenas |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder añadir pequeñas transiciones basadas en un difuminado negro para indicar que se está cargando alguna parte del juego. |
| Dependencias: | |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 69. RQF-42 Añadir transición entre escenas**

| ID y Nombre: | RQF-43 Simular viento |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder simular viento en árboles y arbustos para dar más vida al juego. |
| Dependencias: | |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 70. RQF-43 Simular viento**

| ID y Nombre: | RQF-44 Destruir plataforma |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que una plataforma se destruya cayendo al vacío a una velocidad determinada, cuando el jugador se sitúe sobre ella durante un periodo de tiempo. |
| Dependencias: | |
| Prioridad: | Baja. |
| Frecuencia de uso: | Baja. |
| Otra información: | |

**Tabla 71. RQF-44 Destruir plataforma**

| ID y Nombre: | RQF-45 Saltador |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que determinados objetos hagan saltar al jugador por los aires cuando este se sitúe sobre ellos. |
| Dependencias: | |
| Prioridad: | Baja. |
| Frecuencia de uso: | Baja. |
| Otra información: | |

**Tabla 72. RQF-45 Saltador**

| ID y Nombre: | RQF-46 Definir camino |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder definir caminos entre distintos puntos, a seguir por determinados objetos. |
| Dependencias: | |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 73. RQF-46 Definir camino**

| ID y Nombre: | RQF-47 Seguir camino |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que determinados objetos sigan un camino definido. |
| Dependencias: | RQF-46 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | Determinados objetos son enemigos o plataformas en movimiento. |

**Tabla 74. RQF-47 Seguir camino**

| ID y Nombre: | RQF-48 Definir camino de proyectil |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder definir caminos entre dos puntos, a seguir por proyectiles. |
| Dependencias: | |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 75. RQF-48 Definir camino de proyectil**

| ID y Nombre: | RQF-49 Seguir camino de proyectil |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que los proyectiles sigan un camino definido. |
| Dependencias: | RQF-48 |
| Prioridad: | Baja. |
| Frecuencia de uso: | Media. |
| Otra información: | |

**Tabla 76. RQF-49 Seguir camino de proyectil**

| ID y Nombre: | RQF-50 Aplastar |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder hacer que el jugador aplaste a determinados enemigos, destruyéndolos. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 77. RQF-50 Aplastar**

### 5.3.2. **Requisitos no Funcionales**

| ID y Nombre: | RQNF-01 Acceso al almacenamiento del dispositivo |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema requerirá de acceso al almacenamiento del dispositivo móvil para instalar la aplicación. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 78. RQNF-01 Acceso al almacenamiento del dispositivo**

| ID y Nombre: | RQNF-02 Almacenamiento mínimo |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema necesitará de 26MB de almacenamiento. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 79. RQNF-02 Almacenamiento mínimo**

#### 5.3.2.1. **Requisitos de Rendimiento**

| ID y Nombre: | RQNFR-01 Respuesta rápida |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá ofrecer un buen rendimiento, cargando y respondiendo a las acciones del usuario en un mínimo periodo de tiempo. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 80. RQNFR-01 Respuesta rápida**

#### 5.3.2.2. **Requisitos de Escalabilidad**

| ID y Nombre: | RQNFE-01 Compatibilidad dispositivos Android |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá poder instalarse en distintas versiones de Android, según lo indicado. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 81. RQNFE-01 Compatibilidad dispositivos Android**

| ID y Nombre: | RQNFE-02 Compatibilidad resoluciones dispositivos |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema deberá ser escalable en cualquier dispositivo móvil, adecuándose a la resolución de pantalla de cada uno. |
| Dependencias: | RQE-01 |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 82. RQNFE-02 Compatibilidad resoluciones dispositivos**

### 5.3.2.3. **Requisitos de Hardware**

| ID y Nombre: | RQNFH-01 Dispositivo Android |
|---|---|
| Versión: | v1 |
| Descripción: | El usuario deberá disponer de un dispositivo Android. |
| Dependencias: | |
| Prioridad: | Alta. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 83. RQNFH-01 Dispositivo Android**

### 5.3.2.4. **Requisitos de Internacionalización**

| ID y Nombre: | RQNFI-01 Idioma |
|---|---|
| Versión: | v1 |
| Descripción: | El sistema soportará el idioma Español. |
| Dependencias: | |
| Prioridad: | Media. |
| Frecuencia de uso: | Alta. |
| Otra información: | |

**Tabla 84. RQNFI-01 Idioma**

# 5.4. Casos de Uso

En esta sección se describe el comportamiento entre el sistema y los actores que harán uso de él, en este caso, mediante una secuencia de iteraciones representadas con modelado de casos de uso UML.

Dado que es un videojuego de 1 jugador para móvil, sólo habrá un actor que es el mismo jugador.

## 5.4.1. Requisitos Diagrama de Casos de Uso

El **Diagrama3** especifica los Casos de Uso del videojuego implementando los objetivos principales.

El **Diagrama4** especifica los Casos de Uso ideales del videojuego, es decir, si se implementaran todas las funcionalidades descritas en los objetivos.

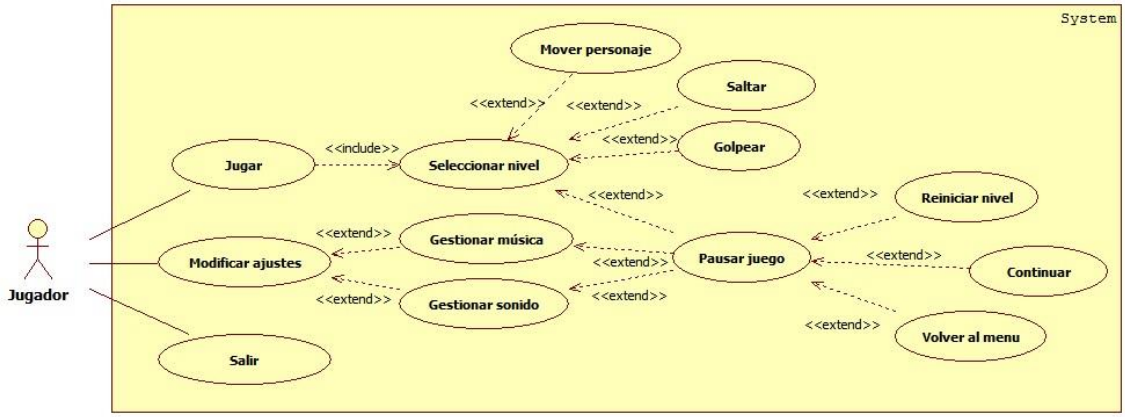



**Diagrama 3. Modelo de Casos de Uso**

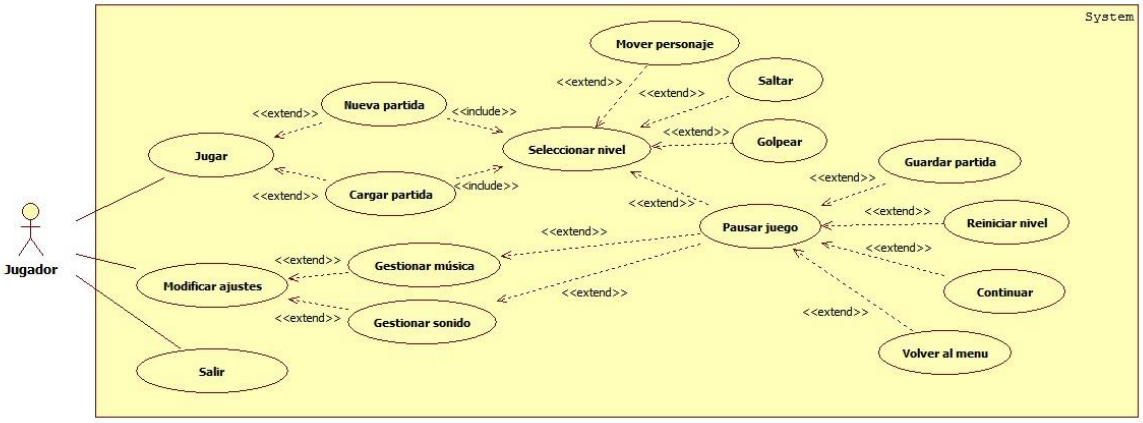



**Diagrama 4. Modelo de Casos de Uso ideal**

La diferencia entre el modelo ideal y el final, es la posibilidad de *Guardar/Cargar* el progreso, que no ha sido posible implementarlo.

### 5.4.2. **Especificación de Casos de Uso**

| **ID y Nombre:** | **CU-01 Jugar** | |
|---|---|---|
| **Actor:** | Jugador. | |
| **Descripción:** | Muestra el menú de selección de nivel. | |
| **Disparador:** | El jugador presiona el botón START. | |
| **Dependencias:** | | |
| **Precondiciones:** | Se debe estar en el menú principal del juego. | |
| **Flujo normal:** | **Paso** | **Acción** |
| | 1 | Presionar el botón START de la pantalla inicial del juego. |
| **Flujo alternativo:** | | |
| **Postcondiciones:** | | |
| **Excepciones:** | | |
| **Prioridad:** | Muy Alta. | |
| **Frecuencia de uso:** | Muy Alta. | |
| **Otra información:** | | |

**Tabla 85. CU-01 Jugar**

| **ID y Nombre:** | **CU-02 Modificar ajustes** | |
|---|---|---|
| **Actor:** | Jugador. | |
| **Descripción:** | Muestra los ajustes de música y sonido del juego. | |
| **Disparador:** | El jugador presiona el botón de AJUSTES. | |
| **Dependencias:** | | |
| **Precondiciones:** | Se debe estar en el menú principal del juego. | |
| **Flujo normal:** | **Paso** | **Acción** |
| | 1 | Presionar el botón AJUSTES de la pantalla inicial del juego. |
| **Flujo alternativo:** | **Paso** | **Acción** |
| | 1 | Pausar el juego. |
| | 2 | Presionar el botón AJUSTES del menú de PAUSA. |
| **Postcondiciones:** | | |
| **Excepciones:** | | |
| **Prioridad:** | Baja. | |
| **Frecuencia de uso:** | Media. | |
| **Otra información:** | | |

**Tabla 86. CU-02 Modificar ajustes**

| ID y Nombre: | CU-03 Salir | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Muestra el menú de SALIR del juego. | |
| Disparador: | El jugador presiona el botón ATRÁS del dispositivo móvil. | |
| Dependencias: | | |
| Precondiciones: | Se debe estar en el menú principal del juego. | |
| Flujo normal: | Paso | Acción |
| | 1 | Presionar el botón ATRÁS del dispositivo móvil. |
| | 2.1 | Confirmar salir del juego. |
| | 2.2 | Cancelar salir del juego. |
| Flujo alternativo: | | |
| Postcondiciones: | Se cierra la aplicación. | |
| Excepciones: | | |
| Prioridad: | Muy Alta. | |
| Frecuencia de uso: | Alta. | |
| Otra información: | | |

**Tabla 87. CU-03 Salir**

| ID y Nombre: | CU-04 Seleccionar nivel | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Carga el nivel seleccionado. | |
| Disparador: | El jugador selecciona un nivel del menú de SELECCIÓN. | |
| Dependencias: | CU-01 | |
| Precondiciones: | Se debe estar en el menú de selección de nivel. | |
| Flujo normal: | Paso | Acción |
| | 1 | Seleccionar el nivel deseado. |
| Flujo alternativo: | | |
| Postcondiciones: | | |
| Excepciones: | El nivel seleccionado no está disponible. | |
| Prioridad: | Alta. | |
| Frecuencia de uso: | Muy Alta. | |
| Otra información: | | |

**Tabla 88. CU-04 Seleccionar nivel**

| ID y Nombre: | CU-05 Mover personaje | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Mueve al personaje en la dirección seleccionada. | |
| Disparador: | El jugador pulsa la flecha de mover al personaje. | |
| Dependencias: | CU-04 | |
| Precondiciones: | Se debe haber seleccionado un nivel.<br>No debe estar pegado a una pared. | |
| Flujo normal: | Paso | Acción |
| | 1 | Seleccionar la flecha de dirección. |
| Flujo alternativo: | | |
| Postcondiciones: | | |
| Excepciones: | El jugador se encuentra contra una pared, impidiendo su movimiento. | |
| Prioridad: | Alta. | |
| Frecuencia de uso: | Muy Alta. | |
| Otra información: | | |

**Tabla 89. CU-05 Mover personaje**

| ID y Nombre: | CU-06 Saltar | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Hace saltar al personaje. | |
| Disparador: | El jugador pulsa el botón A de saltar. | |
| Dependencias: | CU-04 | |
| Precondiciones: | Se debe haber seleccionado un nivel.<br>Debe disponer de saltos. | |
| Flujo normal: | Paso | Acción |
| | 1 | Seleccionar el botón A de saltar. |
| Flujo alternativo: | | |
| Postcondiciones: | | |
| Excepciones: | El jugador no dispone de saltos. | |
| Prioridad: | Alta. | |
| Frecuencia de uso: | Muy Alta. | |
| Otra información: | | |

**Tabla 90. CU-06 Saltar**

| ID y Nombre: | CU-07 Golpear | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Hace golpear al personaje. | |
| Disparador: | El jugador pulsa el botón B de golpear. | |
| Dependencias: | CU-04 | |
| Precondiciones: | Se debe haber seleccionado un nivel. | |
| Flujo normal: | Paso | Acción |
| | 1 | Seleccionar el botón B de golpear. |
| Flujo alternativo: | | |
| Postcondiciones: | | |
| Excepciones: | | |
| Prioridad: | Alta. | |
| Frecuencia de uso: | Muy Alta. | |
| Otra información: | | |

**Tabla 91. CU-07 Golpear**

| ID y Nombre: | CU-08 Pausar juego | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Pausa el juego, impidiendo moverse y mostrando el menú de PAUSA. | |
| Disparador: | El jugador presiona el botón de PAUSA. | |
| Dependencias: | CU-04 | |
| Precondiciones: | Se debe haber seleccionado un nivel. | |
| Flujo normal: | Paso | Acción |
| | 1 | Seleccionar el botón PAUSA del juego. |
| Flujo alternativo: | | |
| Postcondiciones: | | |
| Excepciones: | El jugador está en el menú de GAMEOVER. | |
| Prioridad: | Alta. | |
| Frecuencia de uso: | Media. | |
| Otra información: | | |
| | | |

**Tabla 92. CU-08 Pausar juego**

| ID y Nombre: | CU-09 Reiniciar nivel | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Reinicia el nivel actual, perdiendo el progreso y volviendo al principio. | |
| Disparador: | El jugador selecciona el botón de REINICIAR en el menú de PAUSA. | |
| Dependencias: | CU-06 | |
| Precondiciones: | Se debe estar en el menú de PAUSA. | |
| Flujo normal: | Paso | Acción |
| | 1 | Seleccionar el nivel deseado. |
| Flujo alternativo: | | |
| Postcondiciones: | | |
| Excepciones: | El nivel seleccionado no está disponible. | |
| Prioridad: | Alta. | |
| Frecuencia de uso: | Media. | |
| Otra información: | | |

**Tabla 93. CU-09 Reiniciar nivel**

| ID y Nombre: | CU-10 Continuar | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Despausa el juego, permitiendo moverse, ocultando el menú de PAUSA y mostrando el HUD. | |
| Disparador: | El jugador presiona el botón de CONTINUAR. | |
| Dependencias: | CU-06 | |
| Precondiciones: | Se debe estar en el menú de PAUSA. | |
| Flujo normal: | Paso | Acción |
| | 1 | Seleccionar el botón CONTINUAR del menú de PAUSA. |
| Flujo alternativo: | | |
| Postcondiciones: | | |
| Excepciones: | | |
| Prioridad: | Alta. | |
| Frecuencia de uso: | Media. | |
| Otra información: | | |

**Tabla 94. CU-10 Continuar**

| ID y Nombre: | CU-11 Volver al menú | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Vuelve a la pantalla de inicio del juego. | |
| Disparador: | El jugador presiona el botón de HOME. | |
| Dependencias: | CU-06 | |
| Precondiciones: | Se debe estar en el menú de PAUSA. | |
| Flujo normal: | Paso | Acción |
| | 1 | Seleccionar el botón HOME del menú de PAUSA. |
| Flujo alternativo: | | |
| Postcondiciones: | | |
| Excepciones: | | |
| Prioridad: | Alta. | |
| Frecuencia de uso: | Media. | |
| Otra información: | | |

**Tabla 95. CU-11 Volver al menú**

| ID y Nombre: | CU-12 Gestionar música | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Activa/desactiva la música de fondo del juego. | |
| Disparador: | El jugador presiona el botón de MÚSICA. | |
| Dependencias: | CU-02 | |
| Precondiciones: | Se debe estar en el menú de AJUSTES. | |
| Flujo normal: | Paso | Acción |
| | 1.1 | Presionar el botón de MÚSICA (activar). |
| | 1.2 | Presionar el botón de MÚSICA (desactivar). |
| Flujo alternativo: | | |
| Postcondiciones: | 1.1 Activar la música. | |
| | 1.2 Desactivar la música. | |
| Excepciones: | Ya hay una música activada.<br>La música no existe. | |
| Prioridad: | Baja. | |
| Frecuencia de uso: | Media. | |
| Otra información: | El botón comparte funcionalidad, en caso de estar activado, si se presiona pasa a desactivado y viceversa. | |

**Tabla 96. CU-12 Gestionar música**

| ID y Nombre: | CU-13 Gestionar sonido | |
|---|---|---|
| Actor: | Jugador. | |
| Descripción: | Activa/desactiva el sonido del juego. | |
| Disparador: | El jugador presiona el botón de SONIDO. | |
| Dependencias: | CU-02 | |
| Precondiciones: | Se debe estar en el menú de AJUSTES. O en el menú de PAUSA. | |
| Flujo normal (menú AJUSTES): | Paso | Acción |
| | 1.1 | Presionar el botón de SONIDO (activar). |
| | 1.2 | Presionar el botón de SONIDO (desactivar). |
| Flujo alternativo (menú PAUSA): | Paso | Acción |
| | 1.1 | Presionar el botón de SONIDO (activar). |
| | 1.2 | Presionar el botón de SONIDO (desactivar). |
| Postcondiciones: | 1.1 Activar el sonido. | |
| | 1.2 Desactivar el sonido. | |
| Excepciones: | | |
| Prioridad: | Baja. | |
| Frecuencia de uso: | Media. | |
| Otra información: | El botón comparte funcionalidad, en caso de estar activado, si se presiona pasa a desactivado y viceversa. | |

**Tabla 97. CU-13 Gestionar sonido**

# 5.5. Árbol de características

Con el objetivo de conocer las características principales del sistema, y que cualquier persona que desee conocer sus funcionalidades pueda entenderlo, se realiza un Árbol de Características, sencillo y visual.

| |
|---|
| **CP1**. **Jugar.**<br>**CP3**. **Mover personaje.**<br>**CP5**. mover al personaje a la izquierda.<br>**CP6**. mover al personaje a la derecha.<br>**CP7**. hacer saltar al personaje.<br>**CP8**. hacer golpear al personaje.<br>**CP4**. **Pausar.**<br>**CP9**. reanudar la partida.<br>**CP10**. reiniciar el nivel.<br>**CP11**. salir del juego.<br>**CP2**. **Ajustes.**<br>**CP12**. gestionar música.<br>**CP13**. gestionar sonido. |

**Tabla 98. Características principales**

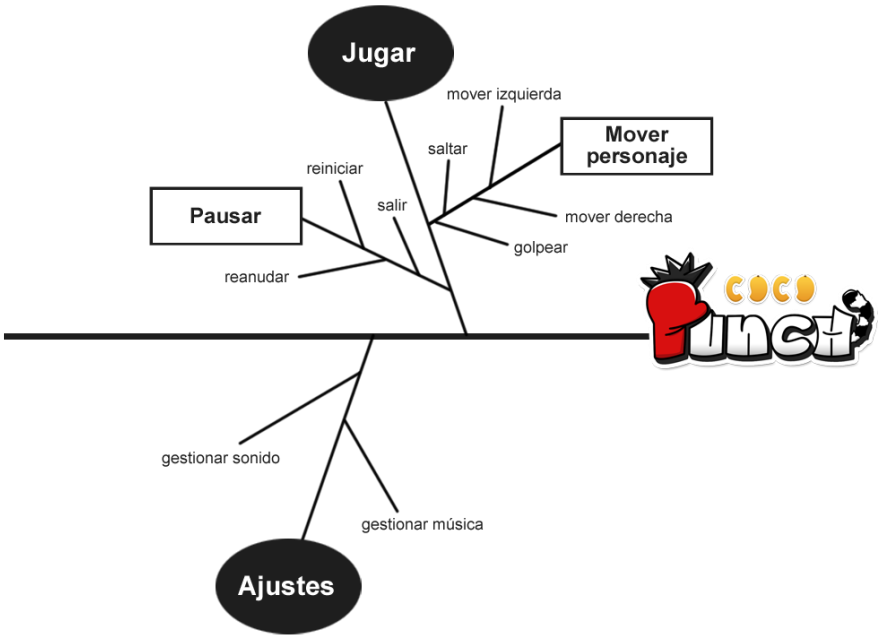



**Figura 8. Árbol de Características**

# 5.6. Diagramas de secuencia

En esta sección se van a describir las secuencias de mensajes que intercambian las clases para realizar una funcionalidad. Se especifica un diagrama de secuencia para las funcionalidades principales del juego.

## Jugar

Este diagrama muestra la secuencia para mostrar los niveles del juego.

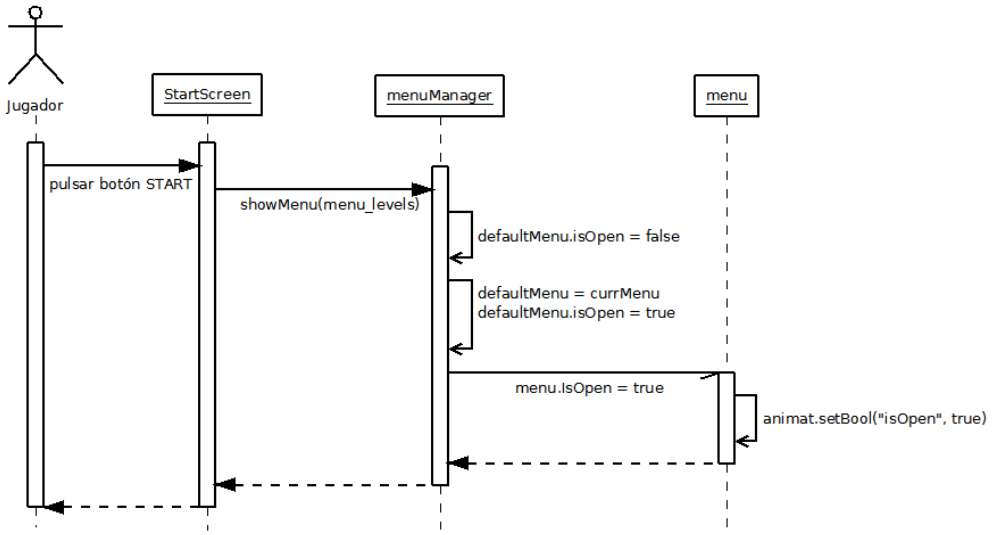



**Diagrama 5. Diagrama de secuencia para Jugar**

## Seleccionar nivel

Este diagrama muestra la secuencia para cargar un nivel del juego.

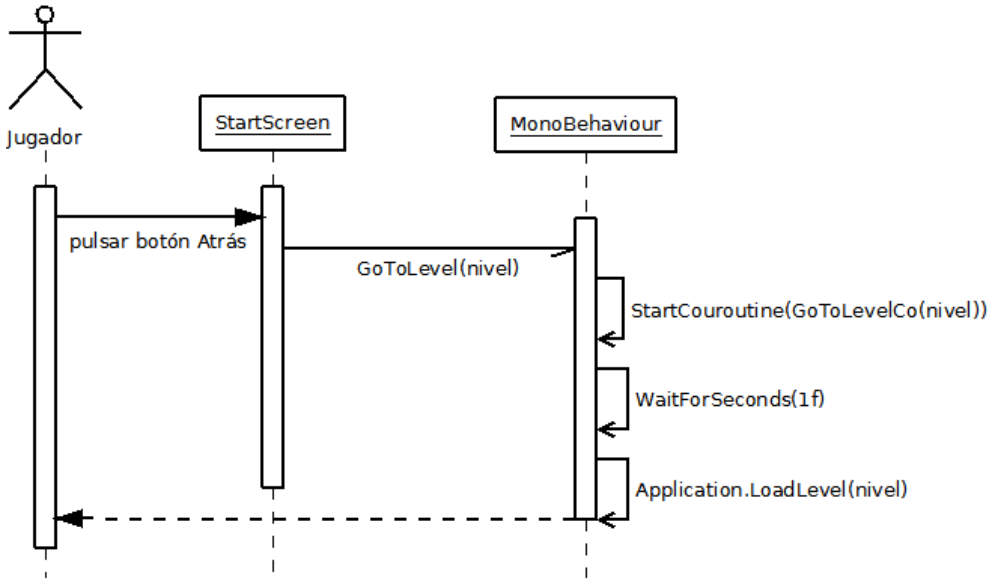



**Diagrama 6. Diagrama de secuencia para cargar un nivel del juego**

# Salir

Este diagrama muestra la secuencia para salir del juego.

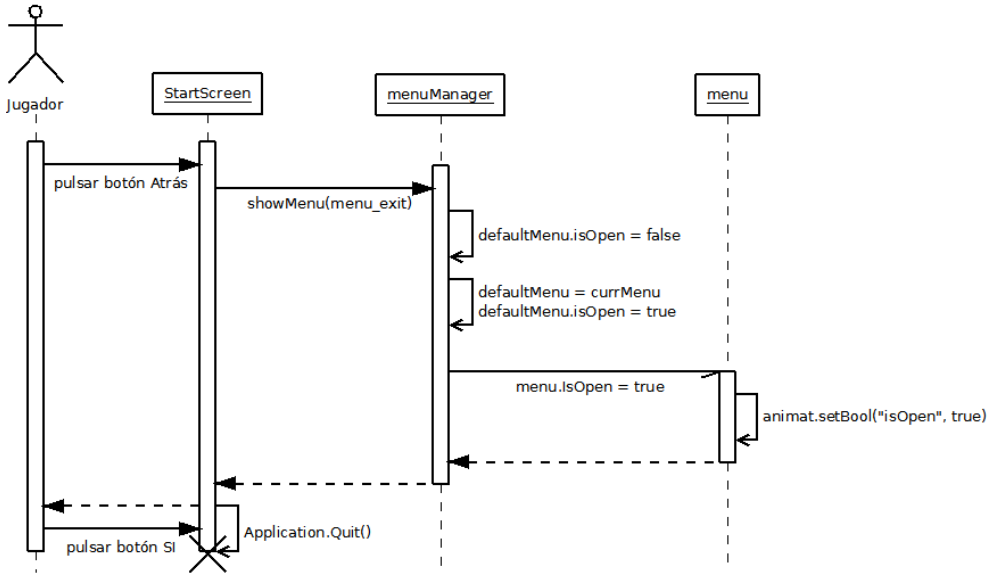



**Diagrama 7. Diagrama de secuencia para salir del juego**

# Mover personaje

Este diagrama muestra la secuencia para mover al personaje del juego.

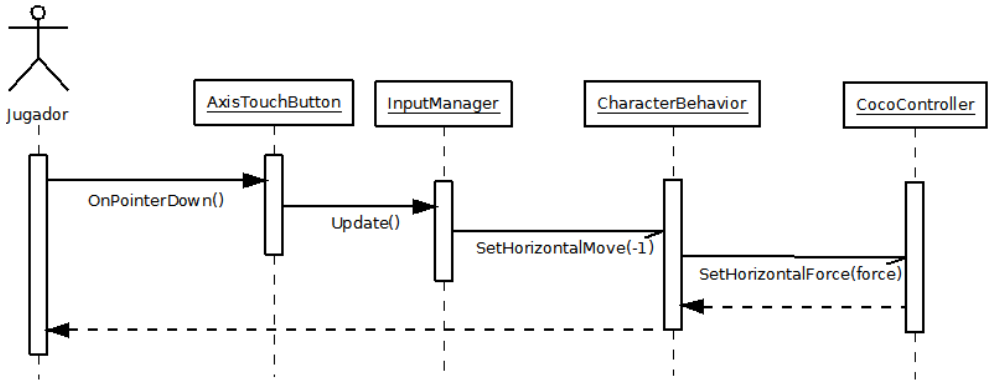



**Diagrama 8. Diagrama de secuencia para mover el personaje a la izquierda**

## Saltar

Este diagrama muestra la secuencia para hacer saltar al personaje del juego.

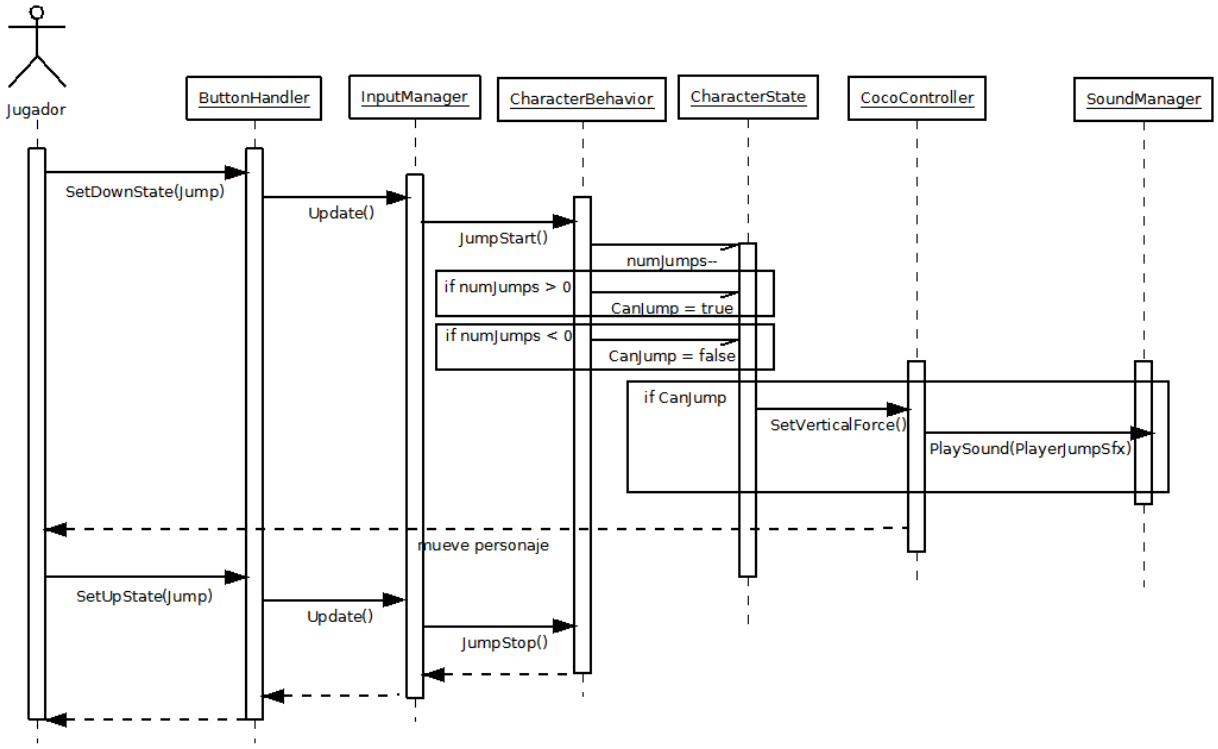



**Diagrama 9. Diagrama de secuencia para hacer saltar al personaje**

## Pausar juego

Este diagrama muestra la secuencia para pausar el juego.

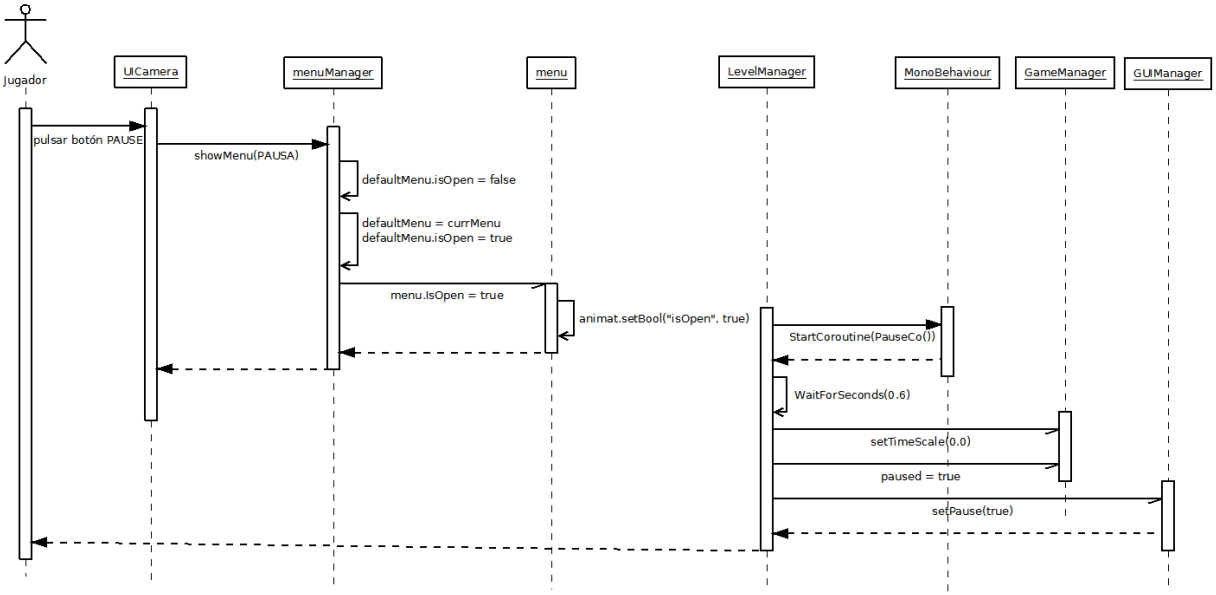



**Diagrama 10. Diagrama de secuencia para pausar el juego**

# Recoger mango

Este diagrama muestra la secuencia para recoger un mango en el juego.

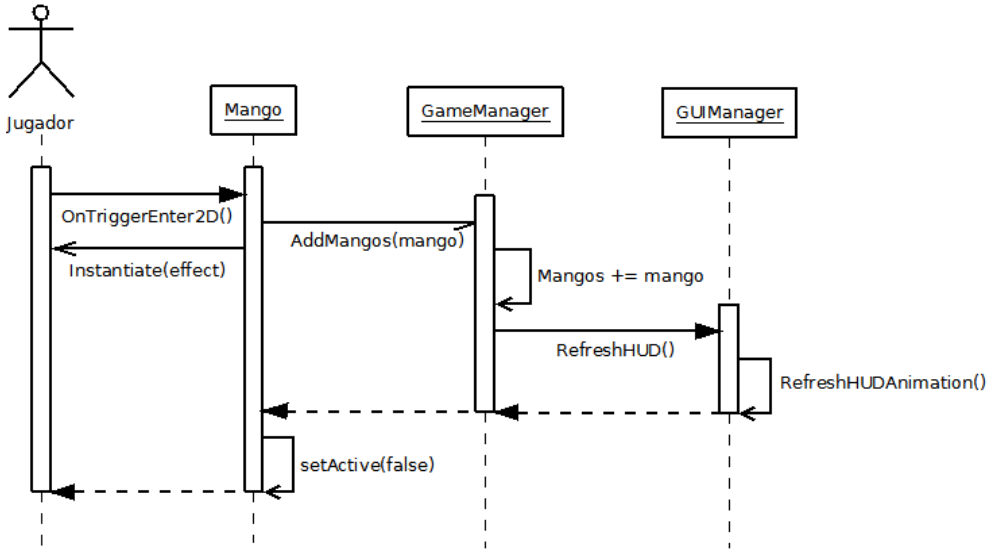



**Diagrama 11. Diagrama de secuencia para recoger un mango del juego**

# Capítulo 6
# Diseño

En este capítulo se va a exponer la arquitectura lógica y física del videojuego y la fase de diseño, compuesta por la historia del juego y la estética, donde se explica cómo se ha diseñado cada componente del videojuego, y para qué. También se expone la Interfaz de Usuario (IU).

## 6.1. Arquitectura lógica

La arquitectura lógica de la aplicación se divide en diferentes capas:

1. **Capa de presentación**: presenta el sistema al usuario, mostrando los componentes de IU (pantallas, menús, etc..) y los componentes de procesos de IU que controlan el flujo de operaciones con el usuario. Se comunica con la *capa de negocio*.

2. **Capa de negocio**: contiene los scripts con las funcionalidades del juego, y las entidades de negocio, que se encargan del intercambio de datos entre capas. Recibe las solicitudes de la *capa de presentación*.

3. **Capa de datos**: en este caso no hay, ya que la aplicación no se comunica en ningún momento con ninguna Base de Datos ni almacenamiento de datos. En un futuro se necesitará para guardar el progreso del jugador, y comunicarse con el GooglePlay para la gestión de los logros del juego.

4. **Capa de componentes comunes**: componentes a los que puede accederse desde cualquier parte de la aplicación, como por ejemplo la configuración del sonido del juego.

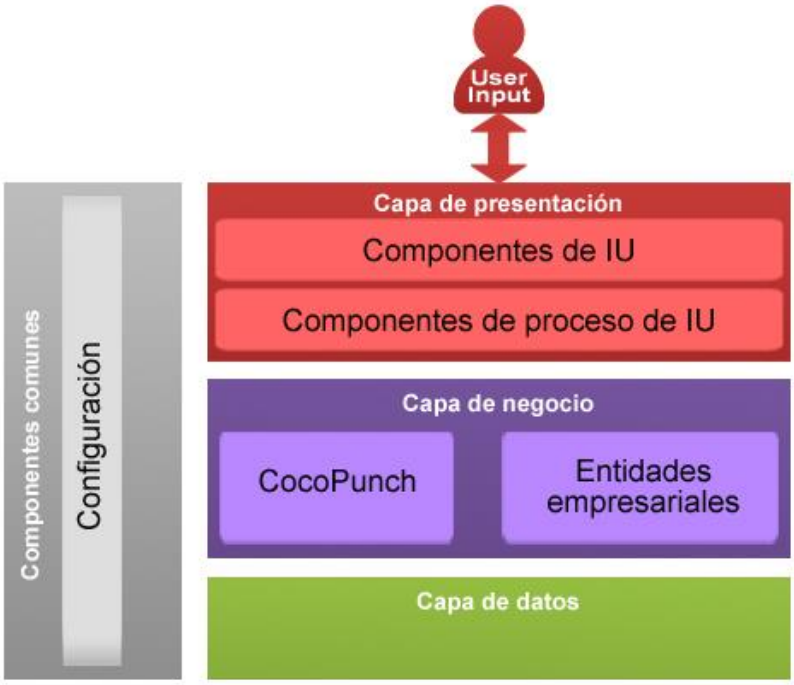



**Figura 9. Arquitectura lógica**

## 6.2. Arquitectura física

El dispositivo móvil es el único elemento físico que interviene, ya que no se requiere de Internet para su funcionamiento, ni acceso a ninguna Base de Datos externa. Es un futuro cuando se implemente el sistema de Logros a través del Google Play, sí será necesario acceder a ese servicio.

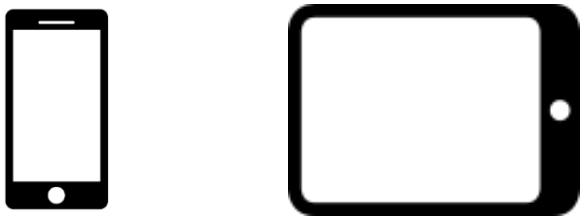


**Figura 10. Arquitectura física**

# 6.3. Diagramas de clases de diseño

En esta sección se van a describir las clases que conforman el sistema, y cómo se relacionan entre sí. De esta forma se detalla la estructura del sistema para tener un conocimiento sobre la implementación de este.

Lo primero es conocer la estructura de Unity3D y sus clases, sabiendo lo explicado en la **Introducción a Unity3D**, sabemos que un proyecto en Unity se basa en *Objetos*, que tienen distintos *Componentes*, y para ser ejecutado en el flujo de ejecución, debe heredar de la clase *MonoBehaviour*. El **Diagrama12** muestra el diagrama de clases de la estructura básica de Unity3D:

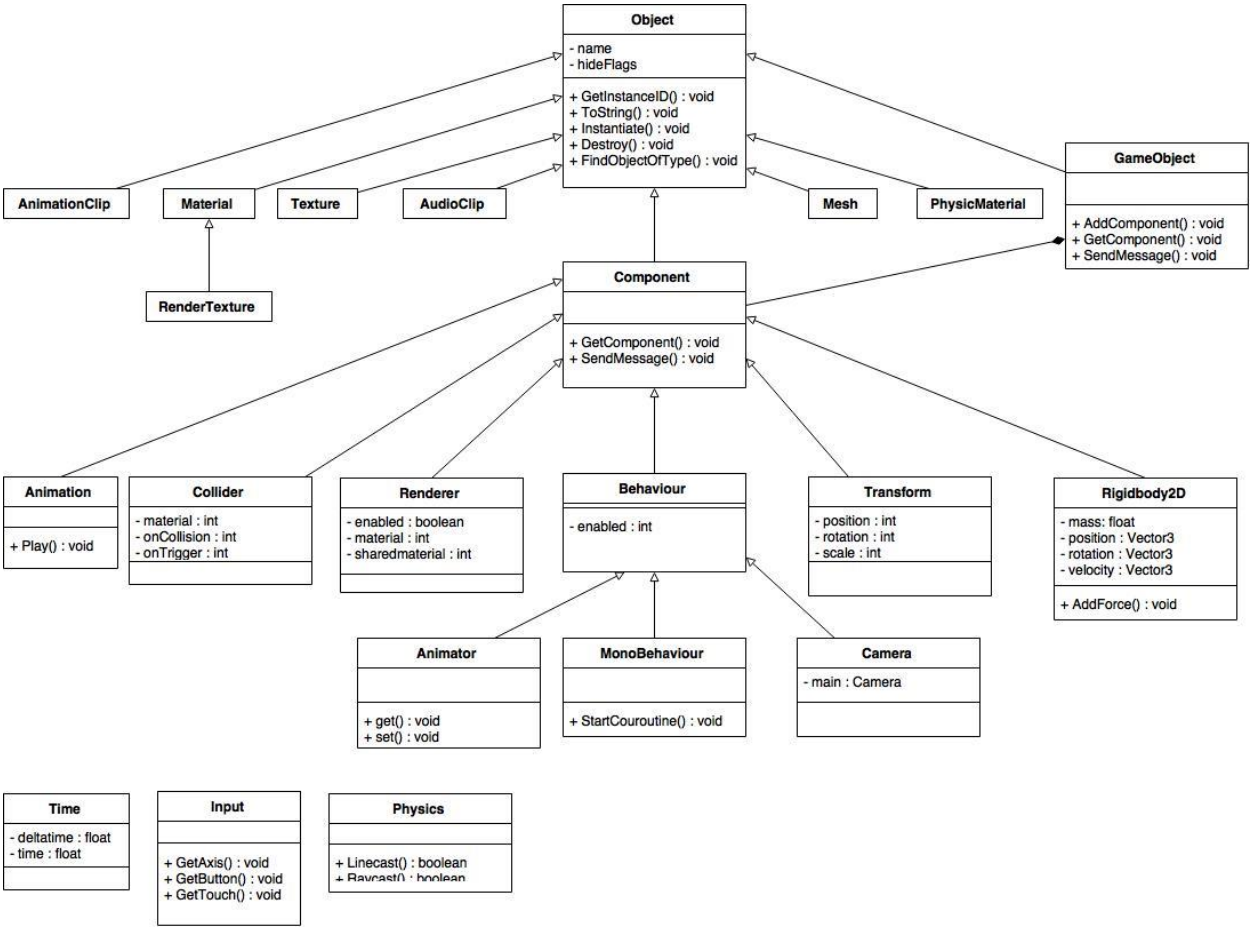



**Diagrama 12. Diagrama de clases de la estructura de Unity3D**

Ahora se pasa a explicar las clases del proyecto. Debido a la magnitud del proyecto, se divide en varios diagramas de clases y se añaden sólo los atributos y funciones principales:

- **Controladores**: diagrama que contiene los controladores del juego, de la cámara y de los objetos que necesiten implementar la clase CocoController.
- **Controladores del juego**: contiene las clases de los controladores del juego, niveles, controles del dispositivo móvil y sonido.
- **Personaje del juego**: clases relacionadas con el personaje del juego.
- **IA**: clases relacionadas con la inteligencia artificial del juego.
- **Objetos del juego**: clases de los objetos que se pueden encontrar en el juego, mangos, vidas, agua, checkpoints, etc..
- **Otros**: clases sobre la Interfaz de Usuario, menús, y otros objetos del juego.
- **Utilidades**: clases que no heredan de ninguna clase, que son de alguna utilidad, y las relacionadas con el editor de Unity3D.

Para ayudar a comprender los atributos, métodos y relaciones entre las clases, se proporciona una documentación detallada en la carpeta de *Documentación*.

# Controladores

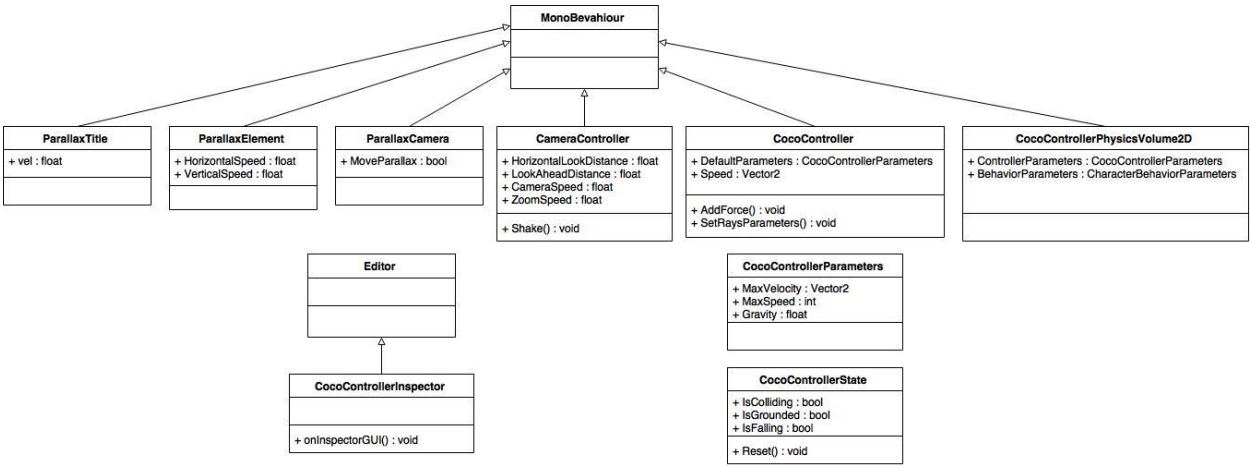



**Diagrama 13. Diagrama de clases de los controladores de la cámara y de CocoController**

# Controladores del juego

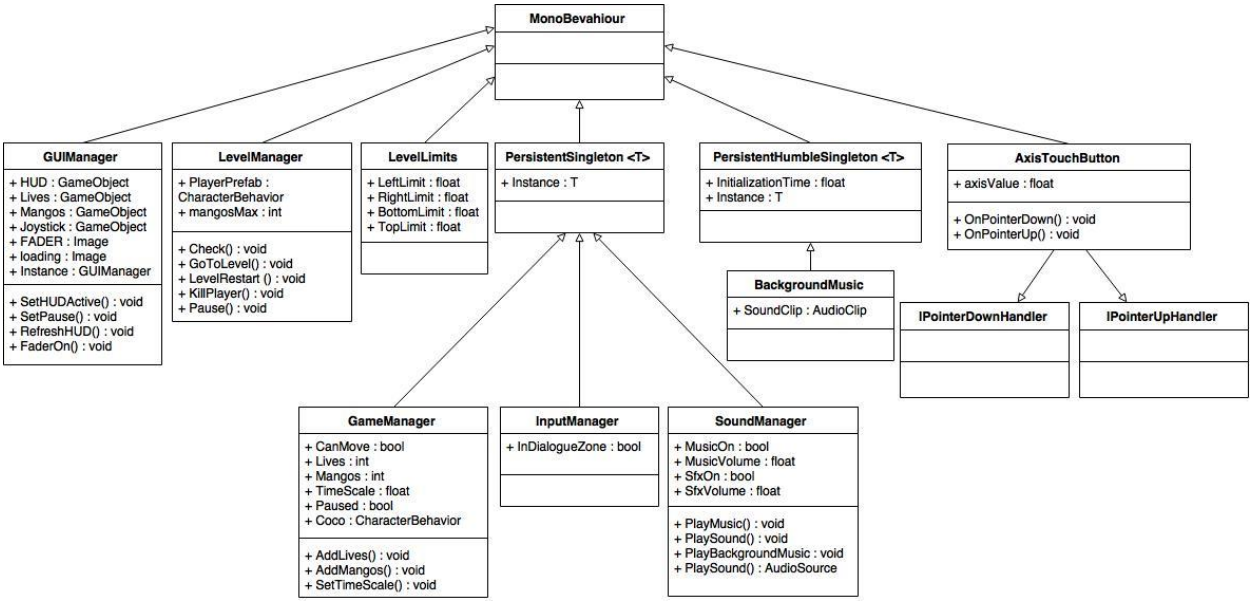



**Diagrama 14. Diagrama de clases de los controladores del juego**

# Personaje del juego

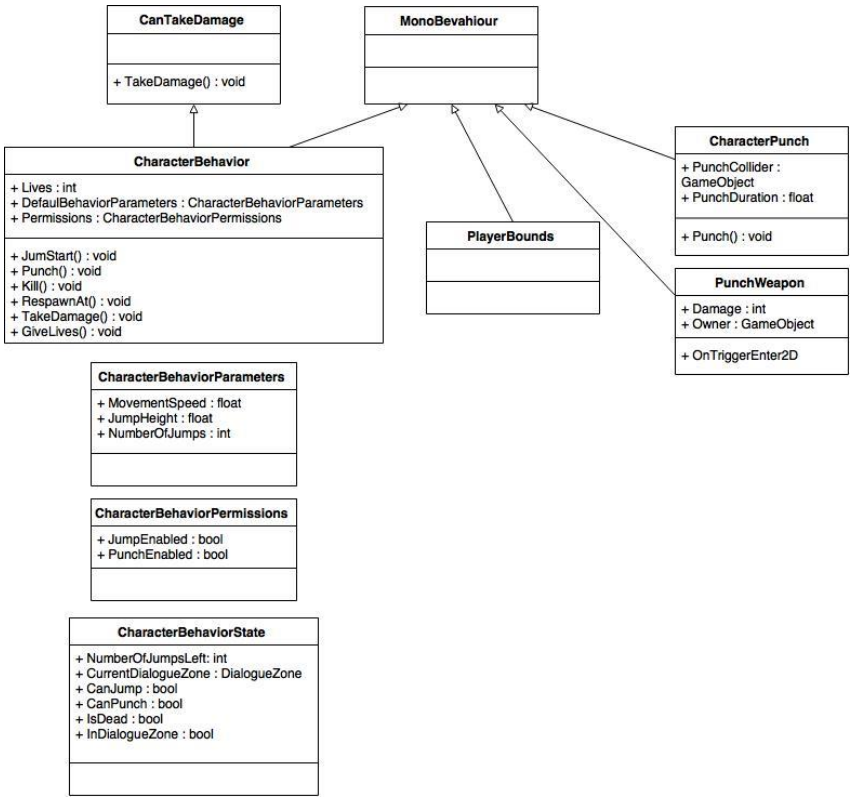



**Diagrama 15. Diagrama de clases del personaje del juego**

# IA

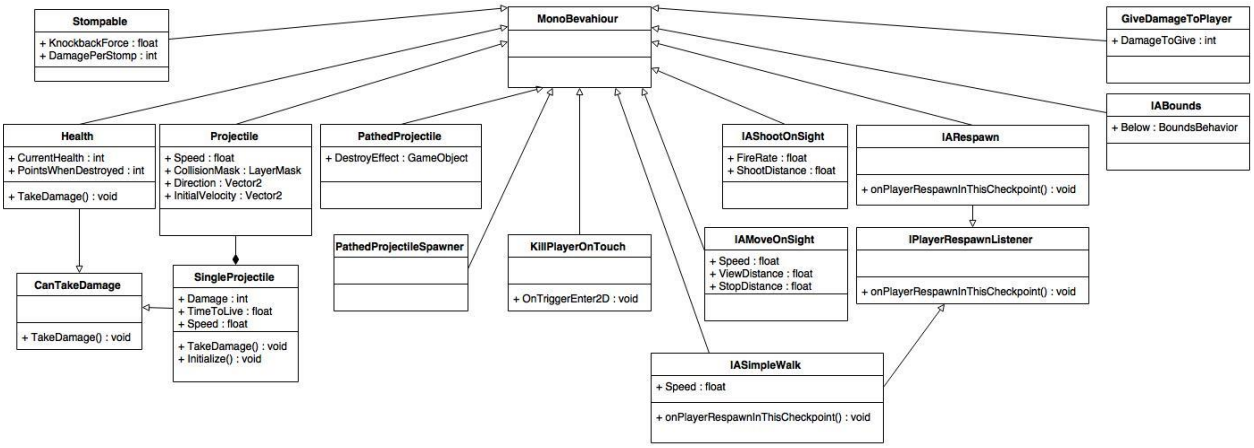



**Diagrama 16. Diagrama de clases de la IA del juego**

# Objetos del juego

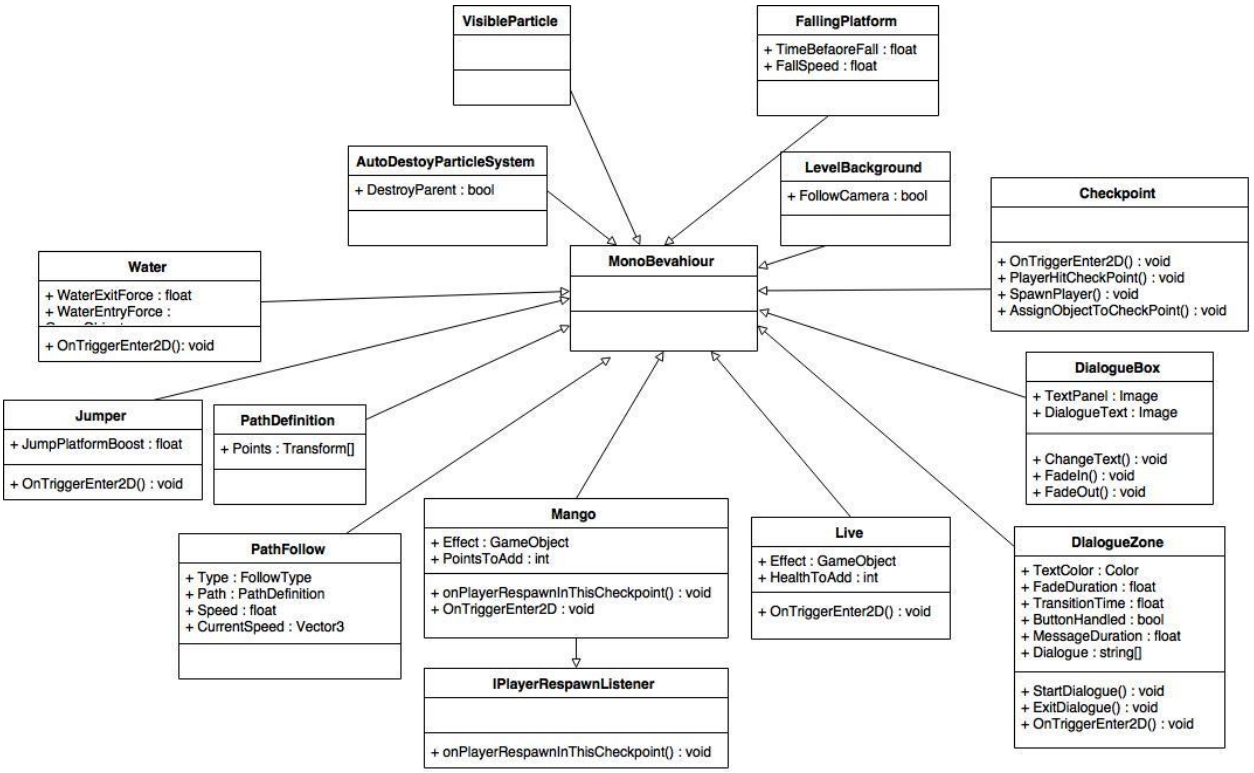



**Diagrama 17. Diagrama de clases de los objetos del juego**

### Otros

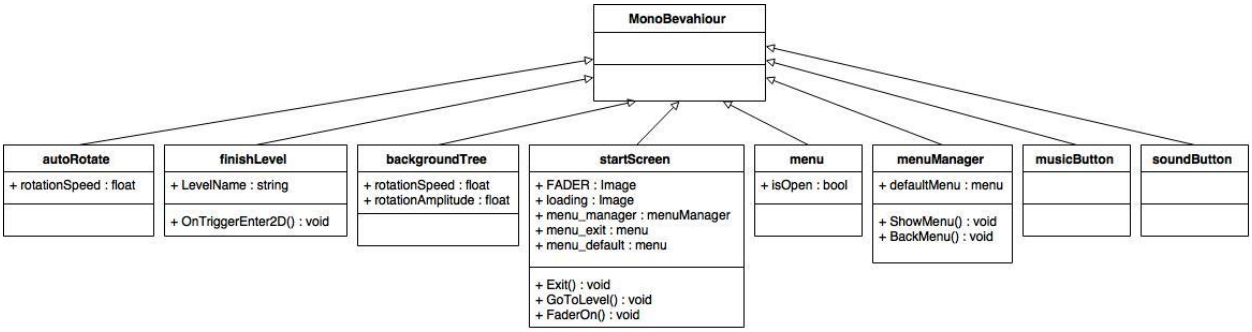



**Diagrama 18. Diagrama de clases de otros objetos del juego y los menús**

### Utilidades

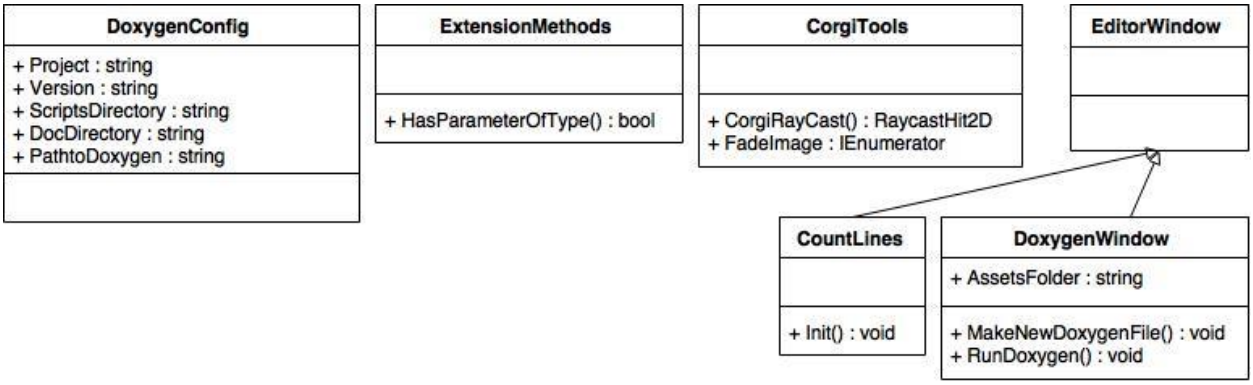



**Diagrama 19. Diagrama de clases de las utilidades de ayuda**

## 6.4. Diagramas de estados

En esta sección se van a describir los distintos estados en el sistema:

- **Personaje**: el personaje del juego tendrá distintos estados dependiendo de lo que el jugador haga.
- **Crunch enemigo**: los enemigos de tipo Crunch, tienen 2 estados, uno de reposo y otro de ataque.
- **Checkpoint**: los checkpoints tienen 2 estados también, uno en reposo y otro al ser sobre-pasado.

- **Música**: la música del juego tiene el estado de activada y desactivada.
- **Sonido**: los efectos de sonido del juego tienen el estado de activados y desactivados.

## Personaje

- **Idle**: el personaje comenzará siempre en este estado por defecto, que realiza una animación de estar parado, esperando a que el jugador realice algún movimiento.
- **Walk**: si el jugador presiona las flechas de movimiento del personaje estando sobre una plataforma, se actualizará su estado, realizando la animación de andar.
- **JumpAndFall**: si el personaje no se encuentra sobre una plataforma, significa que está en el aire, donde puede estar o bajando por una pared, cayendo, o golpeando. Si no se está en ninguno de estos casos, significa que se está saltando, o acabando de saltar, actualizando el estado del personaje. Para controlar este estado, se crea un árbol con 20 estados dentro de este, indicando la animación a realizar en cada estado del salto, o de la caída del salto.
- **WallClinging**: si el personaje está colisionando con una pared en el aire, y el jugador presiona la flecha de movimiento en su dirección, pasará a la animación de bajando por la pared.
- **Punch**: si el jugador presiona el botón de golpear, pasará a este estado, realizando la animación de golpear.
- **Diving**: si el personaje está cayendo (como al morir), se realiza la animación de caer.

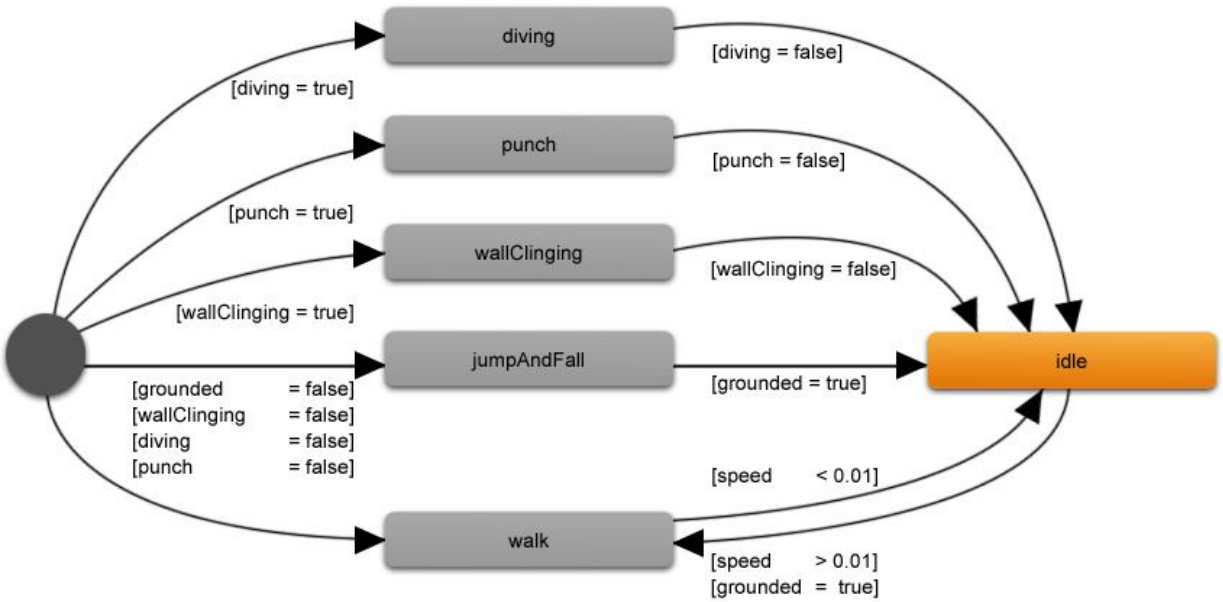



**Diagrama 20. Diagrama de estados del personaje**

## Crunch Enemigo

- **Idle**: el enemigo comenzará siempre en este estado por defecto, que realiza una animación de estar parado.
- **Attack**: si el enemigo se encuentra cerca del jugador, se actualizará a este estado, haciendo una animación de atacar.

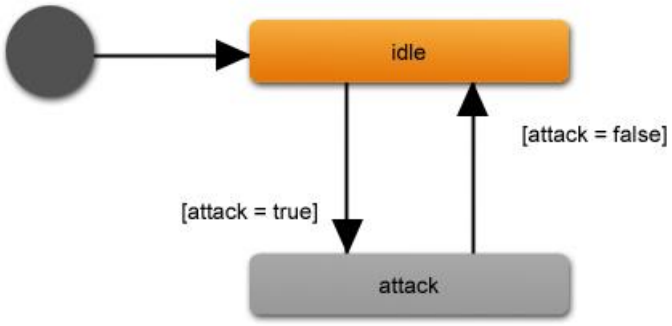



**Diagrama 21. Diagrama de estados del enemigo Crunch**

## Checkpoint

- **Disable**: el checkpoint no ha sido sobrepasado.
- **Normal**: si el jugador sobrepasa el checkpoint, se realiza una animación para indicarlo.

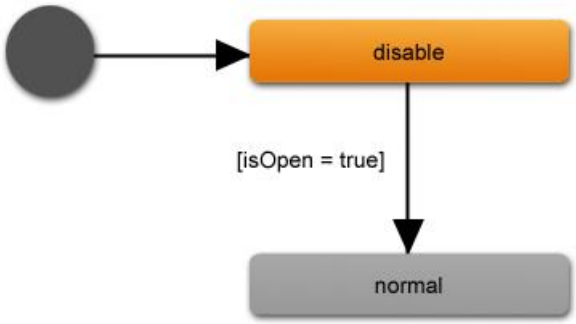



**Diagrama 22. Diagrama de estados del checkpoint**

## Música

- **MusicOn**: la música está activada.
- **MusicOff**: la música está desactivada.

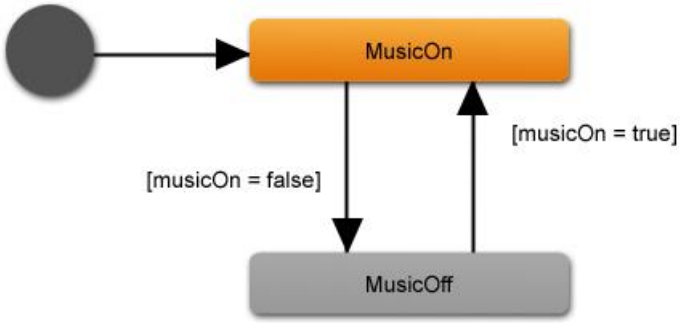



**Diagrama 23. Diagrama de estados de la música del juego**

## Sonido

- **SoundOn**: los efectos de sonido están activados.
- **SoundOff**: los efectos de sonido están desactivados.

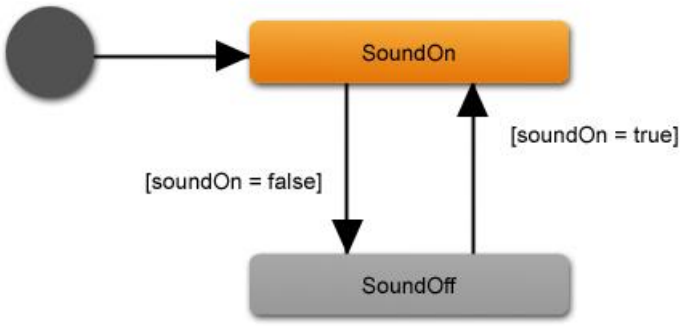



**Diagrama 24. Diagrama de estados de los efectos de sonido del juego**

## 6.5. Historia

"*Un día Coco se levanta sobresaltado observando cómo sus mangos han desaparecido, y en su lugar hay un rastro verde y viscoso que delata a los ladrones.*
*Decide ir en busca de ellos y recuperar lo que es suyo, con sus puños y su coraje, atravesará el lado más oscuro de la isla para expulsar a los invasores*".

## 6.6. Estética del juego

El apartado gráfico se basa en un estilo sencillo pero llamativo, con colores rojo-blanco-amarillo que representan a Coco.

Los niveles se ambientan en la isla de Madagascar, por lo que los árboles serán característicos de esa isla.

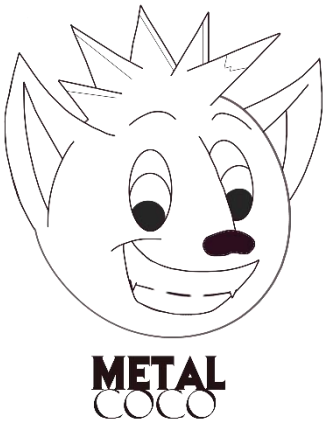
Los personajes, tanto enemigos como amigos son extraterrestres, por lo que su aspecto será verde.

### Personajes

Personaje principal de “CocoPunch” como el nombre indica; nuestro lémur, armado con sus dos puños, tendrá que derrotar a los enemigos que encuentre para poder recuperar sus mangos, y el preciado mango dorado que le ha sido robado.

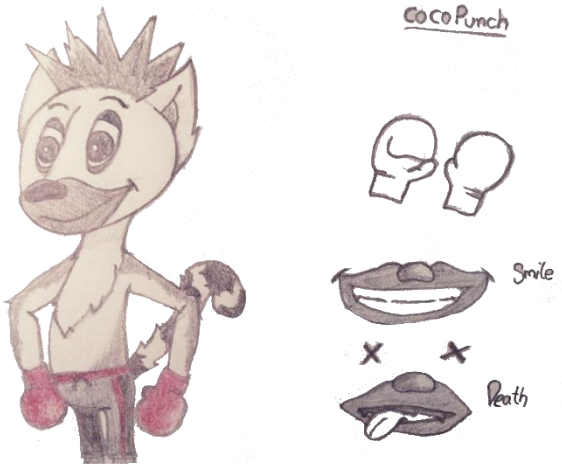
La primera idea fue llamar al juego MetalCoco, y el diseño del lémur era bastante distinto al final de CocoPunch:


Boceto MetalCoco




Diseño MetalCoco

El segundo diseño se acerca más a la idea de CocoPunch:



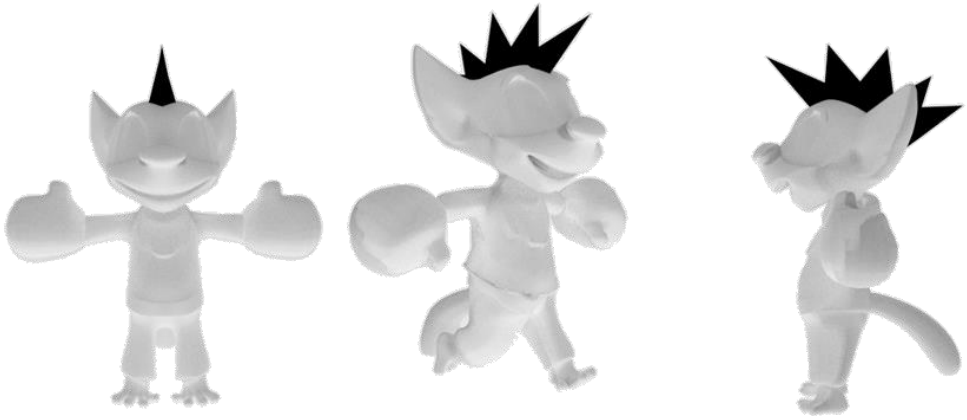

Boceto CocoPunch v1


Diseño CocoPunch v1


Diseño CocoPunch v1 con color

Intenté diseñar el personaje en 3D con el software Blender, pero no conseguí el resultado esperado, y no podía perder demasiado tiempo en un proceso tan extenso, aparte de que al tratarse de un videojuego en 2D, no es necesario el modelado en 3D.

Boceto CocoPunch 3D



Diseño CocoPunch 3D

Finalmente diseñé a Coco con apariencia pequeña y graciosa, de forma que fuese fácil de animar:

Diseño CocoPunch v2.1

Diseño CocoPunch v2.2

Diseño CocoPunch v3

Las animaciones se realizan con el software Spriter, que, a partir de las partes del personaje, permite animar los distintos frames que componen el movimiento.

Animación parado

Animación golpear

## Objetos

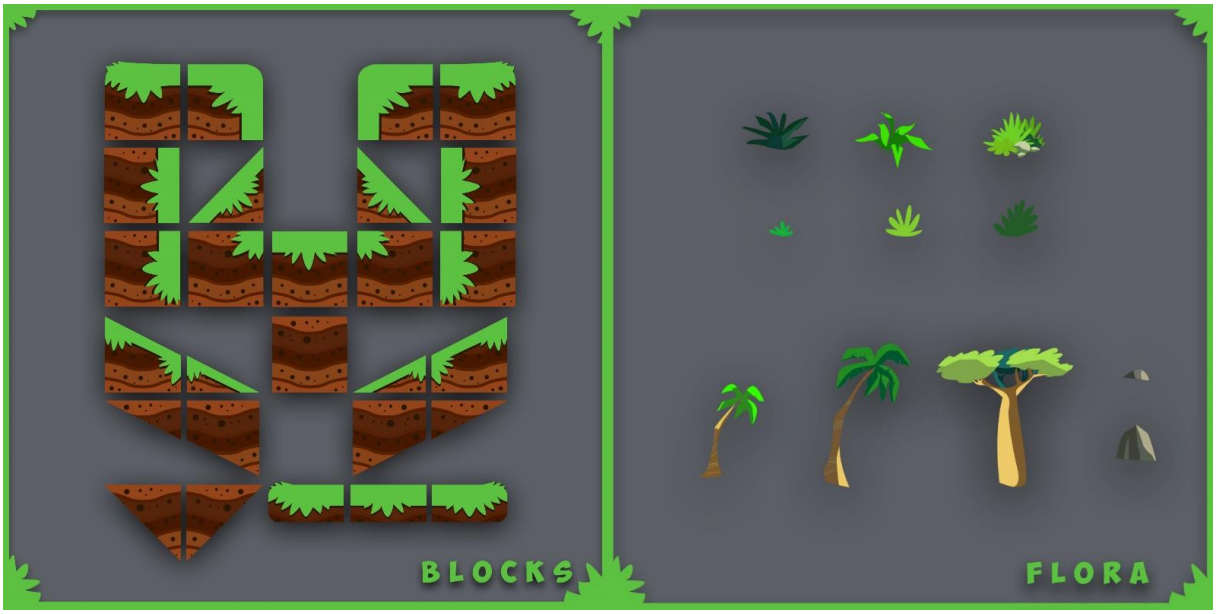

Los mangos son los objetos que deberá ir recogiendo nuestro personaje, cada nivel dispondrá de un número de mangos, que podrán conseguirse tanto a lo largo del nivel, como derrotando a enemigos.

El Mango de Oro

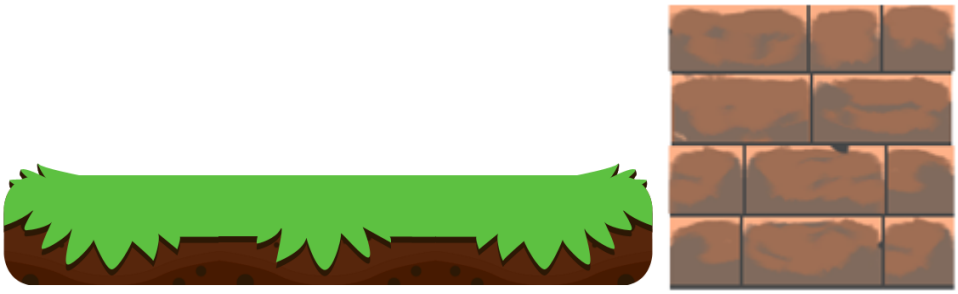
El objetivo final del juego es derrotar al rey invasor y recuperar El Mango de Oro.


Las vidas son objetos que encontraremos raramente en los niveles; al ser recogidos aumentarán en 1 las vidas disponibles de nuestro personaje.

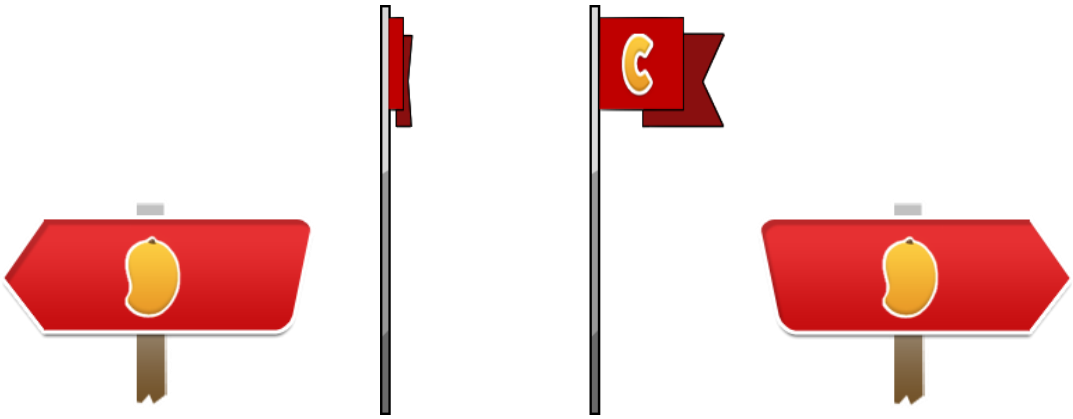
## Entorno



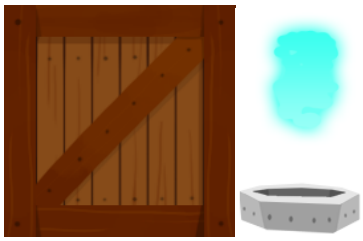

Los niveles se construirán con bloques, por los que el jugador podrá moverse. El entorno del juego dispondrá de distintos elementos, entre ellos la flora (piedras, árboles, arbustos).



A parte de bloques, también hay plataformas en movimiento, y plataformas que se caen al situarse sobre ellas.



A lo largo de cada nivel encontraremos flechas de dirección que indican hacia dónde es el final del nivel. También podremos encontrar con Checkpoints, que, al pasar por ellos, servirán para volver a ese punto si se muere, en vez de volver al inicio del nivel.



También hay otros objetos como Cajas destructibles que se destruyen siendo golpeadas o Saltadores que nos hacen volar por los aires.

## Enemigos

En la aventura por recuperar los mangos robados, Coco se encontrará con varios tipos de enemigos, a los que deberá derrotar para proseguir su camino. Cada tipo de enemigo tiene una característica propia, tanto por tamaño, tipo de ataque, y vida.

Enemigo1 Enemigo2

**Enemigo1**: patrulla un camino. Si toca al jugador lo mata. Para destruirlo, saltar encima o golpearlo.

**Enemigo2**: patrulla un camino. Si toca al jugador lo mata. Para destruirlo, saltar encima o golpearlo. Si está a una distancia cercana al jugador, atacará haciéndose más grande.

Cañón Cuchillas

**Cañón**: dispara proyectiles en una dirección fija cada X segundos. Los proyectiles pueden ser destruidos golpeándose, pero no saltando sobre ellos. Los cañones no pueden destruirse.

**Cuchillas**: giran continuamente. Pueden moverse de arriba abajo. Si tocan al jugador lo matan. No puede destruirse.

**Rey enemigo**: rey de los ladrones que han robado los mangos a Coco, es el jefe final del juego. Dispone de distintas fases, a cada cual más difícil. Se le puede golpear, y saltar dependiendo de la fase. Suelta *El Mango de Oro*.

## Amigos

No todo lo que encontremos serán enemigos, entre los ladrones hay algunos que no vieron bien la acción de robar los Mangos, por lo que nos ayudarán en nuestro camino. Serán reconocidos por una exclamación amarilla encima, que significa que se puede dialogar con ellos, en busca de pistas o consejos.

# Interfaz de Usuario

Las Interfaces de Usuario van a especificarse mediante este tipo de tabla:

| **ID y Nombre:** | **IU-XY** |
|---|---|
| **Descripción:** | *Breve descripción.* |
| **Disparador:** | *Acción/es que activan la Interfaz.* |
| **Boceto:** | *Boceto que representa el prototipo de la Interfaz.* |
| **Eventos:** | *Eventos que puede generar.* |
| **Otra información:** | *Otra información de interés.* |

**Tabla 99. Ejemplo tabla IU**

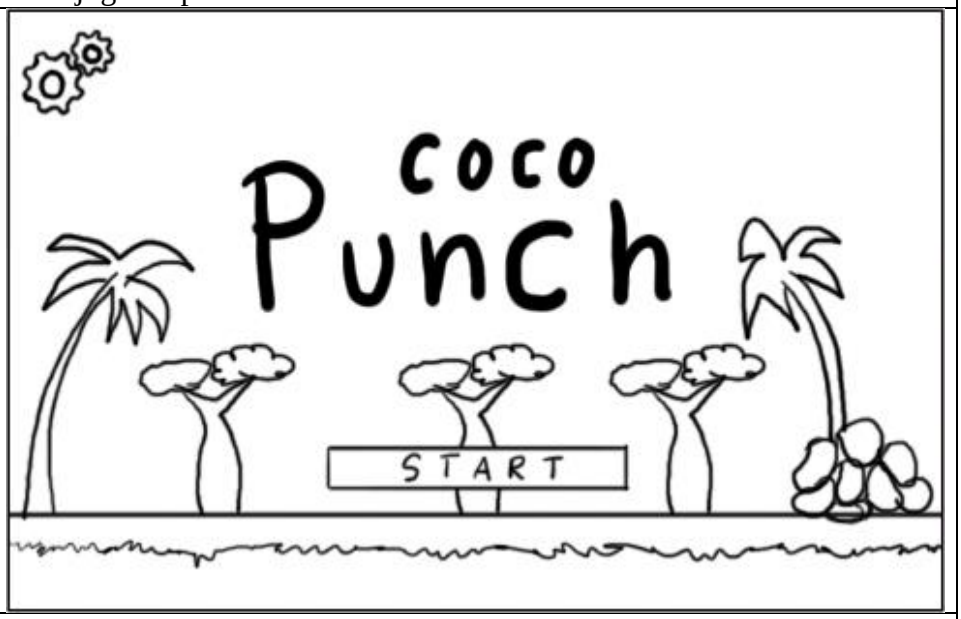
| **ID y Nombre:** | **IU-01 Pantalla de inicio** |
|---|---|
| **Descripción:** | Pantalla inicial del juego que muestra el título del videojuego. Permite gestionar los ajustes del juego, salir de la aplicación o ir al menú de selección de niveles. |
| **Disparador:** | 1. El jugador ejecuta el juego en su dispositivo móvil.<br>2. El jugador pulsa el botón de HOME en el menú de PAUSA.<br>3. El jugador pulsa el botón de SALIR en el menú de GAMEOVER. |
| **Boceto:** |  |
| **Eventos:** | menuManager.ShowMenu(menu_settings)<br>menuManager.ShowMenu(menu_levels)<br>menuManager.ShowMenu(menu_exit) |
| **Otra información:** | |

**Tabla 100. IU-01 Pantalla de inicio**

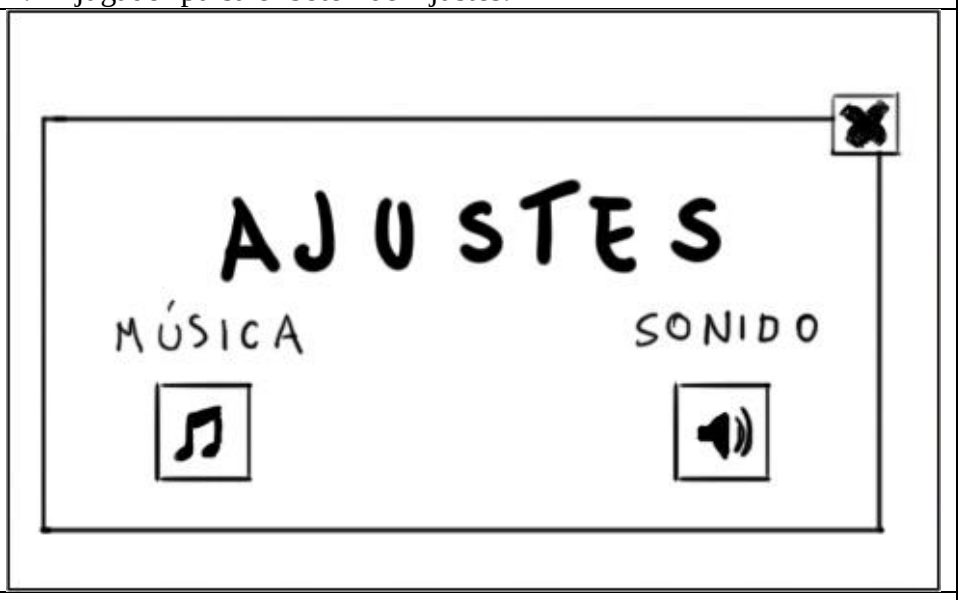
| ID y Nombre: | IU-02 Ajustes del juego |
|---|---|
| Descripción: | Menú de ajustes del juego, permite gestionar la música de fondo y los efectos de sonido. |
| Disparador: | 1. El jugador pulsa el botón de Ajustes. |
| Boceto: |  |
| Eventos: | soundManager.PlayMusic()<br>soundManager.PlaySound()<br>menuManager.BackMenu() |
| Otra información: | |

**Tabla 101. IU-02 Ajustes del juego**

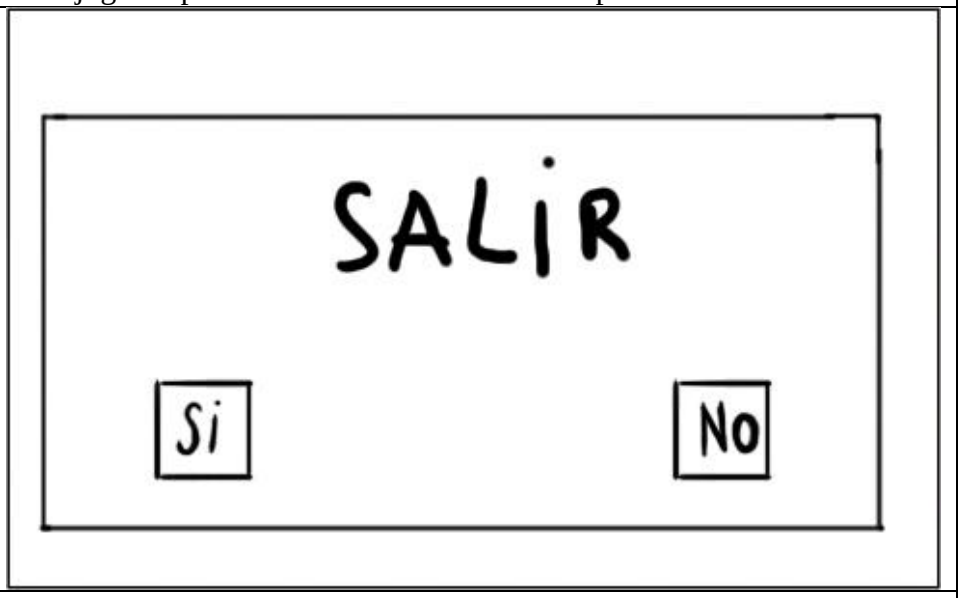
| ID y Nombre: | IU-03 Salir del juego |
|---|---|
| Descripción: | Menú de salir del juego, permite cerrar la aplicación. |
| Disparador: | 1. El jugador pulsa el botón de Atrás de su dispositivo móvil. |
| Boceto: |  |
| Eventos: | StartScreen.Exit()<br>menuManager.BackMenu() |
| Otra información: | |

**Tabla 102. IU-03 Salir del juego**

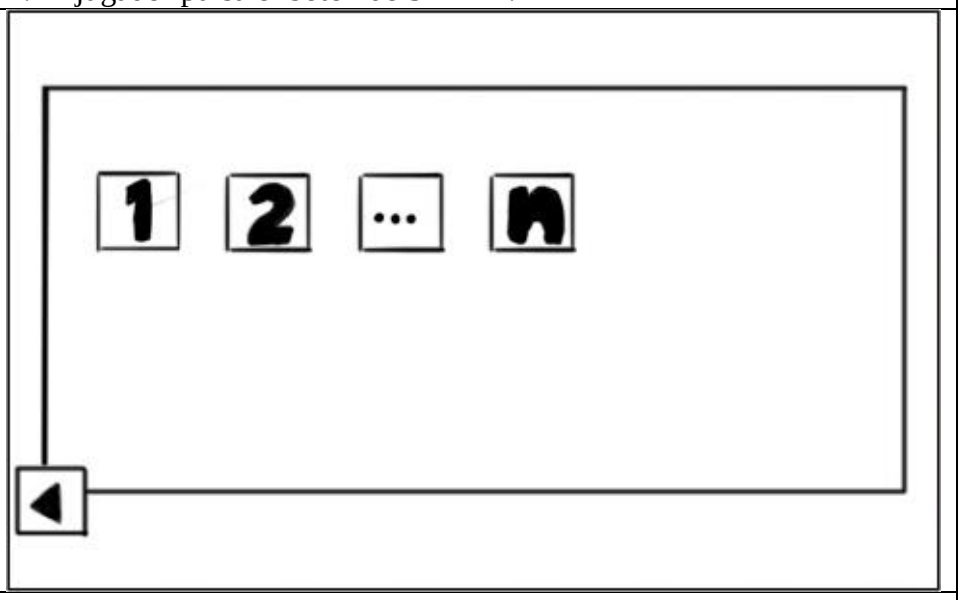
| ID y Nombre: | IU-04 Seleccionar nivel |
|---|---|
| Descripción: | Menú de seleccionar niveles del juego, permite cargar el nivel seleccionado. |
| Disparador: | 1. El jugador pulsa el botón de START. |
| Boceto: |  |
| Eventos: | StartScreen.GoToLevel()<br>menuManager.BackMenu() |
| Otra información: | |

**Tabla 103. IU-04 Seleccionar nivel**

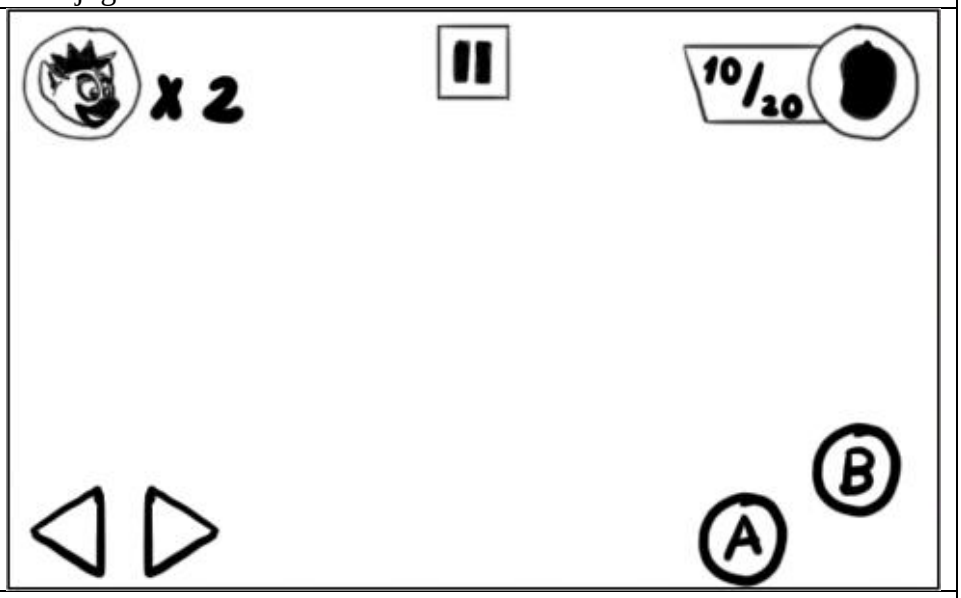
| ID y Nombre: | IU-05 HUD del juego |
|---|---|
| Descripción: | Menú de HUD, que contiene la información del jugador en los niveles, esto son las vidas disponibles, los mangos recogidos en el nivel, el número de mangos máximo del nivel, y un botón de pausa. |
| Disparador: | 1. El jugador selecciona un nivel. |
| Boceto: |  |
| Eventos: | menuManager.ShowMenu(PAUSE)<br>AxisTouchButton.CrossPlatformInputManager.GetAxis("Horizontal")<br>ButtonHandler.SetDownState(Jump)<br>ButtonHandler.SetDownState(Punch) |
| Otra información: | |

**Tabla 104. IU-05 HUD del juego**

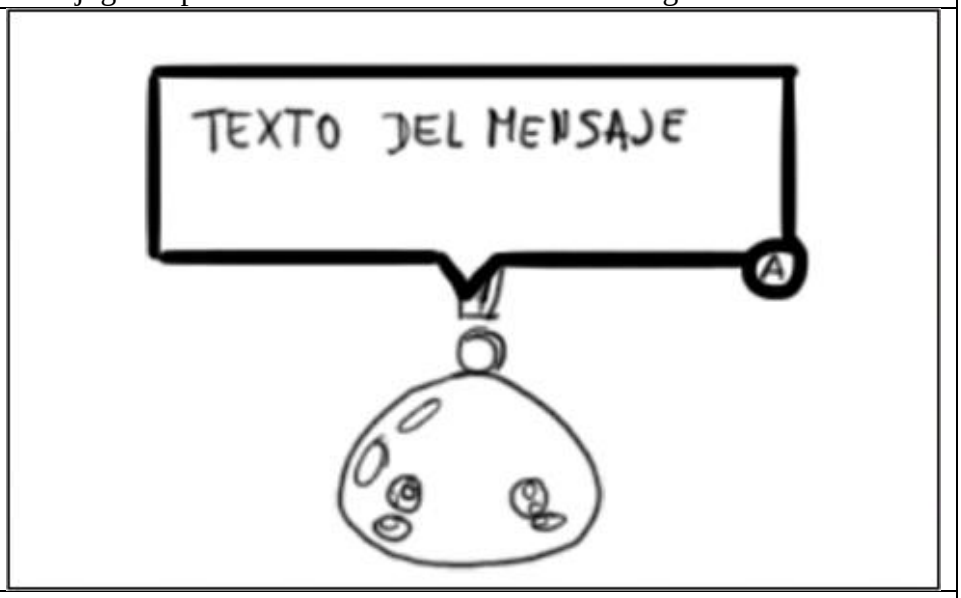
| | |
|---|---|
| **ID y Nombre:** | **IU-06 Diálogo** |
| **Descripción:** | Diálogos de texto que ayudan al jugador en los niveles. |
| **Disparador:** | 1. El jugador pulsa el botón A en una zona de diálogo. |
| **Boceto:** |  |
| **Eventos:** | BehaviorState.CurrentDialogueZone.StartDialogue()<br>BehaviorState.CurrentDialogueZone.ExitDialogue() |
| **Otra información:** | |

**Tabla 105. IU-06 Diálogo**

| | |
|---|---|
| **ID y Nombre:** | **IU-07 Pausar el juego** |
| **Descripción:** | Menú de PAUSA, contiene un botón para volver a la pantalla de inicio, otro para activar/desactivar el sonido del juego, otro para reiniciar el nivel y otro para continuar el juego cerrando el menú. |
| **Disparador:** | 1. El jugador pulsa el botón A en una zona de diálogo. |
| **Boceto:** |  |
| **Eventos:** | LevelManager.GoToLevel(StartScreen)<br>LevelManager.Pause()<br>SoundManager.PlaySound()<br>LevelManager.LevelRestart()<br>menuManager.BackMenu() |
| **Otra información:** | |

**Tabla 106. IU-07 Pausar el juego**

| | |
|---|---|
| **ID y Nombre:** | **IU-08 GameOver** |
| **Descripción:** | Menú de GAMEOVER, contiene un botón para continuar jugando reiniciado el progreso del nivel, y otro para volver a la pantalla inicial del juego. |
| **Disparador:** | 1. El jugador muere y no dispone de más vidas. |
| **Boceto:** |  |
| **Eventos:** | LevelManager.LevelRestart()<br>LevelManager.GoToLevel(StartScreen) |
| **Otra información:** | |

**Tabla 107. IU-08 GameOver**

# Capítulo 7
# Implementación

En este capítulo se explica cómo se ha desarrollado e implementado la estructura y el diseño especificados. Dirigido a quien quiera entender las funcionalidades desarrolladas internamente, a un nivel más técnico.

## 7.1. Organización

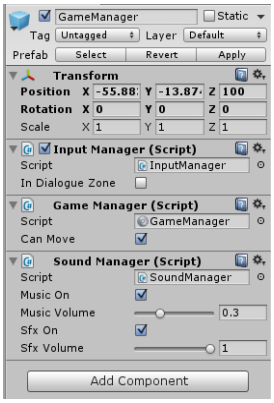
El proyecto se organiza en distintas carpetas:

**CocoPunch**: proyecto Unity3D.

- **Assets**: paquetes de recursos.
  - **CocoPunch**: asset que contiene los recursos del juego.
    - animations: contiene las animaciones del juego.
      - coco: animaciones de Coco.
      - environment: animaciones del entorno (checkpoint, saltador).
      - gui: animaciones de la Interfaz de Usuario.
        - exit: menú de Salir.
        - gameover: menú de Game Over.
        - hud: menú de HUD (Heads-Up-Display).
        - pausa: menú de Pausa.
        - select: menú de selección de Nueva/Cargar partida.
        - settings: menú de Ajustes.
      - IA: animaciones de la Inteligencia Artificial.
      - items: animaciones de los objetos coleccionables del juego.
    - fonts: fuentes usadas en el juego.
    - materials: materiales de las partículas generadas en el juego.
    - materials-physics: materiales de las físicas de los objetos.
    - resources: prefabs/objetos con unas propiedades establecidas.
      - backgrounds: prefabs del entorno de fondo (cielo, nubes, montañas).
      - coco: prefab de Coco.
      - effects: prefabs de los efectos del juegos (daño, vida, mango, etc..).

        - environment: prefabs del entorno (bloques, plataformas, árboles, flora).
        - gui: prefabs de la Interfaz de Usuario (cajas de diálogo, cámaras, etc..).
        - IA: prefabs de Inteligencia Artificial (enemigos, amigos).
        - items: prefabs de los objetos del juego (vida, mango).
        - levelmanagers: prefabs de los controladores de los niveles (límites, inicio, checkpoint, etc..).
        - music: prefabs de la música del juego.
        - weapons: prefabs de las armas del juego.
    - scenes: escenas del juego (pantallas).
    - scripts: código del juego en lenguaje C#.
        - camera: scripts de las cámaras.
        - character: scripts de los personajes.
        - editor: scripts del editor de Unity3D.
        - environment: scripts del entorno (movimiento árboles, saltador).
        - gameManagement: scripts del control del juego.
        - gui: scripts de la Interfaz de Usuario (controlador GUI, menús).
        - helpers: scripts de utilidad de assets externos.
        - IA: scripts de la Inteligencia Artificial.
        - items: scripts de los objetos del juego (vida, mango).
        - weapons: scripts de las armas del juego.
    - sounds: sonidos del juego (música y sonido sfx).
    - sprites: imágenes del juego.
        - coco: imágenes de Coco.
        - environment: imágenes del entorno.
        - gui: imágenes de la Interfaz de Usuario.
        - IA: imágenes de la Inteligencia Artificial.
        - icon: imágenes del icono del juego.
        - items: imágenes de los objetos del juego (vida, mango).
- **UnityStandardAssets**: scripts del asset standard de Unity.
    - CrossPlatformInput: scripts para gestionar los controles en multiplataforma.
    - Editor: scripts del editor de Unity3D.
        - CountLines: script que permite contar el total de líneas de código del proyecto.
        - Doxygen: plugin para autogenerar documentación.
    - Effects: scripts de distintos effectos y shaders de utilidad.

# 7.2. Pantallas del juego




Todas las pantallas del juego van a tener que contener el objeto **GameManager**, que controla el juego, este objeto contiene varios scripts que heredan de la clase *PersistentSingleton*, la cual permite acceder a una instancia de una clase desde cualquier sitio. De esta manera se obliga al juego a disponer del objeto GameManager en todos y cada uno de sus niveles:

- **InputManager**: envía notificaciones al jugador (diálogos,…) y controla los movimientos del personaje.
  Comprueba cada frame si el jugador está en una zona de diálogo, en caso afirmativo, controla si se presiona el botón A (saltar) para iniciar el diálogo, o el botón B (golpear) para salir del diálogo:

```
if(CrossPlatformInputManager.GetButtonDown("Jump")){_player.Beha-
viorState.CurrentDialogueZone.StartDialogue();}
if(CrossPlatformInputManager.GetButtonDown("Punch")){_player.Beha-
viorState.CurrentDialogueZone.ExitDialogue();}
```

  Controla el movimiento del jugador al pulsar las flechas de movimiento:

```
_player.SetHorizontalMove(CrossPlatformInputManager.GetAxis("Hori-
zontal"));
_player.SetVerticalMove(CrossPlatformInputManager.GetAxis("Verti-
cal"));
```

  Y al pulsar los botones de salto y golpeo:

```
_player.JumpStart();
_player.GetComponent<CharacterPunch>().Punch();
```

- **GameManager**: controla el progreso del juego (mangos, vidas, pausa,…).
  Mediante las funciones `AddLives()` y `SetLives()` modifica las vidas del juego, actualizando el HUD `GUIManager.Instance.RefreshHUD()`, `AddMangos` y `SetMangos` modifican los mangos; y `SetMangosMax` modifica los mangos máximos de un nivel.
  Controla el tiempo del juego (para la Pausa por ejemplo) mediante la función `SetTimeScale(float newTimeScale)` y `FreezeCharacter()` para inmovilizar al jugador.

- **SoundManager**: controla la música de fondo del juego y los efectos de sonido.
  La función `PlayMusic()` permite activar/desactivar la música del juego y `PlaySound()` los efectos de sonido.
  `PlayBackgroundMusic(AudioSource Music)` reproduce la música en caso de estar activada y `PlaySound(AudioClip Sfx, Vector3 Location)` reproduce el efecto de sonido indicado.

También tendrán que contener el objeto **UICamera**, que contiene la Interfaz de Usuario de los niveles. El objeto tendrá un componente *Camera* de *Proyección Ortográfica* (2D) y el script *GUIManager* que es el encargado de la IU Gráfica del juego (GUI), activando el HUD (`SetHUDActive(bool state)`), actualizando el HUD (`RefreshHUD()`)que recoge el número de vidas, mangos y mangosMax del GameManager, y actualizando el HUD con la función `refreshHUDAnimation(GameObject spriteObj, float col, float row, float colStart, float total, string type, int index)`, que establece la posición de la textura recogiendo el resto de dividir entre 10 para las unidades, y el resultado de restar el total de las unidades entre 10 para las decenas; aprovechando la característica del componente *RawImage* como ya he comentado anteriormente, se modifica el atributo *uvRect* para modificar el número de la textura. También controla las transiciones entre niveles y la pausa del juego. Como he comentado en la Introducción a Unity3D, desde su versión 4.6 dispone de un sistema de GUI, dicho sistema se centra en el objeto *Canvas*, el cual se adecuará a las distintas resoluciones de pantalla de los distintos dispositivos sin necesidad de programar ningún script para ello. Dispone de los componentes:

- **Rect Transform**: ya no tendrá el componente *Transform*, que como hemos dicho tiene todo GameObject en Unity, sino que tiene el componente *Rect Transform*. El objeto canvas no dispone de valores en Rect Transform, ya que su tamaño depende del siguiente componente.

- **Canvas**: especifica si el Canvas va a depender de una cámara (*Camera*), por defecto (*Overlay*) o personalizado (*World Space*). En mi caso elijo Camera, ya que se encuentra dentro del objeto UICamera.

- **Canvas scaler**: permite tener esta gran escalabilidad a la hora de ejecutarse en dispositivos de distintas resoluciones. Para que se escale en cuanto al tamaño de la pantalla del dispositivo, hay que especificar en el atributo *Ui Scale Mode - Scale With Screen Size*, porque si no se escalará dependiendo de la posición de los anchors, y no es lo que se quiere.

- **Graphic Raycaster**: permite reconocer los eventos en la pantalla.

- **Menu Manager**: script que controla los estados de los Menús. Se establece el menú HUD por defecto.

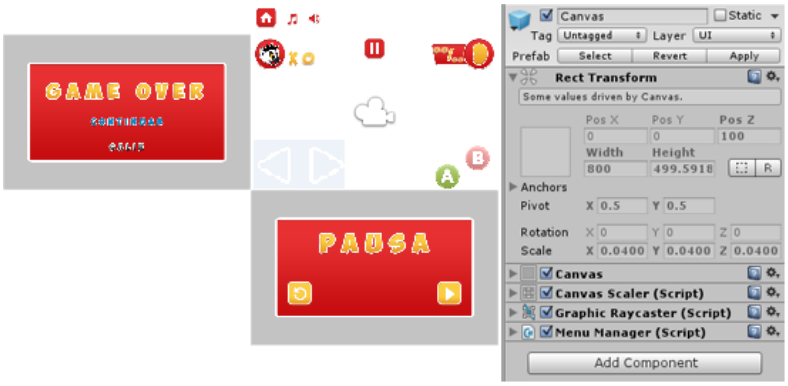
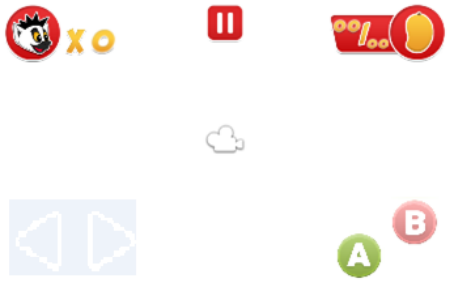



**Figura 11. Canvas**

El objeto **Canvas** contiene a su vez distintos objetos, que son los distintos menús que dispone la IU, todos ellos contendrán los componentes *Menu* (indica que el GameObject es de tipo Menú), *Animator* (animación de activado/desactivado) y *Canvas Group* (grupo de objetos dentro del Canvas):

- **HUD** (*Heads Up Display*): información sobre las vidas del jugador (arriba izquierda) y mangos del nivel (arriba derecha), tanto recogidos como el máximo. También contiene los controles para mover al personaje, las flechas de moverse (abajo izquierda) y los botones de saltar - A y golpear - B (abajo derecha).
    - **HUD**: GameObject que contiene:
        - **Lives**:
            - **hud_lives**: *Canvas renderer* con la *Image* de Coco.
            - **lives**: contador de las vidas del jugador. Para ello, debido a que quería utilizar una font personalizada, se define como *Raw Image*. Esto permite establecer en vez de un sprite, una textura, que mediante el atributo *UV Rect*, va aumentando el número vía script (*GUIManager* función *refreshHUDAnimation*).
        - **Mangos**: igual que Lives pero con el sprite de Mangos. En vez de un Raw Image para las vidas, en este caso se tiene uno para las unidades del número de mangos recogido, otro para las decenas, otro para las unidades del máximo de mangos en el nivel y otro para las decenas del máximo de mangos.
    - **PauseButton**: botón que pausa el juego. Necesita del script *Mobile Control Rig* para poder ser pulsado por un dispositivo móvil.
        - **btn_pause**: botón que, al ser pulsado, llama por medio de la función `OnClickListener()` a las funciones `LevelManager.Pause()` que pausa el tiempo de juego, congelando el juego, y `menuManager.ShowMenu(PAUSE)` que muestra el menú de Pausa.
    - **Buttons**: botones de acción de Saltar y Golpear. También necesita del script Mobile Control Rig.
        - **JumpButton**: botón de salto. Necesita el script *Button Handler* para reconocer el haberlo pulsado, y al ser pulsado genera *Event Trigger* que llaman a funciones del script Button Handler, para realizar la acción pulsada.
        - **PunchButton**: igual que el anterior pero para golpear.
    - **Joystick**: flechas que controlan el movimiento del personaje. Nuevamente se requiere el componente Mobile Control Rig.
        - **ArrowLeft**: botón que hacer mover al personaje a la izquierda. Para ello debe incluir el script *Axis Touch Button* con valor *Axis -1*.
        - **ArrowRight**: igual que el anterior pero en vez de Axis -1, *Axis 1* para moverse a la derecha.



- **PAUSE**: menú de Pausa que permite volver a la pantalla de inicio del juego (HOME), gestionar tanto la música como el sonido del juego, reanudar el juego y reiniciar el nivel.

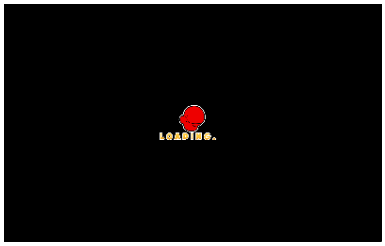
    - **pausa_home**: al ser pulsado llama a las funciones `LevelManager.GoToLevel(StartScreen)` para ir a la pantalla de inicio, y `LevelManager.Pause()` para descongelar el juego.
    - **pausa_music**: al ser pulsado llama a la función `GameManager.SoundManager.PlayMusic()` que activa la música en el caso de que no lo esté, y la desactiva en caso contrario. Incluye el script *MusicButton* que recupera la referencia del objeto GameManager al cambiar de niveles para poder ejecutar las funciones de SoundManager que se encuentra en el objeto GameManager como ya he explicado.
    - **pausa_sound**: igual que pausa_music, pero llamando a `GameManager.SoundManager.PlaySound()` e incluyendo el script *SoundButton*.
    - **pausa_panel**: incluye los botones de Continuar y Reiniciar nivel.
        - **btn_resume**: llama a las funciones `LevelManager.Pause()` y `menuManager.BackMenu()` que cierra el menú de PAUSE, mostrando el anterior que es el HUD.
        - **btn_ restart**: llama a las funciones `LevelManager.Pause()` y `LevelManager.LevelRestart()` que reinicia el nivel, perdiendo el progreso (reinicia mangos recogidos y enemigos derrotados) y empezando en el inicio del nivel de nuevo.


- **GAMEOVER**: menú de Game Over, que aparece al no tener vidas disponibles. Tiene dos botones, uno para salir y volver a la pantalla de inicio, y otro para continuar y reiniciar el nivel, recuperando vidas para seguir jugando.

    - **gameover_panel**: incluye los botones de Continuar y Salir.

- **btn_exit**: llama a la función `LevelManager.GoToLevel(StartScreen)` que vuelve a la pantalla de inicio del juego.
- **btn_restart**: llama a la función `LevelManager.LevelRestart()` que reinicia el nivel y el número de vidas disponibles.


- **FADER**: no se trata de un menú, sino una imagen negra que hace de transición entre niveles, contiene un sprite animado para informar al usuario que se está cargando el nivel.




- **EventSystem**: objeto que controla los eventos del sistema. Para ello contiene 3 componentes que se autogeneran con la creación del GameObject Canvas:

  - **Event System**: script que controla los eventos del sistema.
  - **Standalone Input Module**: script que controla los eventos relacionados con los dispositivos móviles.
  - **Touch Input Module**: script que recoge la información al tocar la pantalla del dispositivo móvil.

Por último, para que el nivel tenga música de fondo, deberá añadirse el objeto *BgMusicUniformMotionThereIsNoWay* o *BgMusicUniformMotionVictory*. He querido tener distintas música para la pantalla de inicio y para los niveles, por lo que en la primera he añadido la música "There Is No Way" y en los niveles "Victory". Estas músicas son propiedad de © Uniform Motion y pueden ser usadas mientras no exista ánimo de lucro. En un futuro las cambiaré si quiero obtener algún beneficio con el juego. Estos objetos contienen el script *Background Music* que reproduce la música de fondo[4].

[4] Sólo puede haber una música de fondo a la vez.

El nivel inicial **StartScreen** no se trata de un nivel sino de la pantalla de inicio del juego, esta pantalla contiene, un script *StartScreen* que contiene el código necesario para salir y cerrar la aplicación, y cargar los menús y niveles. En esta pantalla el botón de Atrás de los dispositivos móviles tienen la funcionalidad de cerrar menús y mostrar el menú de Salir. Aparte de lo obligatorio explicado anteriormente:

- **Main Camera**: cámara ortográfica principal del juego. Contiene:
    - **Parallax Camera**: script que permite tener parallax scroll en el fondo (objetos moviéndose a distinta velocidad).
    - **Bloom Optimized**: efecto de imagen que desenfoca el fondo para dar más importancia a lo importante del juego. Utiliza el *Shader FastBloom*.

- **TitleScreen**: contenido exclusivo de la pantalla de inicio.
    - **backGround**: fondo del juego.
    - **cloudsBack**: nubes de fondo que se mueven constantemente debido al script Parallax Title que las mueve a una velocidad de 0.01m/s modificando su posición, al igual que se hace con los mangos del HUD, modificando el atributo UV Rect del Raw Image.
    - **cloudsFront**: nubes de fondo que se mueven como cloudsBack, pero a velocidad 0.04m/s.
    - **Flora**: objetos del entorno como árboles, arbustos, etc..
    - **logo**: logo del juego.
    - **menu_default**: se compone del botón de Jugar (START) y el botón de Ajustes.
        - **btn_start**: al pulsarse llama a la función `menuManager.showMenu (menu_levels)`.
        - **btn_settings**: al pulsarse llama a la función `menuManager.showMenu (menu_settings)`.


    - **menu_settings**: menú de Ajustes, permite gestionar la música y sonido del juego.
        - **settings_close**: botón que llama a la función `menuManager.BackMenu()` que vuelve al *menu_default*.
        - **settings_music**: botón que activa/desactiva la música de fondo del juego, llamando a la función `SoundManager.PlayMusic()`.
        - **settings_sound**: botón que activa/desactiva los efectos de sonido del juego, llamando a la función `SoundManager.PlaySound()`.


- **menu_exit**: menú de Salir, permite salir y cerrar la aplicación.
  - **btn_option_si**: botón que llama a la función `StartScreen.Exit()`.
  - **btn_option_no**: botón que llama a la función `menuManager.BackMenu()` para cerrar el menú y mostrar el *menu_default*.


- **menu_levels**: menú de selección de nivel, ofrece una lista de los niveles del juego.
  - **levels_panel**: grid que contiene la lista de niveles. Se basa en el componente *Grid Layout Group* que define una tabla con filas y columnas, donde en cada uno hay un nivel manteniendo el formato de tabla. Cada uno llama a la función `StartScreen.GoToLevel(nivelSeleccionado)`.
  - **btn_back**: botón que cierra el menú de selección de nivel llamando a la función `menuManager.BackMenu()`.

# 7.3. Niveles del juego

Todos los **niveles** van a tener esta jerarquía:

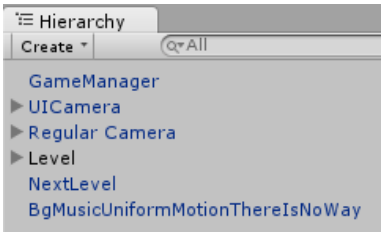



**Figura 12. Jerarquía básica de un nivel**

- **GameManager**: controla el juego.
- **UICamera**: cámara de la Interfaz de Usuario.
- **Regular Camera**: cámara que sigue al jugador. Contiene:
  - **Parallax Camera**: permite el movimiento del fondo a distintas velocidades.
  - **Bloom Optimized**: efecto de imagen que desenfoca el fondo para dar más importancia a lo importante del juego. Utiliza el *Shader FastBloom*.
  - **Contrast Enhance**: efecto de imagen que aumenta el brillo del nivel, para darle más color. Utiliza *los Shader SeparableBlur* y *ContrastComposite*.
  - **Background**: objeto que hace de background del nivel.
- **Level**: controla el nivel mediante el script *Level Manager*, que tiene los componentes:

**Player Prefab**: prefab del personaje, debe implementar *CharacterBehaviour*, Coco. Esto permite instanciar distintos prefabs sin modificar código.

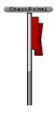
  - **Intro Fade Duration**: duración de la transición de entrada al nivel.
  - **Outro Fade Duration**: duración de la transición de salida del nivel.
  - **Mangos Max**: número máximo de mangos del nivel.

El objeto **Level** contiene todo el contenido del nivel, desde el controlador del nivel, hasta los bloques que dan forma al nivel, enemigos, mangos, etc..

  - **LevelManager**: controla el punto de spawn inicial y los límites del nivel.


**LevelStart**: punto de inicio del nivel, se trata del primer *Checkpoint*, por lo que implementa el script *Checkpoint*.

    - **LevelBounds**: límites del nivel definidos por un *Box Collider*, que adquieren funcionalidad mediante el script *Level Limits*. Se encuentra en la *Layer LevelBounds*, necesario establecerlo para comprobar si el jugador colisiona con alguno de ellos.



**Checkpoints**: contiene los checkpoints del nivel.

**Arrows**: contiene todas las flechas de dirección que ayudan al jugador a seguir el camino correcto hacia el final del nivel.

- **Platforms**: contiene:
  -  **Platforms**: plataformas del nivel, tanto fijas como en movimiento.
  -  **Grass**: bloques que componen el nivel.
  -  **Falling**: bloques que caen al estar sobre ellos cierto tiempo.
  -  **Flora**: árboles, arbustos, objetos de fondo que dan profundidad y naturaleza al nivel.
  -  **Fall**: zonas de caída al vacío.
  -  **Items**: contiene los mangos y vidas que pueden recogerse en el nivel.
  -  **Friends**: personajes amigos que nos ayudarán en nuestra aventura.
  -  **Enemies**: personajes enemigos que se interpondrán en nuestro camino.
- **NextLevel**: puerta al siguiente nivel.
- **BgMusic**: música de fondo.


## 7.4. Character (personaje)

El personaje del juego es **Coco**, será instanciado en cada nivel, con una serie de componentes:


- **coco**: se sitúa en la *Layer Player* que indica que se trata del objeto que será controlado por el jugador.
  - **SpriteRenderer**: imagen de Coco, la primera de la *animación de parado (idle)*. Se sitúa en la *Layer Platforms* para interactuar con las físicas de las plataformas.
  - **BoxCollider2D**: collider que define el área de colisión de Coco. Se establece *Is Trigger para* recibir eventos al colisionar con otro objeto.
  - **Rigidbody2D**: componente necesario para recibir físicas. Establezco *Fixed Angle* a true para que el personaje no rote ante cambios de ángulo, sino que permanezca recto en cuanto a su punto de gravedad.
  - **CocoController**: script que controla los parámetros básicos y físicas. Velocidad, gravedad, raycasting…

    Requiere que el objeto que contiene este script contenga el componente del tipo *BoxCollider2D* `[RequireComponent(typeof(BoxCollider2D))]`.

    Utiliza las clases:
    - `CocoControllerState`: controla el estado del objeto (colisionando, cayendo, en el suelo, etc..)
    - `CocoControllerParameters`: parámetros del objeto (velocidad, gravedad).

Para controlar las colisiones y físicas, Unity dispone de un sistema de *Raycasting*, que se basa en lanzar "rayos" en una dirección, buscando objetos con los que colisionar, en caso de encontrar alguno, notifica de ello. El *LayerMask* es usado para realizar una elección selectiva de los tipos de objeto de los que se quiere recibir notificaciones. Los objetos que implementen este script, tendrán *raycasts horizontales*, y *raycast verticales*; el atributo *RayOffset* indica hasta dónde se quiere comprobar si se colisiona con algo.

Este script controla el movimiento del personaje, añadiéndole las fuerzas necesarias para ello (`AddForce, AddHorizontalForce, AddVerticalForce, SetForce`, etc..).

El método `LateUpdate()`, que como he explicado anteriormente se ejecuta tras el método `Update()`, aplica la fuerza de la gravedad `_speed.y += (Parameters.Gravity + _movingPlatformsCurrentGravity) * Time.deltaTime`, mueve al objeto actualizanco su posición `_newPosition = Speed * Time.deltaTime`, añade los raycasts al objeto mediante los métodos `CastRaysToTheSides, CastRaysBelow, CastRaysAbove` para ello me he ayudado de una clase ya creada por la comunidad de Unity que controla los raycasts, la clase es `CorgiTools`, y el método que permite generar los raycast es `CorgiTools.CorgiRayCast`.

El método `SetRaysParameters()` es llamado para generar un rectángulo de raycasts según el *BoxCollider2D* del objeto. A su vez, se dibuja este rectángulo en el modo `Debug` para visualizarlo durante el desarrollo del proyecto, mediante el método `Debug.DrawLine`.

Para añadir distintos volúmenes de físicas (*agua*), quien debe implementar la clase `CocoControllerPhysicsVolume2D`, si el objeto que contiene el script *CocoController* colisiona con un objeto que contiene el script de volúmenes, éste adquiere las propiedades del volumen mediante los parámetros del objeto.

```
_overrideParameters = parameters.ControllerParameters;
```

Al salir del volumen, el objeto recupera sus parámetros.

```
_overrideParameters = null;
```

- **CharacterBehavior**: script que convierte al objeto en el personaje a controlar. El objeto debe tener un *BoxCollider2D* en la cabeza para comprobar si se colisiona con algo. Contiene las *vidas* del personaje, la *velocidad de movimiento* da cada estado (correr, saltar, pared), los *parámetros de salto* (altura, número de saltos) y la fuerza de rozamiento al colisionar con una pared en salto. También especifica los *permisos*, o acciones que puede realizar, como son Saltar, Golpear, Saltar en la pared, o Deslizarse por la pared. Incluye el efecto al tocar el suelo, que se trata de partículas creadas con el sistema de partículas de Unity, estas partículas implementan los scripts *AutoDestroyParticleSystem* que destruye las partículas una vez han sido generadas, y *VisibleParticle* que permite ver las partículas en el juego. Finalmente contiene los efectos de sonido que realiza el personaje (saltar, golpear, ser golpeado).
  Este script hereda de las clases `MonoBehaviour` y `CanTakeDamage`, la primera ya he explicado por qué en la Introducción a Unity3D, y la segunda para permitir recibir daño de los enemigos. Implementa la clase `CharacterBehaviorState` que contiene los estados de Coco (saltando, golpeando, ca-

yendo, muerto, en zona de diálogo, etc…); la clase `CharacterBehaviorParameters` que define los parámetros explicados; y la clase `CharacterBehaviorPermissions` que define los permisos. Las partículas al tocar el suelo son del tipo `ParticleSystem` y los efectos de sonido son `AudioClip`.
En los parámetros se define el número de saltos permitido `en CharacterBehaviorParameters.JumpBehavior`, esto lo he hecho así para si en el futuro quiero poder realizar un mayor o menor número de saltos en algún nivel, o dar saltos infinitos o ninguno.

- El método `public static CharacterBehavior Instance` permite referenciar la clase desde cualquier lugar, para poder acceder a sus métodos y atributos.
- En el método *`Awake()`* se inicializa lo necesario, los estados, la cámara y el controlador.
- En el método *`Start()`* se inicializa el componente `Animator`, la fuerza de la gravedad a aplicar, y los estados por defecto.
- En el método *`Update()`* que se ejecuta cada frame, se comprueba si hay que actualizar el estado de Coco; en caso de estar en el estado *IsDead*, se impide mover al jugador. Si se presiona el botón de salto, se controla la presión del botón, que controla la altura de salto (`_jumpButtonPressTime`).
- El método *`LateUpdate()`* se encarga de resetear el número de saltos disponibles tras tocar el suelo.
- El método *`JumpStart()`* realiza el salto, reduciendo en 1 el número de saltos restantes, y reproduce el sonido de saltar.

```
SoundManager.Instance.PlaySound(PlayerJumpSfx, transform.position);
```

En caso de bajar por la pared, aplica la fuerza de rozamiento.

- El método *`Punch()`* reproduce el sonido de golpear.

```
SoundManager.Instance.PlaySound(PlayerPunchSfx, transform.position);
```

- El método *`Kill()`* realiza lo necesario al morir el jugador, esto es, indicar el estado como *IsDead*, restar una vida si hay disponibles, si no se muestra el menú de GameOver.

```
if(Lives >= 0){ Lives--;}
if(Lives < 0)  {menuManager.Instance.ShowMenu(GUIManager.Instance.GAMEOVER.GetComponent<menu>());}
```

Se resetean los parámetros y se aplica una fuerza vertical como animación de morir.

```
_controller.SetForce(new Vector2(0, 25));
```

- El método *`RespawnAt(Transform spawnPoint)`* respawnea (renace) al jugador en el spawnPoint indicado. Lo primero es girar al personaje a la derecha en caso de haber muerto mirando a la izquierda, para ello está el método *`Flip()`* que únicamente invierte los valores del transform del objeto:

```
transform.localScale = new Vector3(-transform.localScale.x, transform.localScale.y, transform.localScale.z);
```

Establece el estado a vido `IsDead = false`, resetea las colisiones y se establece la posición del objeto a la del spawnPoint.

```
transform.position = spawnPoint.position;
transform.position = new Vector3(transform.position.x, trans-
form.position.y + 2, transform.position.z + 1);
```

- El método *TakeDamage(int damage, GameObject instigator)* aplica el daño (damage) del enemigo (instigator) al jugador. En él, se reproduce el sonido de ser golpeado:

```
SoundManager.Instance.PlaySound(PlayerHitSfx, transform.posi-
tion);
```

Se indican las Layer que pueden realizar daño, que son las de enemigos (13) y proyectiles (9). También se reproduce el efecto de recibir daño, que consiste en un parpadeo de color rojo, indicando que ha sido golpeado. Finalmente mata al personaje con el método `LevelManager.Instance.KillPlayer()`.

- **Animator**: animaciones del personaje.
- **PlayerBounds**: script que controla lo que pasa cuando se colisiona con los límites del nivel. Para ello se consigue la referencia a los límites del nivel:

```
_bounds = GameObject.FindGameObjectWithTag("LevelBounds").Get-
Component<BoxCollider2D>();
```

Puede indicarse que:

- **Nothing**: no hacer nada cuando colisiona.
- **Constrain**: mantener al jugador dentro de los límites.
- **Kill**: matar al jugador.

Dependiendo de lo establecido que se quiere que pase cuando se colisiona por Arriba, Abajo, Izquierda o Derecha, se llama al método:

```
ApplyBoundsBehavior(BoundsBehavior behavior, Vector2 constrai-
nedPosition
```

Si se establece que al tocar un límite debe morir el jugador:

```
if(behavior == BoundsBehavior.Kill) {LevelManager.In-
stance.KillPlayer();}
```

Si no, mantiene al jugador dentro de los límites del nivel:

```
transform.position = constrainedPosition;
```

- **CharacterPunch**: script que añade la funcionalidad de golpear al objeto. Para ello crea un área de golpeo mediante un collider2D. El método `Punch()` establece el estado de golpear, activa el área de golpeo y reproduce el efecto de golpeo:

```
_characterBehavior.BehaviorState.PunchAttacking =
true;
PunchCollider.SetActive(true);
_characterBehavior.Punch();
```

Se espera los segundos indicados en la duración del golpeo, y se desactiva el collider.

# 7.5. Friend (amigo)

El objeto Friend va a contener una zona de diálogo para dar consejos al jugador:


- **Friend**:
  - **SpriteRenderer**: sprite, la primera de la *animación de parado (idle)*.
  - **BoxCollider2D**: collider que define el área de colisión. Se establece *Is Trigger para* recibir eventos al colisionar con otro objeto.
  - **Animator**: animación del objeto.
  - **CocoController**: script que controla los parámetros básicos y físicas. Velocidad, gravedad, raycasting…
- **DialogueZone**: zona de diálogo que muestra distintos mensajes.
  - **BoxCollider2D**: zona en la que se reproduce el diálogo con el jugador.
  - **DialogueZone**: script que controla el diálogo.

  Requiere que el objeto que contiene este script contenga el componente del tipo *BoxCollider2D* `[RequireComponent(typeof(BoxCollider2D))]`.
  Establece una zona en la que aparecerá una caja de diálogo (`DialogueBox`) con mensajes de ayuda. Controla el color del texto y del fondo de la caja, la velocidad de transición entre mensajes y el tiempo en caso de no activar *Button Handled*, que indica que no se requiere pulsar el botón A para mostrar el siguiente mensaje. También se permite indicar si se desea poder moverse mientras se está en el diálogo, y finalmente el contenido del diálogo, que se almacena en un array de Strings.
  El método *`Start()`* instancia la zona de diálogo y establece el array de Strings al inicio (`_currentIndex = 0`). Si se ha especificado que se muestre siempre la información de que debe pulsarse el botón A para pasar al siguiente mensaje, se muestra el botón con el método *`ShowPrompt()`* este método recoge el objeto buttonA, que se encuentra en *GUI/btnA*, y lo instancia: `(GameObject)Instantiate(Resources.Load("GUI/btnA"))`
  Se instancia de manera suave mediante una transición, usando el método anteriormente mencionado *`CorgiTools.FadeSprite`* y estableciendo **alpha a 1** `new Color(1f, 1f, 1f, 1f)`.
  Para ocultar el botón, se llama al método *`HidePrompt()`* que realiza el inverso al anterior, es decir, estableciendo el **apha a 0** `new Color(1f, 1f, 1f, 0f)` y destruyendo el objeto con el método de Unity `Destroy(buttonA)`.

  El método *`StartDialogue()`* será llamado por la clase `InputManager` si se presiona el botón A dentro de la zona de diálogo:
  ```
  if(CrossPlatformInputManager.GetButtonDown("Jump")){_player.BehaviorState.CurrentDialogueZone.StartDialogue();}
  ```
  Si se presiona el botón B, se sale del diálogo estableciendo `_currentIndex = Dialogue.Length:`
  ```
  if(CrossPlatformInputManager.GetButtonDown("Punch")){_player.BehaviorState.CurrentDialogueZone.ExitDialogue();}
  ```
  Una vez empezado el diálogo, se elimina el botón A encima de la zona, y aparece la caja de diálogo. Se congela al jugador si se ha establecido el no poder moverse durante el diálogo:

```
GameManager.Instance.FreezeCharacter()
```
La caja de diálogo se instancia desde *GUI/DialogueBox*:
```
GameObject dialogueObject = (GameObject)Instantiate(Resources.Load("GUI/DialogueBox"))
```
Y se le aplica el color del texto y de la caja establecidos anteriormente.
Se llama al siguiente mensaje del diálogo mediante el método `PlayNextDialogue()`:

- **Es el primer mensaje**: se muestra el texto:
  `_dialogueBox.DialogueText.text = Dialogue[_currentIndex]`
- **No es el primer mensaje** *(`_currentIndex != 0`)*: espera el tiempo especificado para mostrar el texto:
  ```
  yield return new WaitForSeconds(TransitionTime)
  _dialogueBox.DialogueText.text = Dialogue[_currentIndex]
  ```
- **Es el último mensaje** *(`_currentIndex >= Dialogue.Length`)*: resetea la caja de diálogo estableciendo `_currentIndex = 0`. Destruye la caja y permite de nuevo moverse al jugador.

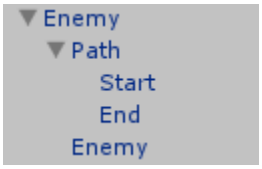
Si se ha especificado que se pueda activar más de una vez, se espera el tiempo necesario y se vuelve a activar con el método *`Reactivate()`*.

En caso de no requerir el pulsar el botónA para mostrar el siguiente mensaje, se llama a la función `AutoNextDialogue()`, que mostrará los mensajes automáticamente con un tiempo entre mensajes.

## 7.6. Enemies (enemigos)

Hay distintos tipos de enemigos, unos que atacan al ver al jugador, otros que lanzan proyectiles, unos pueden ser derrotados saltando sobre ellos, otros sólo siendo golpeados y alguno no puede recibir daño. Todo esto depende de los scripts que tengan adheridos:

**Enemy**: sigue un camino y mata al jugador si este colisiona con él y puede ser derrotado al ser pisado o golpeado.




- **Enemy**:
  - **SpriteRenderer**: sprite, la primera de la *animación de parado (idle)*.
  - **BoxCollider2D**: collider que define el área de colisión. Se establece *Is Trigger para* recibir eventos al colisionar con otro objeto.
  - **Rigidbody2D**: componente para recibir físicas del juego.
  - **Animator**: animación del objeto.
  - **CocoController**: script que controla los parámetros básicos y físicas. Velocidad, gravedad, raycasting…
  - **Health**: script que añade vida a un objeto. Se establece la cantidad de vida que tiene (`CurrentHealth`), si suelta algún Mango al jugador al morir (`PointsWhenDestroyed`) y los efectos de recibir daño, morir y el soltar Mangos. El método `TakeDamage(int damage, GameObject`

`instigator)`, instancia el efecto de ser golpeado, y resta el daño recibido a la vida que tenga; si la vida es menor o igual a 0, significa que está muerto, por lo que se destruye el objeto y se instancia el efecto de soltar Mangos en caso de tener que hacerlo, sumando los Mangos al juego.

- **GiveDamageToPlayer**: este script permite hacer daño al jugador. Se establece la cantidad de daño a realizar (`DamageToGive`). Al colisionar con el jugador, al tener un Box Collider2D y el atributo *`Is Trigger`* activado, se llama a la función `OnTriggerEnter2D(Collider2D collider)`, que primero comprueba si es el jugador quien colisiona comprobando el *tag* del objeto (`collider.tag != "Player"`), en caso que sea el jugador, empuja al jugador y le hace parpadear para indicar que ha sido golpeado, y se llama a la función de la clase `CharacterBehavior TakeDamage(DamageToGive, gameObject).`
- **Stompable**: este script permite a un objeto recibir daño al ser pisado, requiere de los componentes `[RequireComponent(typeof(BoxCollider2D))]` para poder generar los raycasts que comprueben si se es pisado, y `[RequireComponent(typeof(Health))]` para poder reducir la vida del objeto al ser pisado. `NumberOfRays` indica el número de raycasts generados, `KnockbackForce` la fuerza que se aplicará al jugador al pisar el objeto y `DamagePerStomp` el daño que se aplica al objeto al ser pisado. Este script se basa en el método `CastRaysAbove()` que genera raycasts encima del objeto para comprobar si se es pisado `hitConnected = true`, entonces se comprueba si es el jugador quien ha pisado el objeto, y en caso afirmativo se aplica la fuerza indicada y el daño al objeto:

  ```
  cocoController.SetVerticalForce(KnockbackForce);
  _health.TakeDamage(DamagePerStomp, gameObject);
  ```

- **IARespawn**: script que hace renacer al objeto si había muerto pero el jugador muere antes de llegar a un checkpoint o al final del nivel. Para ello debe implementar el método `onPlayerRespawnInThisCheckpoint(CheckPoint checkpoint, CharacterBehavior player)` de la clase `IPlayerRespawnListener`, para almacenarse en el array de checkpoints (explicado a continuación). Este método guarda la posición inicial del objeto, su tamaño y activa el objeto al renacer.

  ```
  _direction  = new Vector2(-1, 0);
  transform.localScale = new Vector3(1, 1, 1);
  transform.position = _startPosition;
  gameObject.SetActive(true);
  ```

- **PathFollow**: script que hace a un objeto seguir un camino establecido, debe referenciarse el objeto **Path**. Se puede elegir el tipo de seguimiento, si poco a poco (`MoveTowards`) o de golpe (`Lerp`), por ahora he usado `MoveTowards`, pero he programado `Lerp` porque tengo pensado usarlo en el futuro. Se establece la velocidad del objeto y la distancia máxima a la que llega en relación a los extremos del camino, esto permite simular paradas si se quiere. Primero se comprueba que el Path existe y tiene algún punto. En caso afirmativo, el objeto modifica su

posición según el `_currentPoint`, que es el punto del camino, para modificar esta posición, Unity3D dispone de las funciones:

```
transform.position = Vector3.MoveTowards(transform.position, _currentPoint.Current.position, Time.deltaTime * Speed)
transform.position = Vector3.Lerp(transform.position, _currentPoint.Current.position, Time.deltaTime * Speed)
```

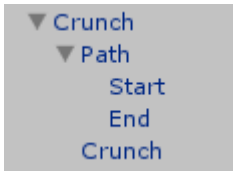
A su vez se va recorriendo el camino con la función preestablecida:

```
_currentPoint.MoveNext()
```

- **Path**: objeto que contiene el script *Path Definition*, que crea un camino mediante un conjunto de puntos establecidos por un array de Transforms (objetos) `Transform[] Points`, que sólo contienen el componente *Transform*, indicando la posición del punto. Para ayudar a la hora de desarrollar a visualizar el camino, se dibuja mediante el método de Unity `OnDrawGizmos()`, y la función:

```
Gizmos.DrawLine(points[i - 1].position, points[i].position);
```

**Crunch**: sigue un camino y ataca al jugador si está en su rango de visión, le mata si este colisiona con él y puede ser derrotado al ser pisado o golpeado.




- **Crunch**:
  - **Componentes de Enemy**.
  - **IA Shoot On Sight**: ataca al jugador si se encuentra dentro del rango de visión (`ShootDistance`) tiene un tiempo entre ataques de `FireRate`. Comprueba cada frame si el jugador se encuentra dentro de la distancia de ataque, para ello se hace uso de raycasts:

    ```
    RaycastHit2D raycast = Physics2D.Raycast(raycastOrigin, _direction, ShootDistance, 1<<LayerMask.NameToLayer("Player"))
    ```

    Si se colisiona con el jugador, se informa de ello, actualizando la animación a atacar, e iniciando el tiempo de espera al siguiente ataque:

    ```
    _animator.SetBool("attack", true);
    _canFireIn = FireRate;
    ```
- **Path**.

**RocketCannon**: sigue un camino y ataca al jugador si está en su rango de visión, le mata si este colisiona con él y puede ser derrotado al ser pisado o golpeado.


- **RocketCannon**:
  - **Sprite Renderer**: sprite.
  - **Box Collider**: collider que define el área de colisión. Este objeto no puede ser destruido por el jugador.
- **ProjectileSpawner**: contiene el script *PathedProjectileSpawner*, que convierte al objeto en un generador de proyectiles, define el destino de los proyectiles (`Destination`), qué proyectiles van a generarse (`Projectile`), el efecto de generar los proyectiles (`SpawnEffect`), y la velocidad (`Speed`) y tiempo entre proyectiles (`FireRate`).

Los proyectiles a generar se tratan del **prefab cannonball**, que incluye:

- *PathedProjectile*: script que mueve al proyectil por los puntos del camino hasta el punto *Destination*, y al llegar a este punto se destruye generando un efecto (*RocketExplosion*).
- *GiveDamageToPlayer*.
- *BoxCollider2D*.
- *SpriteRenderer*.
- *Health*: puede ser destruido si se es golpeado.

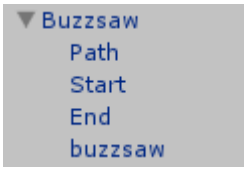
El script comprueba si debe generar un proyectil, según la variable `_nextShotInSeconds` que indica el tiempo entre proyectiles. En caso afirmativo, instancia el prefab cannonball indicándole la velocidad y destino, así como el efecto al ser generado:

```
var projectile = (PathedProjectile) Instantiate(Projectile, transform.position,transform.rotation);
projectile.Initialize(Destination, Speed);
if(SpawnEffect != null) {Instantiate(SpawnEffect, transform.position, transform.rotation);}
```

Al igual que con los caminos, se dibuja una línea para indicar el camino del cañón para facilitar el trabajo a la hora de desarrollar el juego, con la función `OnDrawGizmos()`.

- **Destination**: objeto con componente *Transform* que indica el punto final del camino de los proyectiles.

**Buzzsaw**: cuchillas que siguen un camino y matan al jugador si las toca, no pueden ser destruidas.




- **Buzzsaw**:
  - **Sprite Renderer**: sprite.
  - **PathFollow**: camino a seguir.
  - **CircleCollider2D**: collider que define el área de colisión. Se establece *`Is Trigger`* *para* recibir eventos al colisionar con otro objeto.
  - **Rigidbody2D**: componente para recibir físicas del juego.
  - **AutoRotate**: script que hace rotar automáticamente al objeto. Lo único que hace es rotar el objeto cada frame a una velocidad (`rotationSpeed`), llamando a la función:
    `transform.Rotate(rotationSpeed * Vector3.forward * Time.deltaTime)`
  - **GiveDamageToPlayer**.
- **Path**: camino que sigue el objeto.
- **Start**: punto de inicio del objeto.
- **End**: punto de destino del objeto.

# 7.7. Otros

En los niveles va a haber objetos como Checkpoints, Agua, Saltadores, etc..:

 **Checkpoint**: punto de guardado del nivel que permite renacer en él al morir, en vez de en el inicio del nivel.


- **Checkpoint**:
  - **SpriteRenderer**.
  - **Animator**.
  - **BoxCollider2D**.
  - **CheckPoint**: script que convierte al objeto en un checkpoint. Almacena los objetos previos al checkpoint, es decir, los mangos, vidas y enemigos que hayan sido recogidos/destruidos, después de pasar por el checkpoint, que implementan la clase `IPlayerRespawnListener`. Son buscados y guardados en una lista de este tipo:
    `List<IPlayerRespawnListener>`
    Lo que hace este script es activar de nuevo los objetos que hayan sido recogidos entre checkpoints, mediante la función:
    ```
    foreach(var listener in _listeners){listener.onPlayer-
    RespawnInThisCheckpoint(this, player);}
    ```
    Este script también incluye la función que permite añadir un objeto a la lista:
    ```
    AssignObjectToCheckPoint(IPlayerRespawnListener liste-
    ner)
    ```

 **Falling Platform**: plataforma que cae al vacío a una velocidad (`FallSpeed`) cuando el jugador se sitúa sobre ella durante un tiempo (`TimeBeforeFall`).

- **FallingPlatform**: se basa en modificar la posición del objeto cuando el jugador se sitúa encima de ella:
  ```
  _newPosition = new Vector2(0, - FallSpeed*Time.deltaTime);
  transform.Translate(_newPosition, Space.World);
  ```
  Esto ocurrirá cuando el jugador se sitúe encima, notificará al script de que se ha pisado y empezará a caer.

 **Jumper (saltador)**: objeto que hace saltar al jugador por los aires, permitiéndole llegar a sitios a los que no puede llegar saltando.

- **Jumper**: se basa en añadir fuerza vertical (`JumpPlatformBoost`) al jugador cuando éste colisiona con él.
  ```
  controller.SetVerticalForce(Mathf.Sqrt(2f * JumpPlatformBoost
  * -controller.Parameters.Gravity));
  ```

**Water (agua)**: volumen que modifica los parámetros del jugador, dándole la sensación de estar nadando.


- **water**:
  - **SpriteRenderer**.
  - **BoxCollider2D**.
  - **Water**: script que modifica los parámetros del componente *CocoController* del jugador, por los del script que tiene adherido *CocoControllerPhysicsVolume2D*. Reproduce un efecto de entrar/salir del agua `Splash()`, y aplica la fuerza necesaria para salir del agua:
    `controller.SetVerticalForce(Mathf.Abs(WaterExitForce));`

# Capítulo 8
# Pruebas

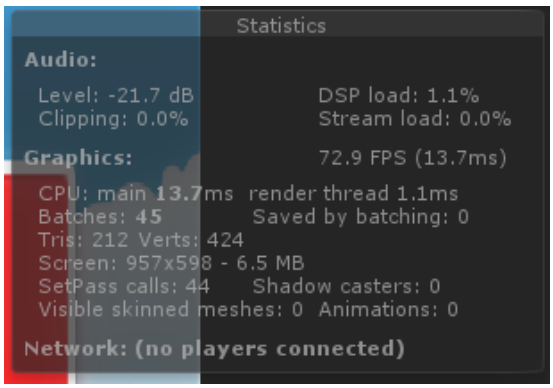
En este capítulo se realizan varias pruebas para validar y comprobar el funcionamiento de la aplicación. Al haber utilizado la metodología incremental, cada iteración debía realizar pruebas sobre lo implementado. Se aprovechan las *Estadísticas de Unity3D* que permiten conocer el comportamiento del dispositivo durante la ejecución del videojuego.

El principal objetivo es ofrecer un buen tiempo de respuesta al jugador, permitiendo jugar sin ningún tipo de restricción. Debido a tratarse de un videojuego, el proceso de pruebas es continuo, se detallan los resultados de las pruebas sobre los eventos más importantes del juego.

Las pruebas se han realizado con un dispositivo móvil THLW100 con versión 4.2.1. Android.

| **Prueba** | **PR-01 Abrir menú** |
|---|---|
| **Prerrequisito** | |
| **Procedimiento** | 1. Pulsar el botón START en la pantalla de inicio. |
| **Resultado esperado** | Mostrar el menú de selección de nivel en un tiempo inferior a medio segundo. |
| **Resultado obtenido** | El menú es mostrado en 1.1ms |

**Tabla 108. PR-01 Abrir menú**




*Estos resultados se aplican a todos los eventos que abren algún menú, debido a que se basan en la misma funcionalidad.

| Prueba | PR-02 Cargar nivel |
|---|---|
| Prerrequisito | |
| Procedimiento | 1. Seleccionar un nivel de la lista de niveles. |
| Resultado esperado | Cargar el nivel en tiempo inferior a 2 segundos. |
| Resultado obtenido | El nivel 1 es cargado en 88.0ms<br>El nivel 2 es cargado en 417.8ms<br>El nivel 3 es cargado en 1811.4ms |

**Tabla 109. PR-02 Cargar nivel**


| Prueba | PR-03 Movimiento del personaje |
|---|---|
| Prerrequisito | |
| Procedimiento | 1. Pulsar los controles de movimiento y acción del personaje. |
| Resultado esperado | Mover al personaje en el mínimo tiempo de respuesta posible. |
| Resultado obtenido | El personaje responde al evento en 1.0ms, tanto para los movimientos como para las acciones de saltar y golpear. |

**Tabla 110. PR-03 Movimiento del personaje**

Se comprueba que los **FPS** se mantienen entre **60-70**, lo que indica el buen rendimiento del juego.

Al tratarse de un videojuego, las pruebas se basan en probar las funcionalidades buscando errores. Se han probado todas las funcionalidades y comprobado su correcto funcionamiento, mostrar diálogos, atacar a un enemigo, ser atacado por un enemigo, etc..

# Capítulo 9
# Manual de Instalación

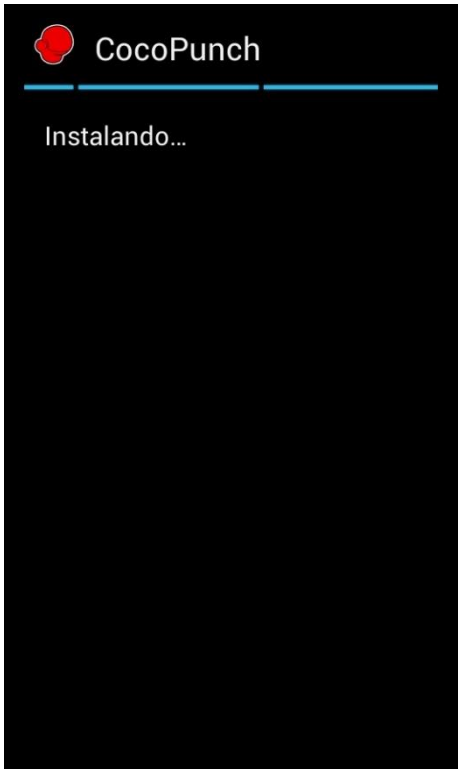
En este capítulo se explicará cómo instalar la aplicación para poder ser usada.

Lo primero es copiar el archivo **cocoPunch.apk** a nuestro dispositivo móvil (en la carpeta *Descargas/Downloads* por ejemplo), ya sea mediante USB, Bluetooth o e-mail. Una vez se tenga el archivo, seguir estos pasos:

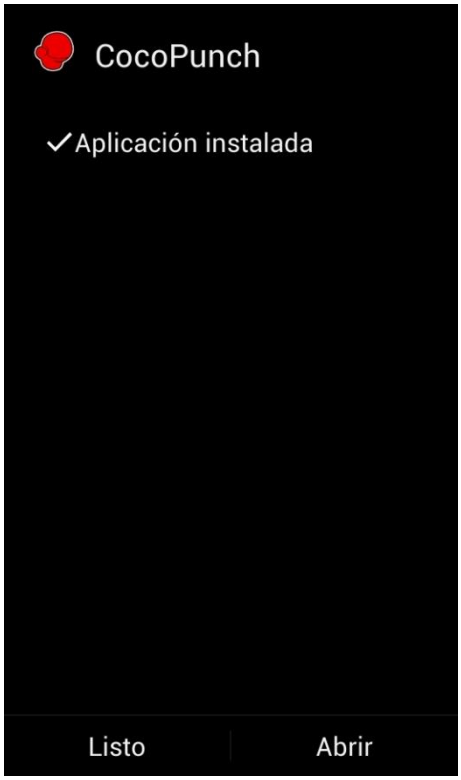
1. Ejecutar el archivo **cocoPunch.apk**:


Aparecerá una pantalla preguntándonos si deseamos instalar la aplicación, la cual no requiere ningún permiso especial a parte de disponer del espacio de almacenamiento disponible. Pulsar **Instalar** para continuar con la instalación:

2. Esperar a que la aplicación se instale en nuestro dispositivo móvil:

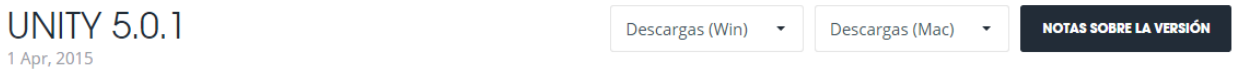



3. Al finalizar se mostrará un mensaje de confirmación, pudiendo cerrar el proceso de instalación pulsando **Listo**, o **Abrir** la aplicación:



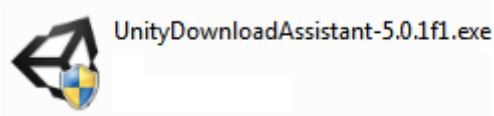
En caso de no disponer de dispositivo móvil Android, puede ejecutarse el juego desde Unity3D, para ello, seguir los siguientes pasos:

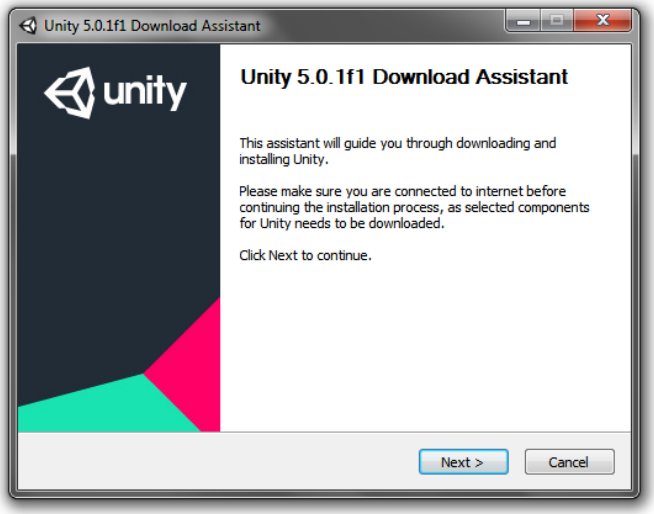
1. **Descargar** la versión 5.0.1 de Unity3D desde la página oficial: https://unity3d.com/es/get-unity/download/archive




Se ha trabajado en esta versión y no se ha actualizado a la última debido a haber comprobado que tiene fallos a la hora de ser exportado para Android.

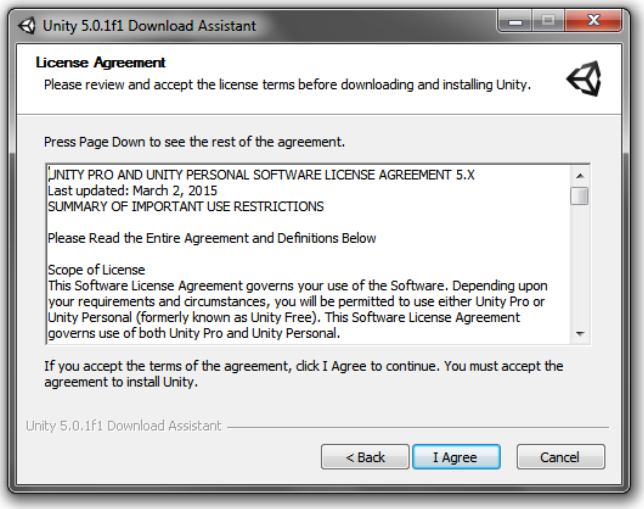
2. **Ejecutar** el ejecutable descargado:



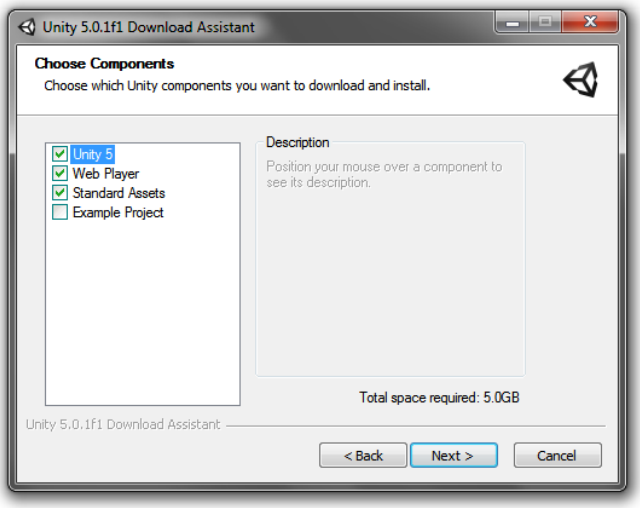

Pulsar en **Siguiente** 

:

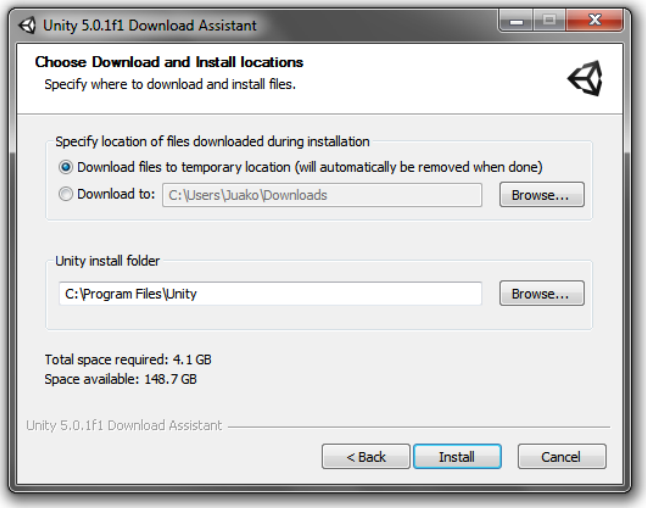


**Aceptar** los Términos y Condiciones 

:

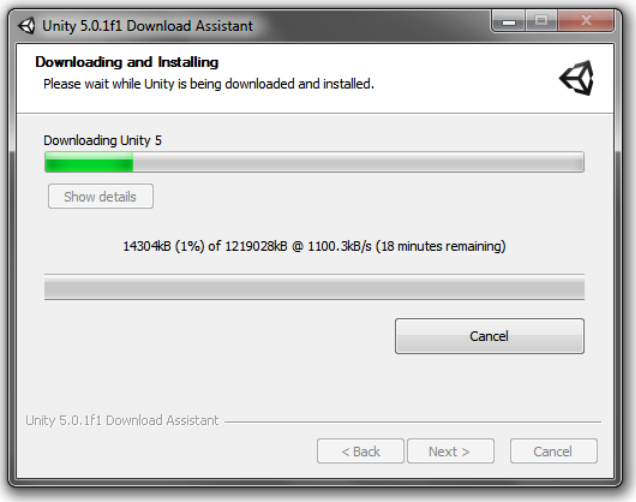



**Seleccionar** los componentes a instalar, dejar las 3 marcadas por defecto, y pulsar en **Siguiente** 

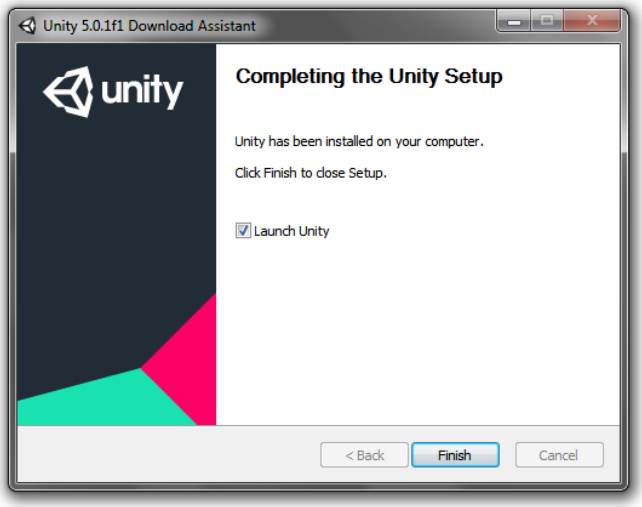

Dejar la primera opción seleccionada. Especificar el directorio donde se quiera que sea instalado el programa, y pulsar en **Instalar** 

:



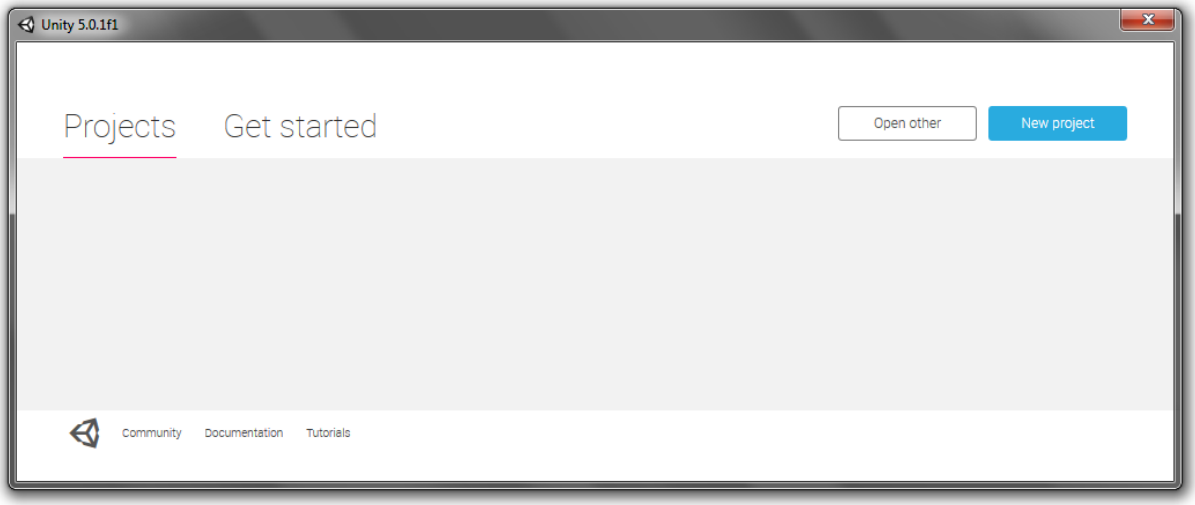

Esperar a que se descargue e instale por completo UNITY 5.0.1.

Abrir el programa:

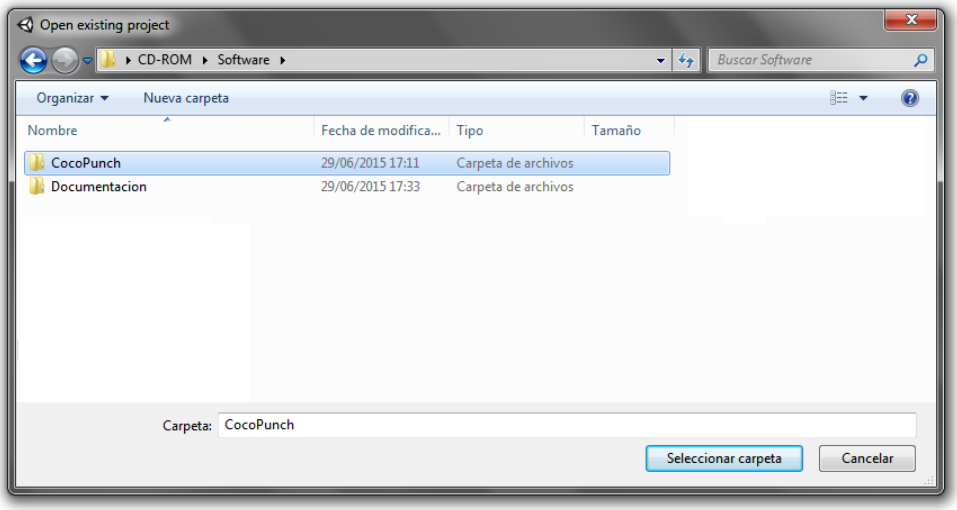



**3.** Abrir el proyecto pulsando en **Abrir Otro** 

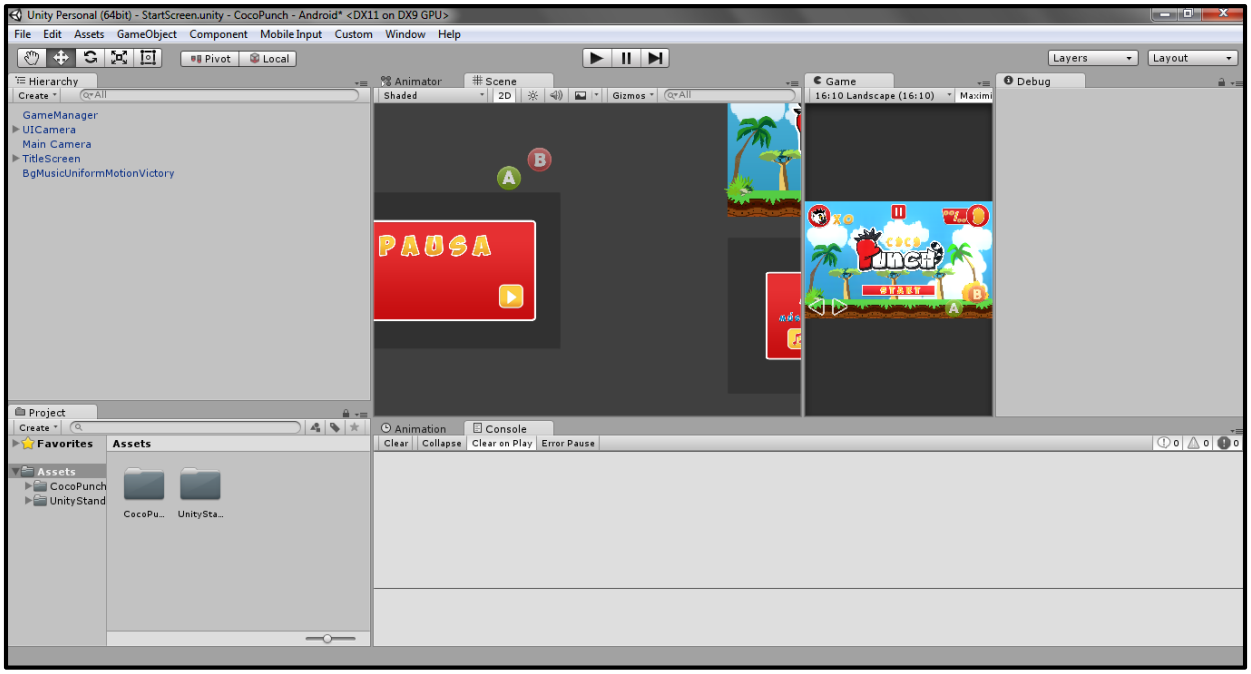
:



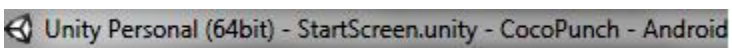
**Seleccionar la carpeta** del proyecto CocoPunch, ubicado en *CD-ROM/Software*, y pulsar Seleccionar carpeta:




Se mostrará la Interfaz de Unity, explicada en el capítulo de Introducción a Unity3D:




Comprobar que se tiene abierta la pantalla de inicio del juego, para ello mirar en la barra superior de Unity3D. Debe aparecer la siguiente información:




**4.** Ejecutar el proyecto pulsando en el botón de **Play** ▶.

# Capítulo 10
# Manual de Usuario

En este capítulo va a mostrarse el funcionamiento de la aplicación, demostrando cómo probar todas sus funcionalidades.

Al abrir la aplicación se mostrará un *splash screen* (pantalla de inicio que dura unos segundos). Al no disponer de la versión pro de Unity3D, se muestra por defecto el haberse desarrollado con este motor:


Después de esperar esos segundos, se mostrará la pantalla inicial del juego, en la que podremos gestionar los **Ajustes del juego** pulsando el *botón de Ajustes* de la esquina superior izquierda, **Salir de la aplicación** pulsando el *botón de Atrás* del dispositivo Android, o mostrar el **menú de Selección de nivel** pulsando el *botón START*:

1. Si se pulsa el botón de **Ajustes**, se muestra el *menú de Ajustes*. Si pulsamos el botón de **Música** y está activada, se desactiva, y viceversa, lo mismo con el botón de **Sonido**:

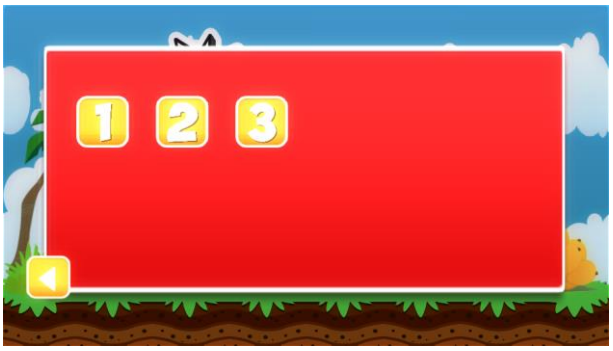

Se puede cerrar este menú pulsando la **X**, o pulsando el botón de **Atrás** del dispositivo, volviendo a la pantalla de inicio.

2. Si se pulsa el botón de **Atrás** estando en la pantalla de inicio, se muestra el *menú de Salir* de la aplicación:


Si se presiona **SI**, se sale de la aplicación.

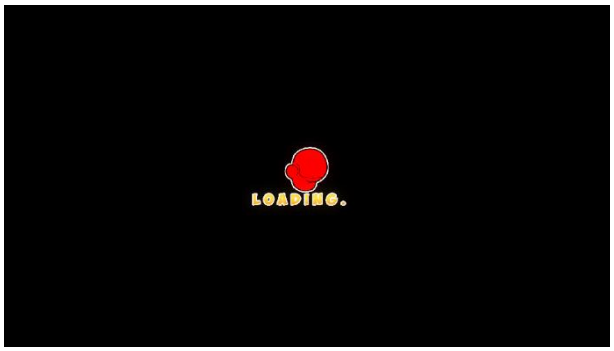
Si en caso contrario se presiona **NO**, se cierra el menú de Salir y se muestra la pantalla de inicio.

3. Si se pulsa el botón **START**, se muestra el *menú de Selección de nivel*:




Si se presiona el botón **Atrás** del dispositivo o el de **Volver** de la esquina inferior izquierda, se cierra el menú de Selección de nivel y se muestra la pantalla de inicio.

Si se selecciona uno de los **niveles** disponibles, el sistema cargará el nivel seleccionado. Durante las partes en que el sistema está cargando, se muestra una pantalla indicándolo para que el usuario no piense que la aplicación no responde:




Al cargar el nivel, tendremos una visión del juego, en el cual podremos realizar varias cosas a parte del hecho de jugar. Los niveles disponen de unas *flechas* en la esquina inferior izquierda para **mover al personaje** a izquierda o a derecha. También pueden realizar las acciones de **Saltar** con el *botón A* y **Golpear** con el *botón B* de la esquina inferior derecha. Está habilitada la opción de **doble salto**, pudiendo presionar una segunda vez el botón de Saltar en el aire una vez presionado ya el mismo botón. El juego dispone de un *menú de Pausa*, que puede abrirse pulsando el botón de **Pausa** de la parte superior centro, lo que pausará el juego, impidiendo moverse al jugador y a los enemigos, y mostrando varias opciones.


1. Si se pulsa el botón de **Pausa**, se muestra el *menú de Pausa*, congelando el juego y mostrando varias opciones:


Si se presiona el botón de **Salir** de la esquina superior izquierda, el juego volverá a la pantalla de inicio, perdiendo el progreso del nivel.

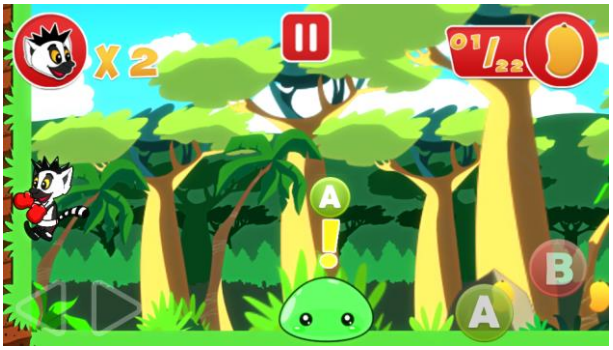
Si se presiona el botón de **Sonido** de su derecha, con forma de altavoz de sonido, se activará/desactivará el sonido del juego, dependiendo de su estado.

Si se presiona el botón de **Reiniciar** de la esquina inferior izquierda, se reiniciará el nivel, perdiendo el progreso.

Si se presiona el botón de **Continuar** de la esquina inferior derecha, se continuará el juego, cerrando el menú de Pausa y permitiendo de nuevo moverse al jugador.

2. Si se pulsa el botón de **Saltar**, el jugador *saltará*. Puede pulsarse una vez más para dar un *doble salto*:


Si se pulsa el botón de **Saltar** cuando se está en una pared, se podrá saltar nuevamente con el botón A:



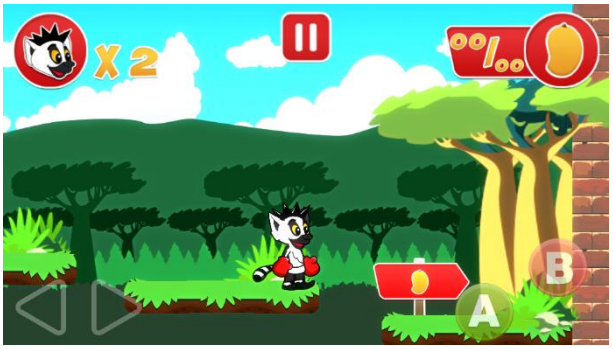

1. Si se pulsa el botón de **Golpear**, el jugador *golpeará* para dañar a los enemigos/objetos con vida:


En los niveles podemos encontrarnos con unos bichos verdes que tienen una ! encima, esto indica que al pulsar el *botón A* a su lado, empezará un **diálogo** que nos proporcionará ayuda:


Si seguimos presionando el botón **A**, se irán mostrando los distintos mensajes del diálogo hasta su finalización.

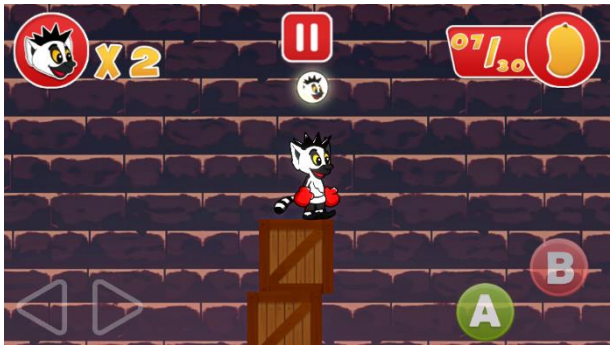
Si presionamos el botón **B**, el diálogo se acabará.

Para **completar el nivel**, se deberá seguir las *flechas indicativas de dirección*, que indican la dirección donde se encuentra el final del nivel:




El objetivo es recuperar los **mangos** que nos han robado, la HUD nos muestra los mangos que hemos recogido y los que hay en total en el nivel, al recoger un Mango, el HUD se actualiza sumándolo:

En ocasiones podremos encontrar **vidas** que nos permitirán seguir jugando al juego:



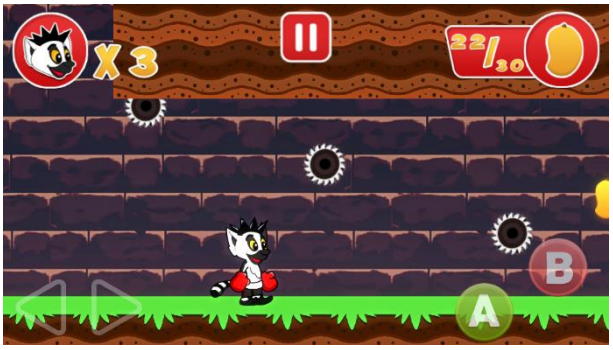

Si nos quedamos sin vidas, aparecerá el **menú de GameOver**:


Si presionamos el botón **Continuar**, continuaremos desde el inicio del nivel recuperando 2 vidas y perdiendo el progreso del nivel.

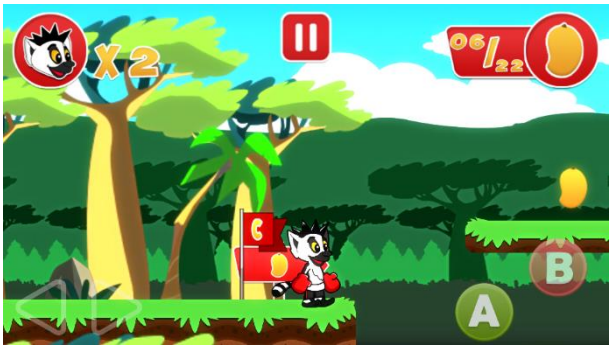
Si presionamos el botón **Salir**, nos llevará a la pantalla de inicio.

Durante el nivel podremos encontrarnos con distintos tipos de **enemigos**, algunos a los que podremos derrotar *golpeándolos* con el botón B, otros *saltando* sobre ellos, y otros que tendremos que *esquivar* sin morir:





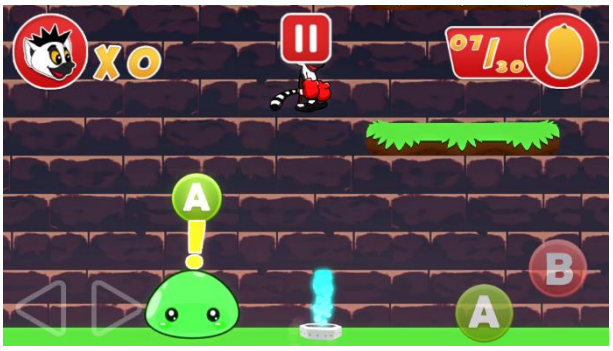

A lo largo de los niveles encontraremos **Checkpoints** que nos permiten renacer en esos puntos en vez de al inicio del nivel:



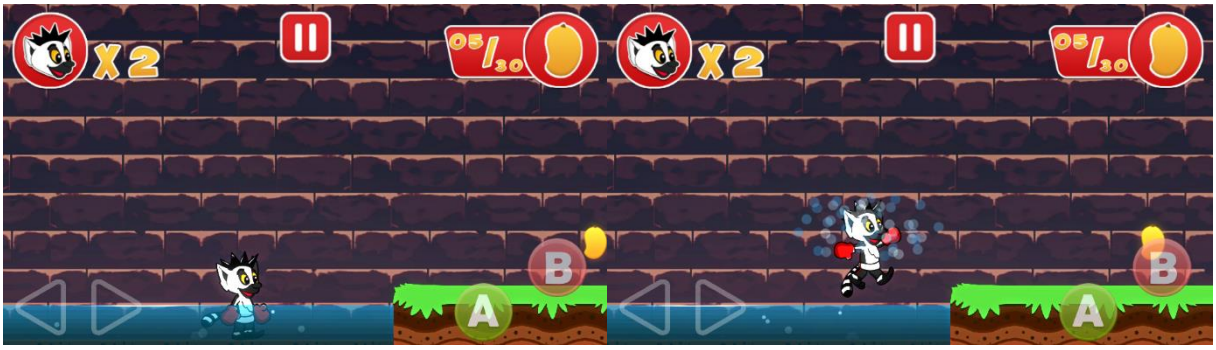
A parte de enemigos y Mangos y Vidas, podemos encontrar objetos como:

1. **Caja destruible**: cajas que pueden obstruir nuestro camino o ayudarnos a llegar a sitios altos. Pueden ser destruidas golpeándolas con el botón B:

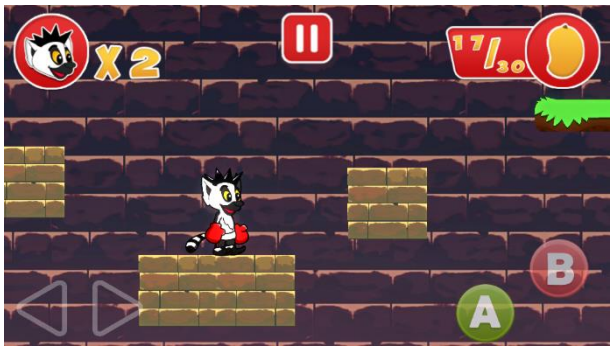
2. **Saltador**: objetos que al situarnos sobre ellos nos elevan por los aires para llegar a sitios inalcanzables:



3. **Agua**: volumen de agua que reduce nuestra velocidad:



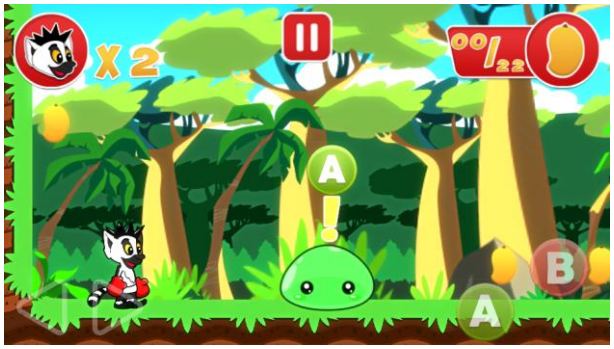
4. **Plataformas destruibles**: plataformas que al situarnos encima de ellas durante un breve tiempo, empiezan a caer al vacío:



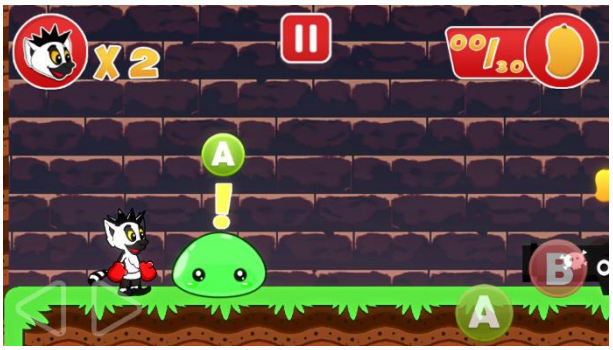

Actualmente el juego dispone de **3 niveles**:

1. **Introducción al juego**: ayuda al jugador a iniciarse con los controles básicos del juego.


2. **Nivel 1**: introduce checkpoints, mangos a recoger y enemigos que derrotar.



3. **Nivel 2**: introduce nuevos enemigos, objetos especiales, plataformas que se mueven y plataformas que se caen.



# Capítulo 11
# Conclusiones

La realización de este proyecto se pensó en un principio en broma con unos amigos, pensando que sería buena idea desarrollar un videojuego de un lémur rockero tipo Crash Bandicoot. Esta idea se pudo llevar a cabo en la asignatura de Plataformas Móviles, pero debido a la necesidad de programar en Java con el IDE Eclipse, se restringía la visión de lo que se quería. Por ello, se pensó que podía ser una muy buena idea para iniciarse en el mundo del desarrollo de videojuegos, el realizarlo como Trabajo de Fin de Grado. Y definitivamente, ha sido una gran decisión, tanto por el interés diario con el que se ha desarrollado, realizando algo que te gusta (excepto cuando un script no compila) como por los conocimientos aprendidos que han sido muchos, tanto sobre Unity3D, como la programación en C# y el diseño de un videojuego completamente desde cero.

La realización de un documento como este también ha sido útil para comprender qué debe realizar un ingeniero del software a la hora de desarrollar un proyecto de esta magnitud, permitiendo aprender los pasos a seguir en futuros proyectos, y poniendo en práctica lo aprendido durante la carrera.

Ha habido varios problemas durante el desarrollo, sobre todo en la fase de Diseño, debido al desconocimiento de cómo empezar a realizar un videojuego en 2D, y sobre todo, cómo hacer que una imagen cobre vida y ande, salte y pueda ser controlado por una persona. Aunque en este aspecto el haber realizado un proyecto en 3D en la asignatura de Informática Gráfica ha ayudado a entender los principios básicos de la animación.

La cantidad de ayuda y documentación ofrecida por Unity3D ha hecho más amena la parte de implementación, y las fases de análisis y planificación debido a lo aprendido en las asignaturas sobre estos temas.

El resultado del proyecto supera las expectativas, y permite ahora seguir desarrollándolo y aumentando sus características y funcionalidades para poder ser ofrecido al público a través de la tienda de Android GooglePlay, además del deseo de poder ser exportado también a la plataforma iOS en un futuro.

## 11.1. Futuras mejoras

En el futuro el videojuego podrá mejorarse de múltiples formas:

- **Modo historia**: se había establecido como sub-objetivo no prioritario el desarrollar una historia con imágenes que cuente la aventura de nuestro protagonista, debido a falta de tiempo no ha podido implementarse, pero en el futuro se quiere conseguir.

- **Guardar/Cargar progreso**: el otro sub-objetivo establecido que no ha podido conseguirse, se quiere implementar en un futuro cercano.

- **Pantalla de selección de nivel**: al igual que en el juego Rayman Jungle Run, tengo la idea de crear una pantalla que sea cargada al pulsar START en la pantalla de inicio, en vez de mostrar una lista de niveles. En esta pantalla habrá distintas puertas indicando distintos tipos de escenario (Bosque, Cueva, Agua, etc..), donde en cada escenario nos encontraremos con niveles con una característica especial, por ejemplo en Bosque serían niveles con árboles y arbustos de fondo, a cielo abierto; en Cueva sería en el subsuelo, sin árboles y con menos luz; en Agua que sean niveles de sólo agua donde tengamos que ir nadando al final del nivel.

- **Bosses**: se ha diseñado ya un Boss (Jefe final) que es el Rey Enemigo, pero no ha sido implementado. Una futura mejora sería incluir un nivel especial al final de cada tipo de escenario, que incluya un Boss al que derrotar para desbloquear el siguiente tipo de escenario.

- **Añadir acciones**: además de las acciones de Saltar y Golpear, se tiene pensado añadir las acciones de Rodar, que permitirá al jugador rodar por el suelo para esquivar cuchillas por ejemplo, o para atravesar partes bajas.

- **Añadir Power-Ups**: una gran mejora será añadir objetos que otorguen distintos tipos de poderes temporales al jugador, como poder lanzar los puños, ser invencible, imán para recoger mangos más fácilmente, etc..

- **Sistema de logros de GooglePlay**: GooglePlay dispone de un sistema de Logros, sería una gran mejora añadir esto al juego para que los jugadores tengan una recompensa por realizar distintas cosas en el juego.

- **Añadir efectos de sonido**: añadir efectos de sonido a los enemigos, zonas de diálogo y otras situaciones.

# Capítulo 12
# Bibliografía

Este capítulo contiene la bibliografía y webgrafía a la que se ha recurrido para realizar el proyecto:


- Unity Technologies. *Unity Manual* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: http://docs.unity3d.com/Manual/

- Nicolás Arroja Landa. *Unity. Diseño y programación de videojuegos*. Buenos Aires: RedUsers, 2013. ISBN 978-987-1857-81-4

- Mario Zechner. *Desarrollo de juegos para Android.* Madrid: Anaya Multimedia, 2011. ISBN 978-844-1530-35-5

- Unity Technologies. *Unity Scripting Reference* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: http://docs.unity3d.com/ScriptReference/index.html

- Unity Technologies. *Unity Tutorials* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: http://unity3d.com/learn/tutorials/modules

- Unity Technologies. *2D Character Controllers* [en línea]. 2015. [Última consulta: 25/06/2015].
  Disponible en: https://unity3d.com/es/learn/tutorials/modules/beginner/2d/2d-controllers

- Miguel Ángel Taramón. *Creación de juegos móviles multiplataforma en Unity*[en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: http://es.slideshare.net/MiguelitoCupra/codemotion-2014-introduccin-a-unity

- WalkerBoys. *Unity lab Project 4 – 2D Mario Clone* [en línea]. 2011. [Última consulta: 25/06/2015]. Disponible en: https://vimeo.com/album/1567704

- Xenosmash Games. *Animation transitions with Mecanim in Unity3D* [en línea]. 2014. [Última consulta: 25/06/2015].
  Disponible en: https://www.youtube.com/watch?v=muoyU8U2yQY Duración: 3:57.

- 3DBuzz. *Modern GUI Development in Unity 4.6.* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/playlist?list=PLt_Y3Hw1v3QTEbh8fQV1DUO-UIh9nF0k6c [Lista de reproducción: 10 vídeos]

- 3DBuzz. *Creating 2D Games in Unity 4.5.* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/playlist?list=PLt_Y3Hw1v3QSFdh-evJbfkxCK_bjUD37n [Lista de reproducción: 38 vídeos]

- Marks Tuto Gamer. *#8 | PAUSE, RESTART, EXIT – Unity3D* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/watch?v=UzS5obb-Js4&index=1&list=PLmnRZOYdMg6mMJ3xT6jjGEkmK138e6yCD [Duración: 26:09]

- Marks Tuto Gamer. *Desnvolvimento de Jogos 2D – Unity3D* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/playlist?list=PLmnR-ZOYdMg6n2irv_9d0fZVArY7U8ATby [Lista de reproducción: 23 vídeos]

- Hagamos Videojuegos. *Creando un juego "Infinite Runner" en 2D con Unity 4.3* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/playlist?list=PLREdURb87ks2qkD9svvlIwYwN35FZ3Afv [Lista de reproducción: 27 vídeos]

- Brackeys. *How to make a 2D Plataformer – Unity Course* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/playlist?list=PLPV2KyIb3jR42oVBU6K2DIL6Y22Ry9J1c [Lista de reproducción: 22 vídeos]

- Brackeys. *How to program in C# - Beginner Course* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/playlist?list=PLPV2KyIb3jR6ZkG8gZwJYSjnXxmfPAl51 [Lista de reproducción: 10 vídeos]

- BrashMonkey. *Getting Started With Spriter* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/watch?v=aQy7eX_CWPM [Duración: 6:56]

- Ana Mocholí. *Desarrollo de juegos con Unity3D ¿Cómo funciona esta herramienta?* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: https://www.yeeply.com/blog/desarrollo-de-juegos-con-unity-3d/

- *bfxr*. *Make sound effects for your games* [en línea]. [Última consulta: 25/06/2015]. Disponible en: http://www.bfxr.net/

- Olga. *Making Sheldon moves* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: http://beesquare.net/making_sheldon_move/

- Juan Ranchal. *La gran oportunidad de la industria del videojuego* [en línea]. 2014. [Última consulta: 25/06/2015].
  Disponible en: http://www.muycanal.com/2014/01/24/oportunidad-de-la-industria-del-videojuego

- Portalic. *La industria del videojuego valdrá más de 100.000 millones de dólares en 2017* [en línea]. Madrid, 2015. [Última consulta: 25/06/2015]. Disponible en: http://www.europapress.es/portaltic/sector/noticia-industria-videojuego-valdra-mas-100000-millones-dolares-2017-20140115164748.html

- Appbackr. *Digi capital mobile internet investment review 2014* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: http://es.slideshare.net/appbackr/digi-capital-mobile-internet-investment-review-q2-2014-summary?related=1

- Christian Brüggemann. *Lep's World 2* [en línea]. 2013. [Última consulta: 25/06/2015]. Disponible en: http://www.androidpit.es/lep-s-world-2

- Gamification Today. *Video games investment review* [en línea]. 2012. [Última consulta: 25/06/2015]. Disponible en: http://es.slideshare.net/GamificationToday/video-games-investment-review?related=1

- Emmanuel Jiménez. *10 razones para elegir Android en lugar de iOS o Windows Phone* [en línea]. 2013. [Última consulta: 25/06/2015]. Disponible en: http://androidayuda.com/2013/07/13/10-razones-para-elegir-android-en-lugar-de-ios-o-windows-phone/

- *Arquitectura Android* [en línea]. [Última consulta: 25/06/2015]. Disponible en: https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android

- Héctor Aguilera. *Plataforma Android en teléfonos móviles* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: https://agdihemi.wordpress.com/

- David Pérez Climent. *Creación de un videojuego en XNA utilzando RT-DESK* [en línea]. 2013. [Última consulta: 25/06/2015].
  Disponible en: https://riunet.upv.es/bitstream/handle/10251/31431/Memoria.pdf?sequence=1

- Jorge Ruiz Magaña. *Conceptualización, análisis, diseño e implementación de un videojuego para Android* [en línea]. [Última consulta: 25/06/2015]. Disponible en: http://e-archivo.uc3m.es/bitstream/handle/10016/18362/TFG_Diego_Trompeta_Urretavizcaya.pdf?sequence=1

- Vicente Bermejo Bermejo. *Simple Android Games* [en línea]. 2013. [Última consulta: 25/06/2015]. Disponible en: http://uvadoc.uva.es/bitstream/10324/4070/1/TFG-B.356.pdf

- Sandra Sastre Muñoz. *Entorno integrado para la simulación de modelos con aplicación a la diagnosis* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: https://uvadoc.uva.es/bitstream/10324/8101/1/TFG-B.585.pdf

- Wikijuegos. *Un ejemplo de la creación de un videojuego* [en línea]. 2009. [Última consulta: 25/06/2015]. Disponible en: http://wikis.uca.es/wikijuegos/w/index.php?title=Temario/Un_ejemplo_de_la_creaci%C3%B3n_de_un_videojuego

- Market Share Stadistics for Internet Technologies. *Market Share Reports* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: http://www.netmarketshare.com/

- G.L.Aris, C.F.Marcos, P.M.Pablo. *Terraform: simulación de vida sobre planetas generados* [en línea]. 2012. [Última consulta: 25/06/2015]. Disponible en: http://eprints.ucm.es/16697/1/Memoria_Terraform.pdf

- S.M.Eduardo. *Desarrollo de un Videojuego para Android* [en línea]. 2013. [Última consulta: 25/06/2015].
  Disponible en: http://deim.urv.cat/~pfc/docs/pfc1311/d1360574623.pdf

- Wikipedia. *Singleton pattern* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://en.wikipedia.org/wiki/Singleton_pattern

- DreamCore90. *Unity 4.6 RawImage UV animator* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: http://forum.unity3d.com/threads/unity-4-6-rawimage-uv-animator.285851/

- Unity Technologies. *2D Game Development Walkthough* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: http://unity3d.com/learn/tutorials/modules/beginner/2d/2d-overview

- Unity Technologies Japan. *2D Shooting game* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: http://japan.unity3d.com/developer/document/tutorial/2d-shooting-game-en/game/

- Adam Tuliper. *Developing Your First Game with Unity and C#* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: https://msdn.microsoft.com/en-us/magazine/dn781360.aspx

- Unity Technologies. *Using the UI Tools* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://unity3d.com/es/learn/tutorials/modules/beginner/live-training-archive/using-the-ui-tools

- Unity Technologies. *Mobile development: converting Space Shooter to Mobile* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: https://unity3d.com/es/learn/tutorials/modules/beginner/live-training-archive/space-shooter-to-mobile

- dcr110. *[Tutorial] Unity Remote install and configuration* [en línea]. 2013. [Última consulta: 25/06/2015]. Disponible en: http://forum.unity3d.com/threads/tutorial-unity-remote-install-and-configuration.187194/

- MikeGeigTV. *Building a 2D Parallax Scrolling Background in Unity* [Youtube]. 2013. [Última consulta: 25/06/2015]. Disponible en: https://www.youtube.com/watch?v=9bhkH7mtFNE Duración: 12:47.

- Carlos Yanez. *Create a 2D Platform Game with Unity and Dolby Audio API* [en línea]. 2014. [Última consulta: 25/06/2015]. Disponible en: http://code.tutsplus.com/tutorials/create-a-2d-platform-game-with-unity-and-the-dolby-audio-api--cms-20863

- UBM Tech. *Gamasutra* [en línea]. 2015. [Última consulta: 25/06/2015]. Disponible en: http://www.gamasutra.com/

- Yury Habets. *[ANDROID] saving data with playerprefs* [en línea]. 2013. [Última consulta: 25/06/2015]. Disponible en: http://answers.unity3d.com/questions/578097/androidsaving-data-with-playerprefs.html

- amd2002a. *How can I pass data between two levels?* [en línea]. 2010. [Última consulta: 25/06/2015]. Disponible en: http://answers.unity3d.com/questions/36738/how-can-i-pass-data-between-two-levels.html

- CodeAndWeb. *What is a sprite sheet* [en línea]. 2014. Disponible en: https://www.codeandweb.com/what-is-a-sprite-sheet

- Gruffi. *Documenting Unity clases/componente/system* [en línea]. 2013. [Última consulta: 25/06/2015]. Disponible en: http://answers.unity3d.com/questions/516482/documenting-unity-classes-components-system.html

- sylvr3. *Sudoku-Game* [en línea]. 2011. [Última consulta: 25/06/2015]. Disponible en: https://github.com/ryanalane/Sudoku-Game/blob/master/documentation/Use%20Case%20Diagram/sudoku_usecasediagram.png

- Scrat. *CountLines* [en línea]. 2012. [Última consulta: 25/06/2015]. Disponible en: http://wiki.unity3d.com/index.php?title=CountLines

- Jay. *XNA Game Development* [en línea]. 2009. [Última consulta: 25/06/2015]. Disponible en: http://xnagamedevelopment.blogspot.com.es/2009/03/use-case-diagram.html

- Miguel A. Martínez. *Gestión de Requisitos*. 2014.

- Miguel A. Martínez. *Elicitación de Requisitos*. 2014.

- Miguel A. Martínez. *Entendiendo los Requisitos de usuario*. 2014.
- Miguel A. Martínez. *Modelado de Análisis*. 2014.
- Miguel A. Martínez. *Documentación de Requisitos*. 2015.
- Aníbal Bregón García. *Arquitectura de las plataformas móviles*. 2014.
- Francisco J. González Cabrera. *Planificación de proyectos*. 2012.
- Francisco J. González Cabrera. *Estimación de costes, esfuerzo y tiempos de un nuevo proyecto software*. 2012.

# Capítulo 13
# Glosario

Este capítulo contiene el vocabulario técnico y argot del documento:

- **Asset**: modelos, texturas, objetos, sonidos, cualquier contenido que compone un juego en Unity3D.
- **Boss**: enemigo final de un juego. Suele disponer de distintas fases y más vida y dificultad que los demás enemigos.
- **Demo**: demostración de las características de un videojuego en vistas a ser publicado, permitiendo la prueba de algunas de las características del juego final.
- **Físicas**: convierten a un objeto en sólido, permitiéndole colisionar con otros objetos, siempre y cuando también dispongan de físicas.
- **Frame-by-frame**: imagen-por-imagen. Secuencia de imágenes que a gran velocidad dan sensación de movimiento.
- **GUI**: Interfaz de Usuario Gráfica.
- **HUD**: Heads Up Display. Información del jugador como las vidas restantes o los mangos recogidos, mostrados en la parte superior de la pantalla.
- **IA**: Inteligencia Artificial.
- **IU**: Interfaz de Usuario.
- **Prefab**: objeto *prefabricado* con unos componentes determinados.
- **Smartphone**: dispositivo inteligente de reducido tamaño con gran capacidad de procesamiento, conexión permanente o intermitente a una red, con memoria limitada y diseñados específicamente para una función, aunque puedan realizar otras más generales.

- **SplashScreen**: pantalla previa al juego que muestra durante unos segundos normalmente el desarrollador de la aplicación, la versión y otra información que se desee dar a conocer antes de cargar la aplicación.

- **Sprite**: imagen representada por una altura y anchura de pixels. Por cada pixel se almacena su color.

- **Spritesheet**: conjunto de sprites utilizados para reducir la memoria utilizada, al reunir en un mismo sprite múltiples sprites.

- **Videojuego AAA**: videojuego que destaca en todos sus apartados, esto quiere decir que tiene calificación A (equivalente a 10) en jugabilidad, gráficos y sonido.

- **Videojuego Arcade**: videojuego diseñado siguiendo los principios básicos de los antiguos juegos para máquinas Arcade. Se caracterizan por tener diseño sencillo y controles fáciles, niveles cortos con mayor dificultad a medida que se avanza en el juego, y con el objetivo principal de obtener la puntuación más alta.

- **Videojuego EndlessRun**: videojuego popular en dispositivos móviles, basado en el continuo movimiento del personaje, no pudiendo controlar su dirección, permitiendo realizar varias acciones como saltar o golpear con el objetivo de llegar al final del nivel recogiendo distintos objetos como monedas.

- **Videojuego Indie**: se denominan indies (independientes) los videojuegos desarrollados por grupos reducidos de personas, con escaso presupuesto. Suelen centrarse en la innovación y sus precios son muy bajos y la mayoría se vende sólo en formato digital.

# Capítulo 14
# Contenido del CD-ROM

Este capítulo especifica lo contenido en el CD-ROM:

- **Título_TFG.pdf**: portada del proyecto.
- **Resumen_TFG.pdf**: resumen del proyecto.
- **Memoria_TFG.pdf**: memoria del proyecto.
- **Diagramas**: diagramas del documento en mayor tamaño.
- **Diseño**: contiene todo el proceso de Diseño del proyecto.
    - **[x][5]3D**: lo realizado en 3D con el software Blender, que no fue implementado.
    - **animations**: proceso de animación del personaje del juego.
    - **bocetos**: arte previo al diseño digital sobre el proyecto.
    - **logo**: proceso de creación del logo del juego.
    - **lookart**: referencias de diseño.
    - **sprites**: proceso de creación de los sprites del juego.
- **Software**: software del proyecto.
    - **CocoPunch**: proyecto Unity3D, contiene lo indicado en la parte de Implementación de este documento.
    - **Documentación**: contiene la documentación del proyecto en formato html generada por Doxygen.
        - **Docs**: archivos html que conforman la documentación.
        - **index.html**: archivo de inicio de la documentación.
        - **Manual de usuario.txt**: manual de usuario de cómo utilizar la documentación.
    - **cocoPunch.apk**: archivo de instalación del proyecto.

[5] [x] significa que no fue implementado finalmente, pero fue realizado en la fase de Diseño.