# Universidad Carlos III de Madrid

Escuela Politécnica Superior


Ingeniería en Telecomunicación

# PROYECTO FIN DE CARRERA

# Diseño e implementación de un juego para smartphones con Android: GyroWorld

Autor: Miguel Ángel Márquez Martínez

Tutor: Moisés Martínez Muñoz

# Resumen

En este trabajo se presenta el diseño e implementación de un videojuego para plataformas móviles en el sistema operativo *Android*. El objetivo principal de este proyecto consiste en aprender a desarrollar este tipo de aplicaciones, resolver las contingencias comunes inherentes a las limitaciones hardware que puede ofrecer un dispositivo móvil y entender el ciclo de vida de un proyecto como éste. Para el desarrollo de este trabajo se ha utilizado el entorno de desarrollo *Unity* (Motor de desarrollo de videojuegos multiplataforma) programando en *C#*, *3DS Max* para el modelado de gráficos 3D y *GIMP* para la creación de texturas. Para el testeo y validación de la aplicación desarrollada se ha utilizado un terminal móvil *BQ Aquaris 5 HD*.

# Abstract

In this document the design and implementation of a video game for mobile platforms on the *Android* operating system is presented. The main objective of this project is to learn how to develop these applications, solving common contingencies pertaining to hardware limitations that can offer a mobile device and understand the life cycle of a project like this. For the development of this work, it has used the *Unity* development environment (Engine game development platform) programming in *C#*, *3DS Max* for modeling 3D graphics and *GIMP* for creating textures. For testing and validation of the developed application it has used a mobile terminal *BQ Aquaris 5 HD*.

# Índice General

# Índice de figuras

# Índice de tablas

# Capítulo 1: Introducción

En este capítulo se realiza una breve introducción del proyecto realizado, la motivación por la cual se ha llevado a cabo, los objetivos iniciales que se definieron y una descripción detallada de la estructura del documento.

## 1.1 Introducción

Un videojuego es una aplicación interactiva orientada al entretenimiento en el que una o más personas interactúan, por medio de un controlador, con un dispositivo que permite simular experiencias en las que intervienen los jugadores a través de un dispositivo visual. Los videojuegos necesitan de un determinado dispositivo electrónico, conocido como plataforma, para poder ser utilizados. Puede ser un ordenador, una máquina *arcade*, una videoconsola o un dispositivo portátil (un teléfono móvil, por ejemplo).

Al dispositivo de entrada usado para manipular un videojuego se lo conoce como controlador de videojuego o mando, y varía dependiendo de la plataforma. Por ejemplo, un controlador podría únicamente consistir de un botón y un *joystick*, mientras otro podría presentar una docena de botones y una o más palancas (*gamepad*). Los primeros juegos solían hacer uso de un teclado para llevar a cabo la interacción, o bien requerían que el usuario utilizara un *joystick* con un botón como mínimo. Muchos juegos de ordenador modernos permiten o exigen que el usuario utilice un teclado y un ratón de forma simultánea. Entre los controladores más típicos están los *gamepads*, *joysticks*, *kinect,* teclados, ratones y pantallas táctiles. Por lo general, los videojuegos hacen uso de otras maneras, aparte de la imagen, de proveer la interactividad e información al jugador. Un claro ejemplo de esto es el audio, usándose dispositivos de reproducción de sonido, tales como altavoces y auriculares; o la vibración, como en los casos de algunos mandos de videoconsolas más modernas y terminales móviles.

El desarrollo de videojuegos desde finales del año 2000 ha sufrido una importante evolución hacia nuevas plataformas de ejecución. Los videojuegos hasta la fecha,

prácticamente estaban limitados a ordenadores y videoconsolas, abriéndose un nuevo mundo para los desarrolladores: los teléfonos móviles de última generación (*smartphones*) y tabletas con pantallas táctiles.

Se prevé que el número de *smartphones* llegará a 2 mil millones para el año 2015. Estos dispositivos tienen unas capacidades gráficas que les permiten ejecutar juegos que son técnicamente comparables a los de mediados de los años 90 para ordenadores de sobremesa. Las tiendas de aplicaciones (*App Store* para *iOS* y *GooglePlay* para *Android*) permiten a los usuarios descargar e instalar fácilmente estos juegos. Los sistemas de pago integrados permiten a un cliente comprar sin problemas juegos completos o hacer compras en el juego (sistema de micro-pagos).

*Unity* [1] es actualmente el motor de juego más utilizado para el desarrollo de videojuegos para plataformas móviles. Es compatible con casi todas las plataformas en las que los juegos se ejecutan en la actualidad: Ordenadores de sobremesa, portátiles, consolas, navegadores web y *smartphones*.

## 1.2 Motivación

Muchas de las personas que de alguna manera han tenido contacto con alguna de las disciplinas necesarias para el desarrollo de videojuegos se han preguntado alguna vez: ¿Cómo se hace un videojuego?, ¿qué tengo que saber?, ¿por dónde empiezo? Y un largo etcétera. Muchos han intentado hacerlo sin resultados concluyentes; otros en cambio han podido crear o participar en sus pequeños juegos con un poco de esfuerzo, pero aun así siempre quedan preguntas no resueltas y mucho por aprender.

Existe gran cantidad de libros y artículos relacionados con el desarrollo de videojuegos. Es un mundo muy amplio en el que cada individuo tiene que tomar su propio camino para iniciarse en él con un objetivo (por pequeño que sea) marcado, e ir aprendiendo todo lo posible para conseguir la meta deseada. Para estar al día con la tecnología que se aplica en el mundo de los videojuegos, hay que ser amplio de miras y renovarse constantemente

para tener todos los elementos necesarios: Es fundamental conocer la evolución de las herramientas de desarrollo para utilizarlas en nuestro beneficio.

Para comenzar en el desarrollo de videojuegos es preferible hacerlo con proyectos pequeños, con el fin de ser capaces de cumplir las metas que nos fijamos y nunca pensar en proyectos extremadamente grandes que se nos escapen de las manos, ya que lo más probable es que nunca los terminemos, lo cual nos llevará a la frustración.

Es común comenzar con remakes de juegos como *Pong*, *Tetris*, *Arkanoid*, *Pacman*, etcétera. Es muy buen ejercicio desarrollar algún juego conocido con modificaciones. Nos hará aprender infinidad de soluciones a problemas que ni llegábamos a imaginar, lo que hará enriquecernos con muchos conocimientos que podremos aplicar en futuros desarrollos. Es deseable también la mezcla de los juegos antes citados con tecnologías que eran impensables para la época. Yo mismo, sin ir más lejos, recientemente he desarrollado el *Pong* para dos jugadores, testeado en teléfonos móviles.

Cada individuo tiene que tener claro cuál es el fin cuando dedica tiempo y esfuerzo en el desarrollo de un videojuego. En el caso de tener como objetivo un desarrollo profesional en el mundo de los videojuegos, es preciso estudiar y entender todos los procesos que se ven involucrados, desde los llevados en una pequeña empresa de desarrollo hasta los que se llevan a cabo en los grandes estudios de la industria.

## 1.3 Objetivos del trabajo

El objetivo principal de este proyecto es conocer y aprender todas las fases del desarrollo de un videojuego de plataformas para dispositivos móviles. Todo ello, a través de la tecnología de desarrollo *Unity*, ampliamente utilizada para el desarrollo de juegos (sobre todo a nivel de desarrolladores independientes y estudios de mediana envergadura). En base a este objetivo principal, se proponen los siguientes objetivos parciales:

- Estudiar el funcionamiento del entorno de desarrollo *Unity*, enfocado particularmente a plataformas móviles.
- Aprender a utilizar el lenguaje de programación C#, orientado al desarrollo de videojuegos.
- Diseñar y documentar, con cierto nivel de abstracción, el conjunto de eventos y situaciones que se darán en el juego a desarrollar.
- Dotar al juego de un nivel mínimo de jugabilidad. Se tiene que poder jugar y que "aporte diversión" al usuario.
- Desarrollar una aplicación que permita añadir nuevas funcionalidades con facilidad. Debe estar modularizado y debe haber una documentación amplia acerca del funcionamiento de cada uno de los métodos y propiedades.

## 1.4 Estructura del documento

A continuación se realiza una descripción detallada del contenido de cada uno de los capítulos y anexos que incluye este documento:

- El capítulo 1, **Introducción**, presenta una introducción al proyecto que se ha realizado, las motivaciones que han llevado a su desarrollo, así como los objetivos iniciales definidos para este proyecto.
- En el capítulo 2, **Estado del Arte**, se presenta un breve resumen del estado actual de la industria de desarrollo de videojuegos. Además se describen las tecnologías que existen actualmente para el desarrollo de videojuegos; se repasará un conjunto de motores de videojuegos actuales, describiéndolos y comparándolos entre sí para determinar cuál es el motor más apropiado para el desarrollo del juego. El capítulo termina con una descripción detallada del funcionamiento de *Unity*, el motor elegido para la realización de este proyecto.
- El capítulo 3, **Descripción del sistema**, describe de forma detallada el análisis y el diseño de la aplicación, así como el proceso de implementación que se ha seguido para llegar a la versión final. En este capítulo se presenta la metodología utilizada

para el desarrollo de la arquitectura de la aplicación, descripción de los distintos módulos con su funcionalidad, entradas y salidas, funcionamiento de las reglas del juego, etcétera.

- El capítulo 4, **Experimentación**, presenta una serie de procesos de evaluación aplicados sobre el videojuego con el fin de analizar su correcto funcionamiento. Mediante estos procesos se han analizado una serie de elementos que indican el correcto funcionamiento de la aplicación y el cumplimiento o no de los objetivos del proyecto. Por lo tanto, se detallará en qué han consistido las pruebas y la serie de resultados que han sido obtenidos.
- En el capítulo 5, **Gestión del proyecto**, se describe la información referente a la planificación de las distintas fases del desarrollo de este proyecto, los medios empleados y los costes totales derivados.
- En el capítulo 6 se presentan las **Conclusiones y trabajos futuros**. En primer lugar se muestran las conclusiones generales obtenidas tras la realización del proyecto. A continuación se presentan las conclusiones obtenidas para cada uno de los objetivos descritos en el capítulo 1. Finalmente se describen los posibles trabajos futuros que podrían realizarse sobre este proyecto.
- Para finalizar, en el capítulo 7 se incluyen un conjunto de **Anexos**, así como toda información que no se ha podido añadir en capítulos anteriores, tales como manuales de instalación y de uso.

# Capítulo 2: Estado del Arte

A continuación se presenta un repaso al contexto del mundo de los videojuegos, poniendo énfasis en la parte del desarrollo; de esta manera será más fácil entender la situación de la que se parte en la actualidad para abordar este tipo de proyectos. Además se comparan algunos de los motores que se utilizan en la actualidad y finalmente se profundizará en el motor de desarrollo *Unity*.

## 2.1 Desarrollo de videojuegos

Un videojuego es una aplicación interactiva con el propósito principal de entretener al usuario (jugador). Inicialmente los videojuegos fueron desarrollados por informáticos principalmente para entretenerse a sí mismos y a sus colegas. A finales de 1970, con un inmenso aumento en la potencia de cálculo, capacidades gráficas y la producción masiva de ordenadores de sobremesa, el desarrollo de juegos se convirtió en una industria. La competencia entre los desarrolladores y el rápido aumento de las expectativas de los jugadores obligó a los desarrolladores a hacer juegos computacionalmente complejos y tan atractivos gráficamente como fue permitido por el *hardware* medio de los usuarios finales.

La década de 1990 vio una gran mejora en la presentación de los juegos: De una paleta de tan sólo 4-8 colores y personajes “pixelados” a comienzos de dicha década, se pasó hasta las obras maestras de 32 bits de color de gráficos 2D a finales de la misma. Una transición de gráficos 2D a 3D siguió rápidamente a esta evolución tecnológica en videojuegos. Hoy día un juego comercial puede tardar hasta 2-3 años en tiempo de desarrollo y tener un volumen de 20 a 100 desarrolladores (salvo algunos casos que pueden llegar a exceder sobradamente el centenar de trabajadores). A finales de 1990, con la adopción en todo el mundo de los teléfonos móviles, una parte del esfuerzo de desarrollo de las empresas de videojuegos consiguió hacer pequeños juegos 2D de bajo presupuesto para estos nuevos dispositivos. Estos juegos no eran realmente exitosos debido a las limitadas posibilidades de distribución, la insuficiente aceptación de los usuarios respecto a los medios de pago,

la extrema fragmentación de *hardware* y el bajo rendimiento que los teléfonos móviles usaban para videojuegos.

En 2007 un nuevo dispositivo llamado *iPhone* de *Apple Inc.* se convirtió en el nicho principal para videojuegos en móviles, marcándose un espacio muy rentable tanto para pequeños desarrolladores independientes como para las grandes empresas. *Apple* rápidamente instauró un solo canal de distribución (*AppStore*), donde el usuario podía acceder directa y fácilmente desde el dispositivo; se convirtió en una plataforma de desarrollo normalizada estable, ofrecida a un público que cada vez más rápidamente estaba dispuesto a pagar por las aplicaciones (y por lo tanto los juegos) que se ofrecían.

En *iPhone* se ejecuta el sistema operativo *iOS*, que se basa en el sistema operativo *Macintosh* de *Apple* para los ordenadores portátiles y de sobremesa. Un nuevo término, *smartphone* (teléfono inteligente), fue acuñado para describir a los teléfonos móviles con una gran pantalla sensible al tacto de aproximadamente 5 pulgadas y más potentes respecto a las capacidades gráficas y de cálculo que un teléfono móvil normal.

En 2009 se lanzó un sistema operativo llamado *Android*, adquirido previamente y mejorado por *Google Inc.*, el cual en 2011 superó en popularidad a *iOS* debido a la liberación de código y a tener menos restricciones en cuanto al desarrollo. El número de juegos desarrollados para *smartphones* creció a un ritmo sin precedentes durante la etapa desde 2009 a 2012 como se refleja en la Figura 1 [2]. Como podemos observar, a partir del año 2009 el número de títulos anuales sacados al mercado se disparó, sobre todo debido a la cantidad de demanda de los usuarios con dispositivos móviles tales como *iPhone* y *iPad.* Como conclusión, actualmente los videojuegos para móviles suponen el crecimiento más notable del mercado de los videojuegos, mientras que el resto de mercados (consolas, juegos de navegador Web, juegos para los dispositivos portátiles) o bien se han estancado o están disminuyendo.

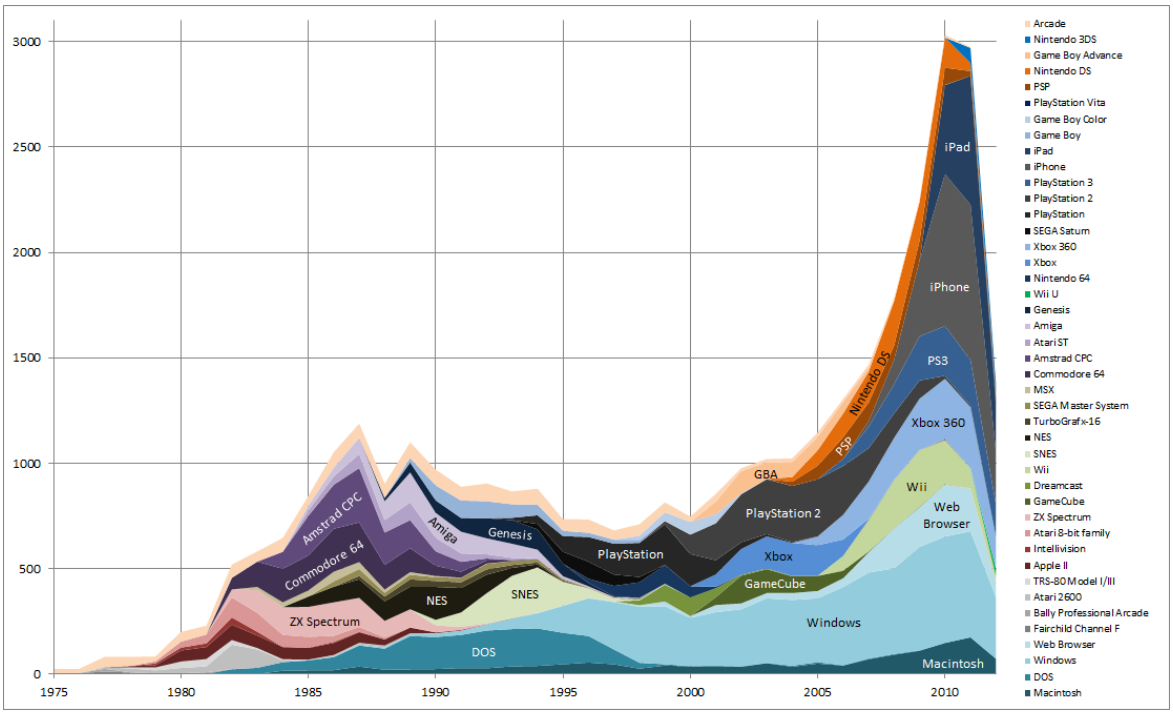


**Figura 1. Evolución de la cantidad de títulos según plataformas de juego**

El desarrollo de un videojuego requiere la colaboración de los desarrolladores de muchas disciplinas, incluyendo la programación, efectos de arte, música y sonido en 2D y 3D, diseño de juego y pruebas. Las tareas de programación durante el desarrollo de videojuegos se dividen en unas pocas áreas, incluyendo Inteligencia Artificial (IA), jugabilidad, gráficos en 3D, redes, herramientas y físicas.

Como ya se ha mencionado, inicialmente los videojuegos fueron diseñados por los propios programadores: Se pensaba simultáneamente en la idea del juego y su aplicación. Este sigue siendo el caso de los juegos "*Indie*" (Independientes) y juegos realizados por equipos muy pequeños (2 o 3 desarrolladores). Los diseñadores escriben un documento de diseño del juego que en el sector de los videojuegos actúa como un documento de especificación funcional.

Este documento contiene toda la información referente a la estructura del videojuego. En él se especifican los niveles con monstruos, modificar las reglas de juego, establecer los parámetros de los elementos en cada escena y escribir textos en el juego (por ejemplo, las conversaciones de los personajes, tutoriales, tareas de misión) son algunas de las

responsabilidades de esta disciplina. Con contenidos de juego también nos referimos a las imágenes, sonidos, textos del juego, modelos 3D, secuencias de comandos y otros activos que son específicos al juego, los cuales no son generalmente desarrollados por programadores.

## 2.2 Técnicas de implementación comunes de videojuegos

A continuación se presentan una serie de conceptos y técnicas básicas que hay que tener en cuenta para el desarrollo de videojuegos.

### 2.2.1 Motor de juego

Un motor de juego es un *middleware* (un paquete de software o una biblioteca grande y compleja) que implementa la funcionalidad común utilizada para los juegos. Los motores de juego comerciales potentes suelen ser multi-plataforma: Se ocultan del sistema operativo subyacente, y se le proporciona al desarrollador de videojuegos una interfaz de programación de aplicaciones (API, *Application Programming Interface*), que es más adecuada para el desarrollo del juego.

### 2.2.2 Bucle de juego

Una aplicación de juego por lo general tiene un bucle principal que se repite siempre que el juego no ha terminado. El bucle principal de juego generalmente se encuentra en el interior del motor de juego. El bucle del juego se presenta en la Figura 2. Este se encuentra comúnmente formado por los diferentes pasos que se muestran a continuación, los cuales se ejecutan en cada actualización:

1. Entrada del usuario: Este proceso consiste en leer la entrada del usuario.
2. Simulación de mundo de juego: Este segundo proceso actualiza la simulación del mundo de juego de acuerdo con la entrada del usuario, actualización de IA y físicas en el juego.
3. Renderizado: Este último proceso renderiza el mundo de juego con la simulación actualizada.

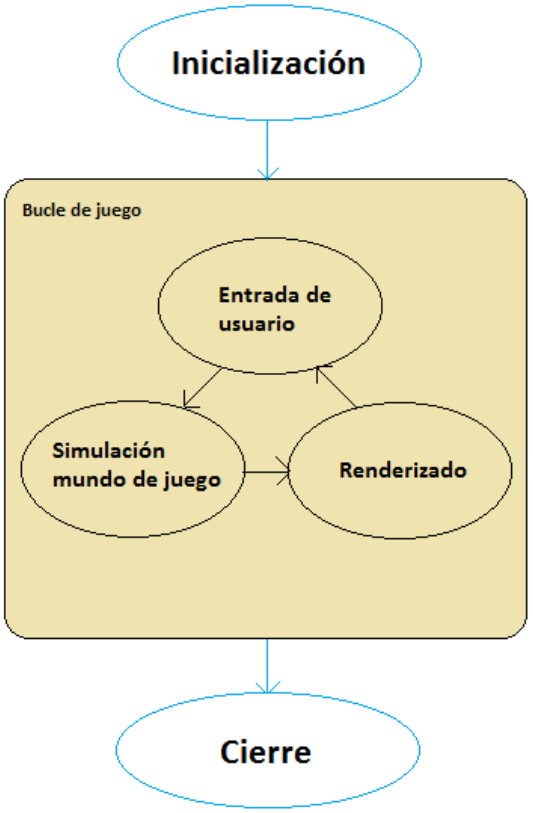


Figura 2. Bucle de juego

Cuando el usuario elige salir del juego, el bucle termina y, finalmente, el método principal devuelve el control al sistema operativo.

### 2.2.3 Máquinas de estados finitos

Las máquinas de estados finitos (FSM, *Finite-State Machine*) se definen como un conjunto de estados que sirven de intermediarios entre entradas y salidas, haciendo que el historial de señales de entrada determine, para cada instante, un estado para la máquina, de

forma tal que la salida depende únicamente del estado y las entradas actuales. Un ejemplo sencillo es el caso de un ascensor, como vemos en la Figura 3. Al recibir una señal de entrada (una persona llama al ascensor desde otra planta) en el estado actual del ascensor (es decir, el piso en el que se encuentra y si está en movimiento o no) se evalúa si cumple una serie de condiciones para pasar a otro estado (ir al piso donde ha sido llamado).


**Figura 3. Máquina de estado finito para un ascensor**

Extrapolando lo explicado al proyecto en cuestión, los comportamientos semi-automáticos de los monstruos, enemigos, unidades y otras entidades son a menudo implementados estas máquinas de estados finitos. Dependiendo del estado actual, la entidad se comportaría de alguna manera específica a ese estado. Dichas técnicas son fáciles de entender por los jugadores y fáciles de utilizar por los diseñadores. En algunos juegos, el jugador puede ordenar a las entidades de que cambien a un estado determinado. Por ejemplo, en *World of Warcraft* un cazador puede ordenar a su perro permanecer en modo defensivo y sólo atacar a los enemigos que atacan al cazador.

### 2.2.4 Scripting

Los *scripts* son ficheros de código en lenguajes interpretables generalmente con sintaxis simple como *Lua*, *Python*, *JavaScript*, o *C#* (existen muchos más), muy utilizados para el desarrollo de aquellas funcionalidades que están relacionadas con la programación de lógica en la aplicación. Por ejemplo, imaginemos que tenemos la necesidad de implementar un sistema que detecte el contacto con el personaje principal y se notifique al gestor del flujo de juego que el personaje ha muerto (típico funcionamiento en pinchos, lava, etcétera). Mediante *Scripting* podemos hacer que si se reconoce el contacto con el personaje, acto seguido se avisa a dicho gestor como se puede ver en el *script DeadTrigger.cs* mostrado en la Figura 4.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class DeadTrigger : MonoBehaviour {
5
6      #region Private Members
7      // Encargado del flujo general del juego
8      private GamePlayScript m_GameplayScript;
9      #endregion
10
11     #region MonoBehaviour
12     void Start ()
13     {
14         // Obtenemos la referencia
15         m_GameplayScript = GamePlayScript.GetSingleton();
16     }
17
18     void OnTriggerEnter(Collider other)
19     {
20         /* Si el personaje entra en el trigger, notificamos a GameplayScript
21          * que se ha cumplido la condición de muerte */
22
23         if(other.tag == "Player")
24         {
25             m_GameplayScript.Dead();
26         }
27     }
28     #endregion
29 }
```

**Figura 4. Script DeadTrigger.cs**

Hay que destacar el hecho de que la compilación de un programa puede durar horas, por lo que los desarrolladores intentan minimizar la necesidad de compilar. Para satisfacer esta necesidad, los programadores intentan extraer toda la lógica y los datos que potencialmente pueden cambiar durante el desarrollo mediante archivos separados (*scripts*) que no tienen que compilar, e incluso idealmente no requieren que el juego se vuelva a "lanzar" para observar los cambios realizados. En videojuegos, los programadores deben separar el código común a todos los juegos respecto del código específico para el juego que se desarrolla en un momento concreto. Se debe mantener el código genérico en el motor de juego, mientras que el código para proyectos concretos se produce en forma de *scripts*.

## 2.3 Tecnologías para el desarrollo

En este apartado se describe el concepto de motor de juego, en qué consiste y se describen las características principales de los motores de juego más accesibles.

### 2.3.1 Motor de juego

En general, el concepto de motor de juego es muy sencillo de entender. Se trata de una plataforma que aúna las tareas relacionadas con la representación, la física, los cálculos computacionales e interpretación de entradas. Los motores son en realidad una colección de componentes reutilizables que se pueden manipular para llevar un juego al final de su desarrollo (se puede asumir que es un motor de motores). En la Figura 5 se muestra la relación entre el motor de juego y algunos de los motores típicos que cubren la gran mayoría de la funcionalidad en un videojuego tipo. Por lo tanto se consideran justamente a los motores del juego como soluciones *middleware*.


Figura 5. Motor de juego

Existen claras diferencias entre un motor de juego y un juego en sí mismo. Los personajes, los enemigos, los terrenos, la razón detrás de la colisión, las conductas requeridas en el mundo de juego, etcétera, son los elementos que junto al motor, hacen que sea un juego.

### 2.3.2 Comparativa de motores de videojuegos para el desarrollo

En este apartado se describen un conjunto de motores de videojuegos enumerando sus características más relevantes desde la perspectiva del desarrollador amateur o independiente.

#### *2.3.2.1 Unity*

*Unity* empezó como una herramienta de producción para los desarrolladores de videojuegos exclusivamente en *Mac*. Aunque era una herramienta diseñada específicamente para la plataforma de Apple. Cualquier proyecto debía ser desarrollado en un entorno Apple, pero podía ser desplegado en otro entorno eligiendo el sistema

operativo para el cual se quería desarrollar el ejecutable. Las imágenes que era capaz de producir eran aceptables, pero su punto fuerte era el entorno de desarrollo: Un entorno consistente y fácil de manejar. En la Figura 6 se muestra una vista del entorno de desarrollo de Unity.

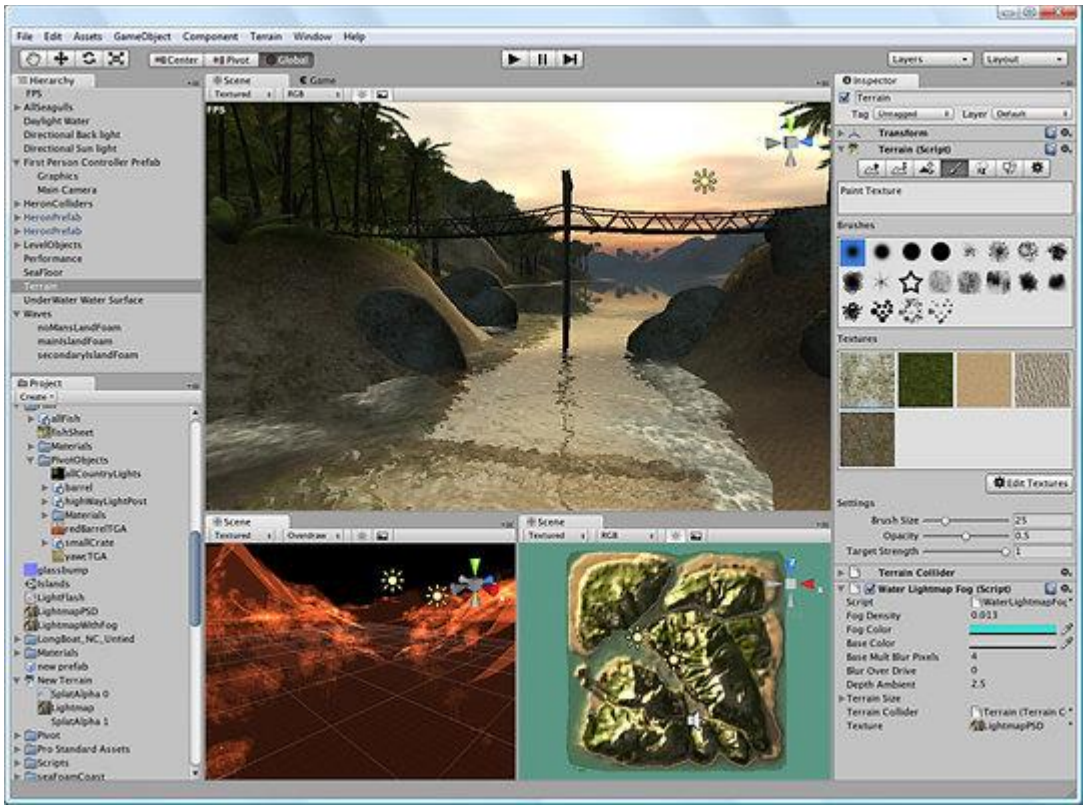

Figura 6. Entorno de desarrollo de Unity

Características:

- *Unity* actualmente va por su cuarta versión. El núcleo del sistema de procesador aún se basa en *OpenGL* [3]. Una de las mejoras más importantes que se han desarrollado en los últimos años para ser cualitativamente competitivo ha sido principalmente mejoras en el rendimiento.
- Utiliza *Umbra* [4], una de las mejores herramientas de la industria para la selección de oclusión. Con una configuración de escenario adecuada a tal efecto, el motor es capaz de evitar renderizados innecesarios, haciendo hincapié en aquellos objetos

de la escena que en un momento determinado se ven ocluidos visualmente por otros.

- Otra de las innovaciones importantes introducidas en *Unity* es el "cocinado" de iluminación mediante el sistema *Beast* [5]. La iluminación *Beast* se hizo famosa cuando fue utilizada por primera vez con gran éxito en el título *Mirror´s Edge*. Este sistema se basa en el "quemado" de un conjunto de iluminaciones en texturas para modificar éstas, así luego se pueden suprimir las luces dinámicas con este pre-procesado y las texturas parecen iluminadas en el juego.
- Otro de los puntos fuertes de *Unity* es que está disponible gratuitamente para cualquier persona que quiera usarlo, incluso para proyectos comerciales y tiene casi todas las características del motor, salvo efectos de post-procesado, sombras en tiempo real y renderizado de vídeos. Estas son características importantes a tener en cuenta, pero aun así se pueden hacer grandes cosas con el conjunto de funcionalidades que tenemos en la versión gratuita.
- *Unity* tiene compatibilidad con la red. No hay muchos lujos desde el punto de vista del desarrollador, pero eso no es un gran problema: La mayoría de los motores de juegos no vienen con capacidades de red extremadamente potentes.
- Para programar en *Unity*, podemos utilizar *JavaScript*, *C#* o *Boo (*este último es una evolución de *Python)*, y como no se tiene la obligación de utilizar un editor determinado, podemos utilizar el que queramos para desarrollar nuestros juegos.
- Respecto a la documentación, no es la mejor que existe, pero es mejor que la documentación de muchos otros motores independientes que hay. Eso sí, las comunidades de foros son bastante buenas pues la gente, sin ánimo de lucro, contestan con buena voluntad a las preguntas que hagamos en cada momento.
- La física viene en la forma que nos provee *PhysX* [6], el motor estándar de físicas de la industria.
- También añadieron a *Unity* un editor de animaciones cronológico, que es muy útil, y recuerda mucho al editor de animaciones del *Unreal* [7] (del cual hablaremos más adelante).

- En la Figura 7 se presenta una imagen del resultado final de la demostración tecnológica desarrollada por *Unity*.

**Figura 7. Demo tecnológica Bootcamp (Unity)**

Conclusiones

En realidad, por sus características y por su bajo precio para desarrollo profesional, no se puede decir que tenga un punto débil de peso para no pensar en este motor como una apuesta segura. El motor ha madurado mucho y claramente apunta a la dirección correcta.

#### *2.3.2.2 ShiVa 3D*

Es un motor de videojuegos *indie* muy interesante, producido por la empresa francesa *Stonetrip* [8].

Características

- Es similar a *Unity* en funciones, herramientas y precios. Lo negativo es que actualmente no hay ninguna versión gratuita.

- El entorno de desarrollo (el cual vemos en la Figura 8) es fácil de usar, una vez te acostumbras a su diseño algo extravagante. No obstante tiene todas las funciones que un motor de videojuegos de este calibre puede desear.

**Figura 8. Entorno de desarrollo de Shiva 3D**

- Tiene una buena iluminación dinámica, pero no hay soluciones de pre-procesado para la iluminación de niveles todavía.
- Tiene físicas a través de *ODE* (*Open Dynamics Engine*, motor de físicas de código abierto) y se codifica en *Lua*. Admite la conversión a *C++*.
- El apoyo de la comunidad de *Shiva* es muy pequeño lamentablemente; eso es un punto importante a tener en cuenta.
- *Shiv*a también se ejecuta en los navegadores y los teléfonos móviles, pero todavía no se ha visto en demostraciones tecnológicas lo que es capaz de ofrecer para estas plataformas. En la Figura 9 se muestra el aspecto visual de una demo tecnológica para PC.


Figura 9. The Hunt, demo tecnológica de Shiva 3D

- El motor tiene un sistema bastante bueno de oclusión por sectores.
- Respecto a la IA, permite el dimensionado de mallas de navegación y *pathfinding*.
- También tiene un editor de *shaders* incorporado.

Conclusiones

Es un motor de juego muy interesante, no obstante está por debajo de otros motores que existen en el mercado en la actualidad; en la demo tecnológica se puede ver que las capacidades del motor están muy por debajo de las de otros motores.

#### *2.3.2.3 Unreal Engine*

*Unreal Engine* es uno de los motores comerciales más utilizados del mundo. Existen una gran cantidad de videojuegos de última generación muy exitosos desarrollados en este

motor, y debido a su alta calidad y a su bajo precio hacen de él una opción muy buena para el desarrollo tanto *amateur* como profesional.

Características

- *Unreal* soporta renderizado avanzado con *DirectX11*.
- Tiene un sistema muy optimizado de materiales basados en simulación de propiedades físicas.
- Tiene un editor de *shaders* basado en nodos muy bueno y eficiente que hace que, en general, permita de base un aspecto visual mejor que otros motores del mercado. Además, tiene una serie de elementos de post-procesado visual muy buenos y bien optimizados. Como se puede ver en la Figura 10, el acabado de los videojuegos hechos en *Unreal* puede ser espectacular.


**Figura 10. Mass Effect 3, juego realizado en Unreal Engine**

- Tiene un sistema nuevo de prototipado bastante aceptable que permite rápidamente montar un entorno para esquematizar lo que queremos de nuestro juego sin escribir una línea de código.

- Se tiene acceso al código fuente del motor. Esto permite tener un control total sobre la gestión de todos los recursos del motor, aunque si no se ha profundizado mucho en la arquitectura del mismo, se puede tardar mucho tiempo en poder mejorar o adaptar alguno de los subsistemas de *Unreal*.
- Se programa en *C++*.
- Comparte muchas de las características de *Unity*, como integración con *PhysX*, *Umbra*, *Oculus VR*, etcétera.
- Un desarrollador independiente (exceptuando algunos casos) tiene que pagar una cuota mensual de 19 $, precio más que asequible considerando la potencia y el conjunto de optimizaciones y herramientas que se proporciona con el motor.

Conclusiones

Este motor de juego es una herramienta tanto potente como asequible; como a se ha dicho es un motor muy recomendado tanto para el desarrollo *amateur* como profesional. Quizás la única pega comparando con *Unity* que se le puede poner a este motor es el tiempo: *Unity* a corto plazo es más fácil de manejar a muchos niveles (entre ellos la programación) ya que *Unreal* tiene una cantidad enorme de posibilidades y subsistemas que hacen que nos lleve más tiempo aprender a manejarlos.

#### *2.3.2.4 CryEngine*

*CryEngine* es un motor gráfico creado por la empresa alemana *Crytek*. Nace como motor de demostración para la empresa *Nvidia* pero al mostrar su gran potencial se implementa por primera vez en el videojuego *Far Cry*, desarrollado por la misma empresa creadora del motor.

Características

- Se usa el famoso editor *SandBox 3* [9], que da a los desarrolladores un control total sobre sus creaciones multiplataforma en tiempo real. Este editor permite separar en capas los distintos niveles de juego y da la posibilidad de que trabajen varios desarrolladores en una misma capa sin la preocupación de un impacto en lo que haga el compañero.
- *Sandbox 3* permite a los diseñadores controlar los comportamientos de la *IA* poniendo más control de la misma en sus manos. Cuenta con un sistema automatizado de generación de mallas de navegación. En la Figura 11 se muestra una secuencia de uno de los mejores juegos hechos en este motor: *Ryse: Sons of Rome.* Uno de los puntos fuertes es el nivel de realismo de la *IA* respecto al personaje principal y al entorno.

**Figura 11. Ryse: Sons of Rome. Juego realizado en CryEngine 3**

- En el aspecto gráfico, *CryEngine 3* tiene como estandarte uno de los renderizados más rápidos de la industria, con nuevas características diseñadas especialmente para consolas.

- Tiene un sistema de edición visual que da a los desarrolladores una intuitiva interfaz para crear y controlar los acontecimientos, factores desencadenantes y otras lógicas del juego.
- En cuanto al diseño de niveles, la vegetación se rige por las reglas naturales con parámetros como la altura y la densidad. El sistema genera automáticamente una vegetación realista y en tiempo de ejecución.
- También se simplifica la creación de fuego, humo y explosiones utilizando la nueva generación de partículas suaves.
- Permite crear fácilmente todo tipo de vehículos con un control intuitivo sobre sus características como pueden ser los daños o las posiciones de los pasajeros y sus funciones.
- *CryEngine 3* está preparado para trabajar con varios núcleos, repartiendo así el trabajo y mejorando el rendimiento.
- Ofrece también una nueva dinámica en tiempo real: La solución de iluminación global, totalmente optimizada para las plataformas actuales y de próxima generación. Los bancos de niebla, nubes, gases de efecto visible y demás son presentados de forma más que realista.
- Tiene un editor facial que utiliza el análisis del audio para extraer fonemas y así animar los rasgos faciales y proporcionar un movimiento de labios convincente.
- El 3D del agua es otro de los puntos fuertes de este motor. Puede ser modificado por la acción del viento u otros agentes externos y genera automáticamente tonos más suaves en las costas o bordes al variar la profundidad del agua.
- Permite *streaming* para carga de niveles dinámica en memoria. Los datos son agrupados para acceso rápido y se comprimen.
- El paquete de física incluido se puede aplicar a casi todo, desde edificios hasta el agua pasando por fuego, etcétera.
- Lo que quizás llama la atención de este motor es la destructibilidad de la mayor parte de la naturaleza. Una alta interactividad con el entorno nos permite destruir la vegetación, madera, acero e incluso hormigón.

- Como lenguajes de programación, *CryEngine* usa para el código fuente *C++*, y *LUA* como lenguaje de *scripting*.
- Recientemente *Crytek* ha sacado una oferta para competir económicamente con otros motores como *Unreal* y *Unity*, en la cual un desarrollador pagando una cuota de 9.90 $ al mes, tiene acceso a todas las funcionalidades del motor (también el código fuente).

Conclusiones

A pesar de ser uno de los mejores motores de creación de videojuegos de la industria, la comunidad de desarrollo es menor que en los casos de *Unreal* o *Unity*, siendo una opción menos favorable simplemente porque el número de empresas que usan dicho motor es menor.

### 2.3.3 Conclusión de comparativa entre los motores más populares

Para poder elegir el motor que mejor se adapta a este proyecto se han seleccionado una serie de características de cada uno de los motores. En la Tabla 1 se presenta una comparativa con sus aspectos más relevantes desde el punto de vista del desarrollo de este proyecto.

| | Capacidad técnica del motor | Desarrollo sencillo para plataformas móviles | Nivel de facilidad de aprendizaje | Comunidad de desarrollo | Proyección de futuro | Precio |
|---|---|---|---|---|---|---|
| **Unity** | Alta | Sí | Alto | Muy buena | Alta | Bajo |
| **Shiva** | Media | No | Medio | Aceptable | Media | Bajo |
| **Unreal** | Alta | Sí | Medio | Muy buena | Alta | Medio |
| **CryEngine** | Alta | No | Medio | Buena | Media | Medio |

**Tabla 1. Comparativa de los motores de videojuegos más relevantes**

*Unity* es el único motor en la industria comparable (en cuanto a capacidades) a los motores *Unreal* y *CryEngine*, aunque su coste es menor que los mencionados, lo que hace que sea popular entre los pequeños estudios y desarrolladores independientes. Tiene una de las comunidades de desarrollo más activas, además de ser el motor más "fácil" de aprender a usar, incluida la parte de desarrollo para plataformas móviles. Además, cada año *Unity* se afianza como uno de los motores de desarrollo de videojuegos más usados a nivel mundial, lo que hace que cualquier desarrollador se interese por aprender a utilizar este motor. Existen muchos más motores de videojuegos más o menos populares y accesibles, no obstante tras repasar algunos de los más importantes y por lo explicado hasta ahora, se ha tomado como motor para el desarrollo de videojuegos *Unity*.

### 2.3.4 Motor de juego *Unity*

A continuación se realiza una descripción detallada del motor de juego que ha sido seleccionado para el desarrollo de este proyecto.

#### *2.3.4.1 Introducción*

Desde el año 2012, *Unity* (o *Unity3D*) es el motor de videojuegos más popular para el desarrollo 3D multiplataforma, especialmente en el caso de desarrollo de juegos para plataformas móviles que soporten *iOS* y *Android*. Esta herramienta se creó fundamentalmente para el desarrollo de videojuegos, no obstante debido a que es una herramienta multiplataforma y a su potente conjunto de motores para desarrollo de entornos 3D, es muy usado en otros ámbitos tales como la animación, simulación, realidad aumentada, desarrollo de aplicaciones didácticas, etcétera.

*Unity* puede ser utilizado por programadores, artistas y/o diseñadores de videojuegos. Cuenta con un editor de juego (Figura 12) con una interfaz que se asemeja a muchos de los programas de modelado 3D (*3DS MAX*, *Maya* o *Blender*). Un desarrollador de juegos

especializado en arte puede llenar la escena de forma sencilla con objetos 3D (objetos del juego), los materiales que se les asigna, colocar cámaras y luces.

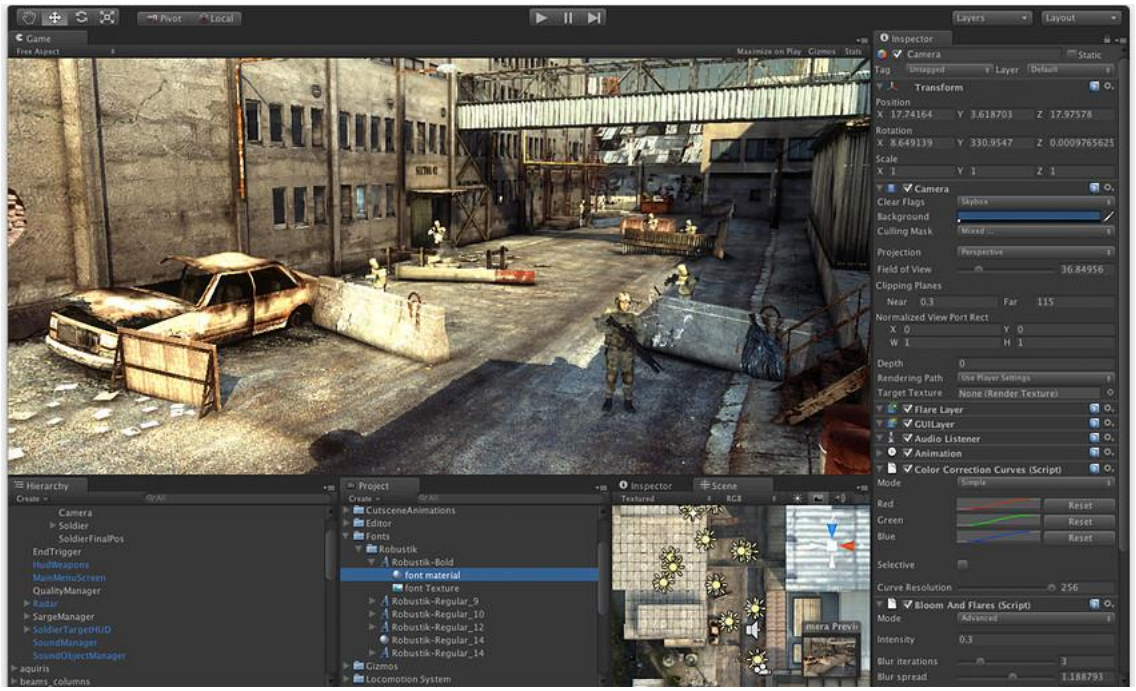


**Figura 12. Visión de la interfaz del editor de Unity**

Para definir comportamientos de los objetos del juego, un programador crea *scripts* y los asigna a los objetos del juego. Para cada nivel del juego, el desarrollador crea una escena de juego independiente. Cabe destacar que la característica principal de esta tecnología es la facultad de implementar rápida e intuitivamente gran cantidad de funcionalidades para entornos 3D sin necesidad de un alto nivel de programación.

### *2.3.4.2 Proyecto en Unity*

Un proyecto en Unity en el conjunto de recursos y carpetas que se necesitan para poder ejecutar y construir un videojuego en *Unity*.

El proyecto de *Unity* se estructura en el disco de la siguiente forma:

- *Assets:* Contiene todos los recursos (gráficos, sonidos, *scripts*, etc.) del proyecto. El concepto de *Asset* se describirá más adelante.
- *Library:* Contiene todos los datos relacionados con la configuración del proyecto, además de los datos auto-generados por *Unity* para la caché de imágenes, metadatos y otros ficheros utilizados por el editor.

Para poder replicar el proyecto en otra parte, basta con copiar las carpetas (manteniendo la estructura) de todo el proyecto en la localización destino que se desee. Una vez esté el proyecto localizado en el lugar deseado, basta ejecutar Unity y abrir el proyecto en la ruta en la que se haya puesto.

#### *2.3.4.3 El interfaz*

Una de las ventajas de *Unity* es que integra casi todos los elementos de la creación de un juego en una sola aplicación. La interfaz del editor, mostrada en la Figura 13, consta de 5 ventanas principales, que explicaremos a continuación.


**Figura 13. Secciones de la interfaz del editor de Unity**

### 1) *Scene*: Editor de escenas

Desde esta herramienta (Figura 14) es posible editar todos los objetos presentes en la escena del juego. Podemos posicionarlos, rotarlos, escalarlos, de forma que se puede crear los escenarios de una forma visual y muy rápida.


**Figura 14. Visión de *Scene:* Editor de escenas**

Podemos modificar la perspectiva con la que vemos la escena utilizando para ello el *Scene Gizmo* (Figura 15), el cual se sitúa en la esquina superior derecha de la vista *Scene* (Figura 14). Podemos hacer clic en cualquiera de sus brazos para ver la escena desde esa perspectiva.


**Figura 15. Scene Guizmo**

También disponemos de herramientas visuales de posición, rotación y escalado que nos permite trabajar de forma intuitiva con los elementos de la escena (Figura 16):

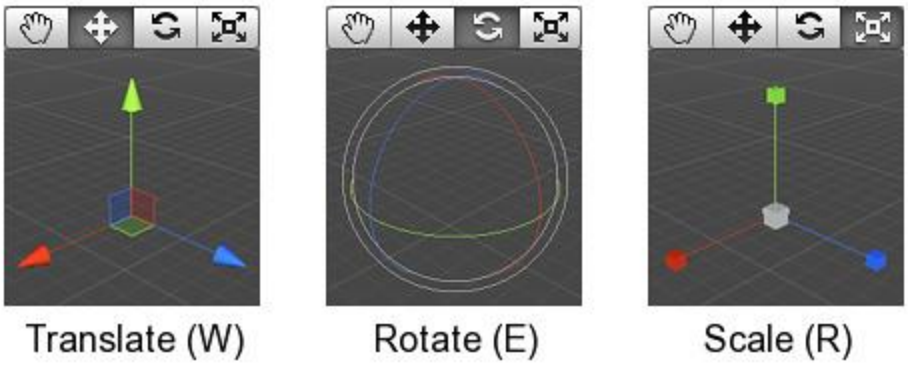


Figura 16. Posición, Rotación y escalado de objetos

**2) *Game*: Visualización del juego**

En esta pestaña se muestra un acercamiento del acabado final del videojuego en ejecución. Básicamente Unity te permite simular la ejecución del videojuego, de forma que se pueden probar todas las funcionalidades desarrolladas hasta el momento.

**3) *Hierarchy*: Ventana de Jerarquías**

En la pestaña *Hierarchy,* se presenta una lista con todos los elementos que han sido cargados actualmente en la escena. Como vemos en la Figura 17, esta lista se refresca dinámicamente durante el juego por lo que si creamos elementos nuevos durante el mismo, se verán aquí reflejados.

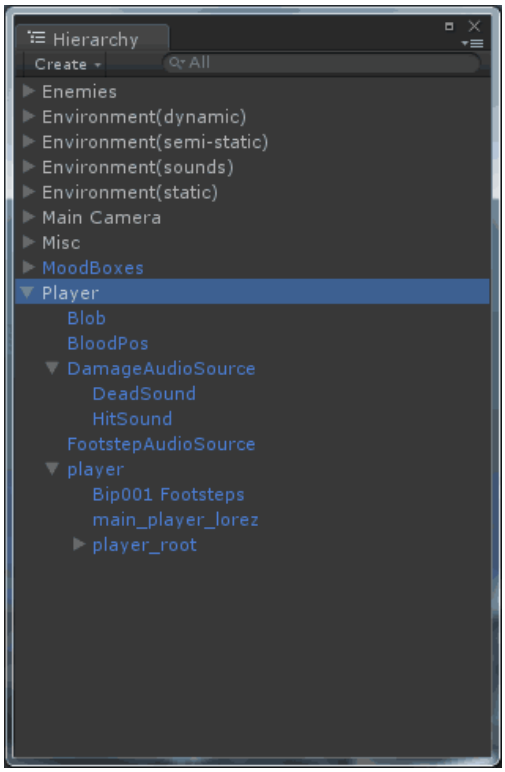


**Figura 17. Hierarchy: Ventana de jerarquías**

Los elementos pueden ser objetos "hijos" de otros, los cuales heredarán características tales como posición, rotación y escalado relativas al padre.

### 4) *Project*: Ventana de proyecto

La pestaña Project define todos los recursos definidos en el proyecto: Escenas, *prefabs*, texturas, audios, materiales, *scripts*, *shaders*, etcétera. En la Figura 18 se muestra un ejemplo de la jerarquía de recursos de un proyecto en esta pestaña.

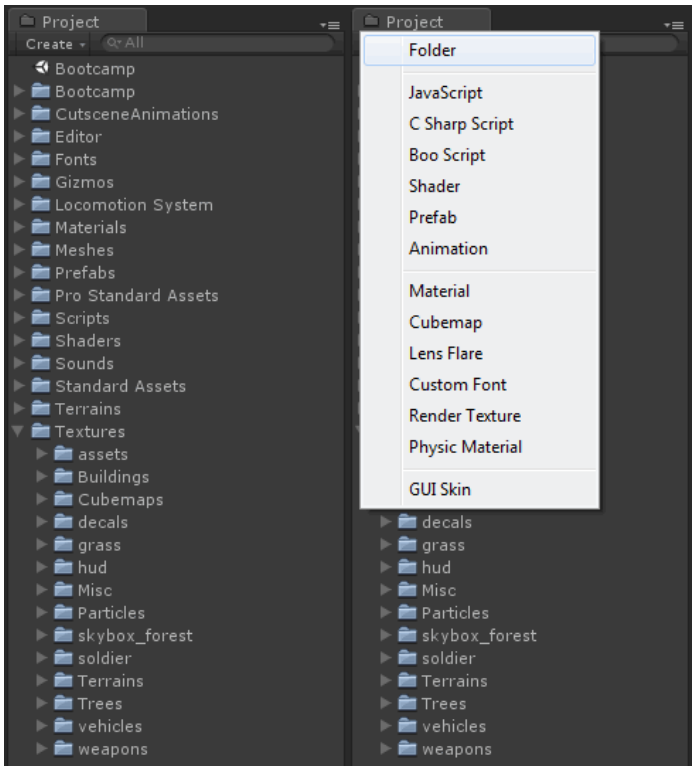


**Figura 18. Project: Ventana de proyecto**

La representación de las carpetas y de los ficheros de recursos en esta vista es muy significativa, de forma que adopta la misma estructura que tengan estos recursos en el disco (si se modifica desde aquí esa estructura también se modificará en el disco).

### 5) Inspector: Ventana de inspector

La pestaña *Inspector*, representa un conjunto de atributos y parámetros que definen un objeto seleccionado, ya sea en la pestaña *Hierarchy* o en la pestaña *Project.* A modo de ejemplo se presenta la configuración típica de una cámara en la Figura 19.

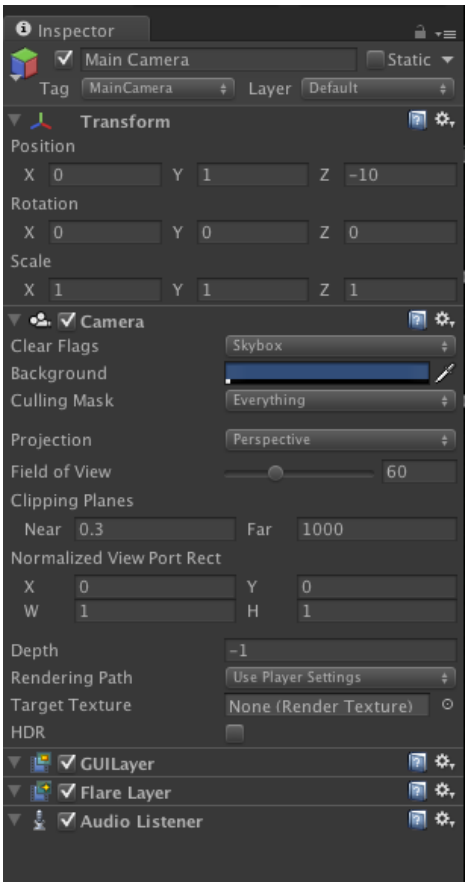


**Figura 19. Inspector: Ventana de Inspector**

A parte de las ventanas principales, existen otras ventanas también muy útiles, que podemos añadir en cualquier momento como:

- ***Console:*** En esta consola se registran todos los fallos y eventos que se produzcan en el juego o en el editor. Es muy útil para saber que está pasando, por qué nos está fallando algo o para lanzar mensajes de depuración de forma sencilla.
- ***Stats:*** Muestra las estadísticas de consumo de memoria, velocidad de refresco del juego, etcétera.
- ***Animation:*** Permite editar animaciones de objetos.
- ***Profiler:*** El *profiler* ayuda a optimizar el juego. En él se informa de cuánto tiempo se gasta en las diversas áreas del videojuego. Por ejemplo, se puede reportar el

porcentaje de tiempo dedicado al refresco de objetos en escena, animación o en la lógica del juego.

#### *2.3.4.4 Conceptos básicos de Unity*

- **GameObject**: El concepto más importante en *Unity*, el cual se muestra en la Figura 20, es sin duda el de ***GameObject***. En dicha figura podemos ver un *GameObject* vacío posicionado en el espacio y representados por coordenadas x, y, z. Estos objetos son la base de la arquitectura de *Unity*.

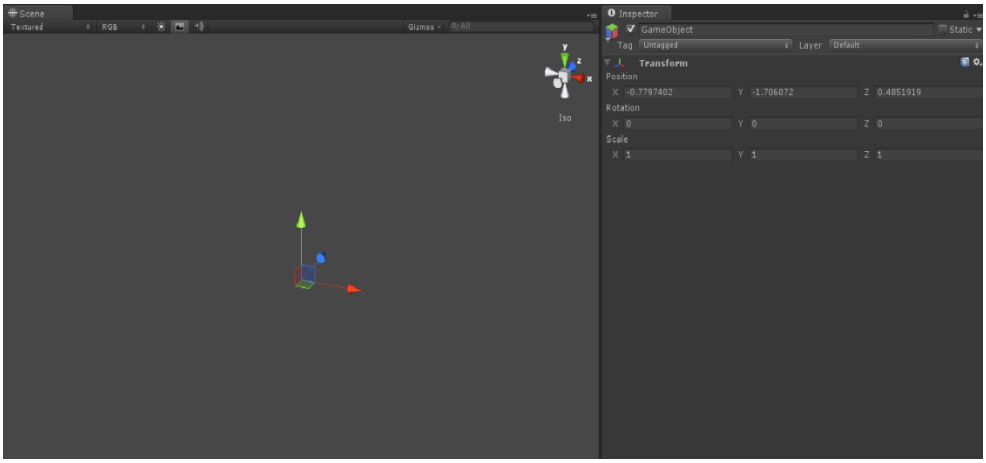


**Figura 20. Jerarquía básica de GameObject**

Para que estos objetos tengan funcionalidad y se conviertan en objetos con propiedades físicas, gráficas o con comportamientos específicos necesitan tener **componentes**. De esta forma combinando diferentes componentes podremos crear desde objetos inteligentes hasta piedras estáticas que solo son simples decorados.

Los *GameObjects* están formados por tres elementos básicos: un *layer* (capa de física, que sirve para gestionar de qué manera el objeto colisiona con otros objetos que tengan determinadas capas de física); un *tag* (etiqueta, se usa principalmente "marcar" un objeto y utilizar dicha marca a nivel de programación para

discriminarlo respecto de otros objetos que no tengan el mismo *tag*); y un componente *transform* que le proporciona al objeto la posición espacial, la orientación y la rotación dentro de la escena. Este componente es obligatorio para todos los *GameObjects* y no puede ser eliminado.

- ***Asset*:** Es la palabra que engloba cualquier tipo de recurso utilizado para el desarrollo en un motor de videojuegos. Recursos como modelos 3D, texturas, audios, vídeos, etcétera son *assets.* A continuación se muestran algunos de los formatos soportados por Unity de distintos tipos de *assets:*
  - Gráficos 3D: *Maya*, *Cinema 4D*, *3ds Max*, *Cheetah3D*, *Modo*, *Lightwave*, *Blender*, *.FBX*, *.dae*, *.3DS*, *.dxf* y *.obj*.
  - Gráficos 2D: *PSD, TIFF, JPG, TGA, PNG, GIF, BMP, IFF, PICT,* incluso los *PSD* multicapa y *TIFF* multicapa.
  - Audio: *WAV, AIFF, MP3, OGG* (éste último **no** es soportable en *iOS* y *Android*).
- **Prefab:** Otro concepto básico de *Unity* y que también es muy útil es el denominado *Prefab.* Estos objetos prefabricados (que se almacenan en la carpeta de Proyecto) nos permiten guardar diferentes tipos de *GameObjects* configurados como nosotros queramos, facilitándonos la reutilización de los mismos de forma que podemos instanciarlos en tiempo de juego o de edición, o modificar todas las instancias de una sola vez.
- **Escena:** Una escena es un conjunto de *GameObjects* con determinados componentes configurados de una forma concreta. Estas escenas pueden ser guardadas y luego cargadas tanto en el modo de editor como desde el propio juego, permitiéndonos hacer niveles. Cuando cargamos una escena, todos los elementos de esa escena se muestran en el panel de *Hierarchy*.

### *2.3.4.5 Flujo de Assets*

El flujo de los *Assets* en *Unity* es mucho más simple de lo habitual, ya que el editor los auto-importa permitiendo en la mayoría de las ocasiones añadir los recursos de forma automática, listos para ser usados en el motor.

Por ejemplo si se quiere añadir un modelo en 3D, se puede incluir en la carpeta de proyecto (con arrastrarlo a dicha carpeta es suficiente). Unity permite modificar una serie de parámetros del *Asset* auto-importado mediante la ventana del inspector y modificar algunos parámetros a nuestro gusto (por ejemplo se puede modificar la escala que tiene el *Asset* por defecto o añadirle un componente de animación). Este proceso sirve para cualquier tipo de *Asset*, ya sea modelo 3D, texturas, audio, videos, etcétera.

Otra forma de incluir *Assets* en el editor es importando ***Packages*** (paquetes de *Unity*). Estos *Packages* (Figura 21) contienen un conjunto de *Assets* que normalmente están relacionados y tienen una dependencia entre sí. Esto es muy útil cuando queremos llevarnos *GameObjects* o funcionalidades que involucran a muchos ficheros del proyecto a otro ordenador (pueden estar involucrados modelos, *scripts*, animaciones…).

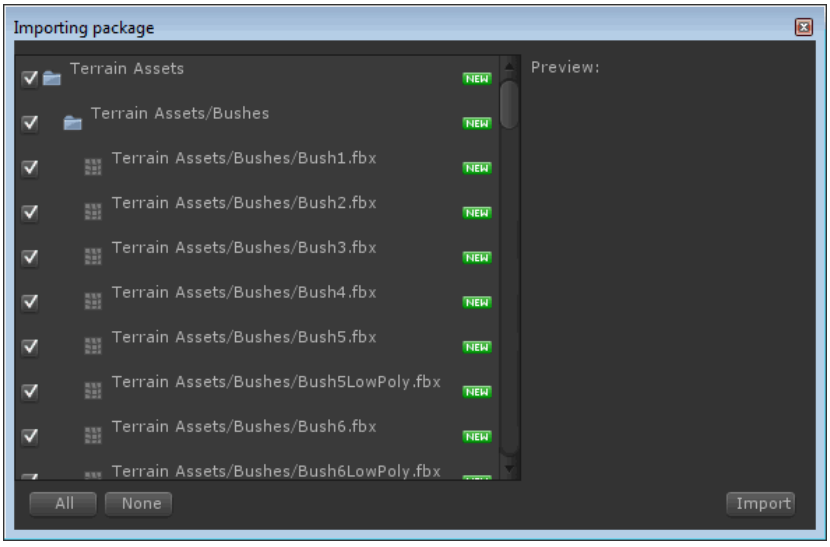


**Figura 21. Importación de Packages (paquetes)**

Al igual que podemos importar, también podemos exportar estos paquetes para poder ser importados después (Figura 22).

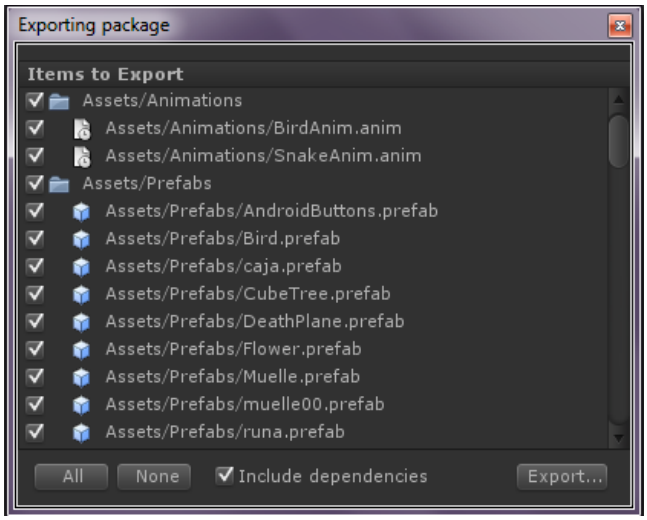


**Figura 22. Exportación de *packages* (paquetes)**

Todos estos *Assets* se organizan en el disco del ordenador donde estamos trabajando en una estructura similar a la que se puede observar en el panel de *Project,* de forma que si movemos una textura (por ejemplo) de una carpeta a otra, ésta se moverá de igual forma en el disco. Esto nos permite poder importar *Assets* simplemente añadiéndolos al disco duro y dejando que *Unity* los detecte.

#### *2.3.4.6 Scripting en Unity: Programación basada en componentes*

El mundo del juego está representado en *Unity* como un escenario compuesto por un conjunto de objetos, los cuales están formados a su vez por un conjunto de componentes que definen su comportamiento; a esto lo podemos llamar modelo de *GameObjects* basado en componentes.

En el modelo de *GameObjects* basado en componentes, existe un componente obligatorio en todos los objetos de escena que es el componente *GameObject*. Además como podemos ver en la Figura 23, se pueden agregar una serie de componentes a un objeto, donde cada uno de ellos define una propiedad o comportamiento. Esto hace que el sistema sea muy modular y permita tener clases mucho más pequeñas y manejables.

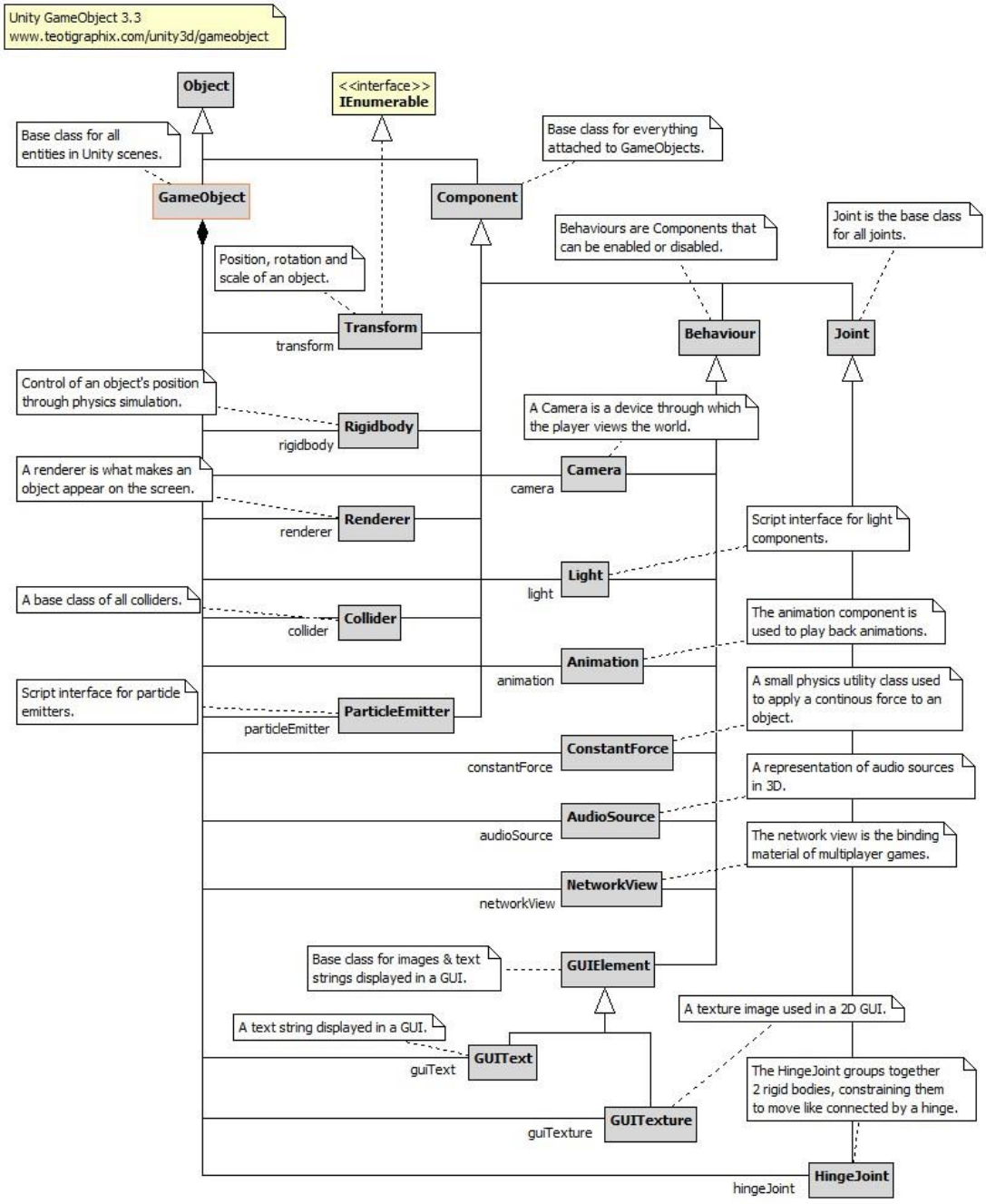


**Figura 23. Diagrama de clases de UnityEngine.GameObject**

A continuación de describen los componentes representados en la Figura 23:

- *Transform*: Componente encargado del posicionamiento, rotación y escalado en el mundo de juego.
- *RigidBody:* Encargado del posicionamiento de un objeto que esté influido por el motor de físicas.
- *Renderer:* Este componente se encarga de representar por pantalla un objeto determinado.
- *Collider:* Estructura geométrica invisible que interviene en las colisiones calculadas en el motor de físicas.
- *ParticleEmitter:* Permite configurar un emisor de partículas en un objeto. La representación gráfica de las partículas, la velocidad de emisión, aleatoriedad de aparición, etcétera, son algunos de los parámetros configurables de este componente.
- *Camera:* Este componente establece la región que se quiere representar por pantalla.
- *Light:* Este componente emula la representación de una luz. Se pueden configurar su color, su intensidad, rango de emisión, etcétera.
- *Animation:* Permite ejecutar animaciones de modelos 3D.
- *ConstantForce:* Componente usado para aplicar fuerzas constantes a objetos.
- *AudioSource:* Permite que un objeto pueda emitir sonidos.
- *NetworkView:* Componente que permite serializar datos para replicarlos por red.
- *GUIText:* Componente que permite mostrar texto en la interfaz de usuario.
- *GUITexture:* Componente que permite mostrar texturas en la interfaz de usuario.
- *HingeJoint:* Realiza la unión física entre 2 *RIgidBodies;* el motor de físicas hace que este componente funcione como bisagra.

Por ejemplo, si un *GameObject* tiene un componente de *Rigidbody*, como se ha dicho éste se considera por *Unity* como un cuerpo físico y es procesado por el motor de físicas: las fuerzas gravitatorias se aplicarán al objeto. Si el objeto tiene además un componente

*Collider*, en ejecución el objeto puede "chocar" con otros objetos que tengan *Colliders* además de verse afectado por la gravedad.

La mayoría de las secuencias de comandos que un programador de *Unity* escribe son clases que heredan del componente *MonoBehaviour*. Para controlar el objeto, este tipo de *scripts* pueden reemplazar un conjunto de métodos virtuales predefinidos. Queda claro que además de todos los componentes definidos, un programador puede desarrollar sus propios componentes y crear funcionalidades adicionales.

# Capítulo 3: Descripción del sistema

## 3.1 Introducción

En este capítulo se realiza una descripción detallada de las tres principales fases del desarrollo de un proyecto. En el apartado 3.2 se presenta el análisis del sistema, en el cual se realizó una identificación de los requisitos y funcionalidades que debía tener el videojuego. En el apartado 3.3 se presenta el diseño del sistema mediante un diagrama de clases que es descrito de forma detallada con el fin de describir la estructura del videojuego que ha sido desarrollado en este proyecto.

## 3.2 Análisis del sistema

En este apartado se presentan las diferentes características y requisitos que han sido identificados para el desarrollo del videojuego GyroWorld. El objetivo principal de este proceso consiste en detallar aquéllas funcionalidades que debe tener nuestra aplicación.

### 3.2.1 Descripción de las características funcionales

A continuación vamos a describir las características que tiene que cumplir nuestro sistema. Se partirá de una descripción general de la idea de juego, pasando por el conjunto de mecánicas centrales que definen el entorno de juego.

- **Descripción de juego:** GyroWorld es una vuelta de tuerca a los juegos de plataformas convencionales. En este juego 2.5D (Objetos del escenario 3D vistos desde una cámara con perspectiva 2D) controlaremos a una simpática pelota que irá avanzando en mundos circulares. Tendremos que demostrar nuestra destreza superando obstáculos y conseguir monedas en este mundo circular que se mueve bajo nuestros pies, repleto de trampas y sorpresas.

- **Plataforma:** La plataforma objetivo será principalmente dispositivos móviles con sistema operativo *Android*. Simultáneamente se creará una aplicación análoga para ordenador con la finalidad principal de poder agilizar el periodo de pruebas.
- **Interfaz de usuario:** El usuario tendrá como elementos de comunicación de entrada y salida con el videojuego los que se detallan a continuación:
  - Entrada: El usuario podrá mover al personaje como se detalla en el apartado 7.2.2.
  - Salida: Se hace una representación visual de los menús y del mundo de juego mediante la pantalla del dispositivo. Así mismo se usan los altavoces para reproducir sonidos que complementan la experiencia del jugador.
- **Ambientación:** *GyroWorld* no tiene historia, pero sí que debe tener un conjunto de mundos temáticos y coloristas, con estéticas muy bien definidas visualmente. Los enemigos tienen que presentar un aspecto visual coherente con el resto del mundo.
- **Movimiento:** El personaje principal puede **moverse** y **saltar**. Todos los movimientos tendrán en cuenta la inercia del personaje y el rozamiento con el mundo; al ser un mundo "que se mueve", estaremos muy a menudo en pendientes que dependiendo de si son ascendentes o descendentes complicarán el movimiento.
- **Mundo de juego:** Serán anillos que nos atraerán a su superficie, por lo tanto tendremos que movernos en consonancia con el mundo. La velocidad de giro de los anillos será variable en función de lo rápido que se mueva el personaje. En la Figura 24 se muestra la forma de un mundo del juego:

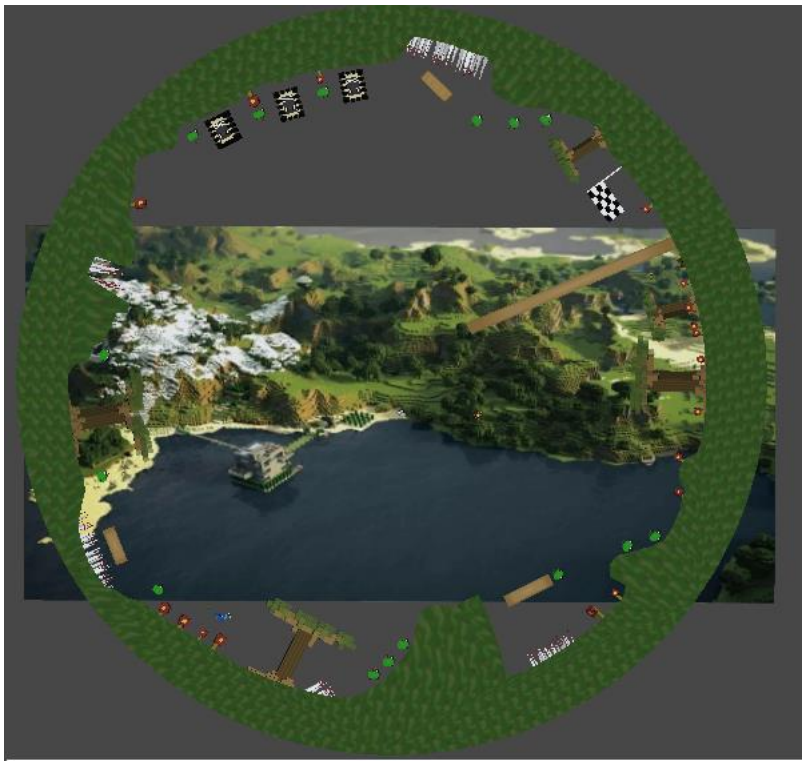

Figura 24. Mundo de juego

- **Acciones Especiales:** Como caso excepcional el personaje principal dispondrá de un salto extra cuando se encuentre en el aire. El impulso del mismo y la dirección de este salto dependerá de la combinación de la inercia del personaje y la dirección que elija el usuario para realizarlo. Esta última característica se introdujo por primera vez en Super Mario Bros.
- **Enemigos:** Existirán enemigos que tendremos que evitar para que no nos maten. Cada uno de ellos tendrá un comportamiento diferente e interpretable por el usuario para superarlos de manera satisfactoria.
  - Pájaro: Enemigo volador que lanzará huevos cuando el personaje se encuentre cerca.
  - Serpiente: Este enemigo escupe veneno en la dirección del personaje.
  - Machacador: Aplasta al personaje si éste no se aparta a tiempo al pasar por debajo de él.
- **Salud:** A principio de cada nivel tendremos una serie de *slots* de vida. Al perder todos los *slots*, volveremos al menú de selección de niveles. Cada vez que

perdamos un *slot,* volveremos al principio del nivel, pero con los puntos recogidos acumulados.

- **Niveles:** Los niveles estarán predefinidos, no se generarán de forma procedural.
- **Flujo de funcionamiento:** Al superar un nivel, se desbloqueará el siguiente. Esta acción será persistente en la aplicación, por lo que cada vez que ejecutemos la misma, el registro de niveles superados y por consiguiente niveles desbloqueados, quedará guardado.

### 3.2.2 Restricciones del sistema

A continuación se presenta la descripción de las restricciones que tendrá el sistema a desarrollar. Estas se dividen en dos categorías: Aquéllas que son impuestas por el *hardware* de los dispositivos utilizados y las que provienen del *software* utilizado para el desarrollo de todas las características del sistema.

#### *3.2.2.1 Restricciones hardware*

Son aquellas producidas por las características *hardware* de los diferentes dispositivos en los cuales será desplegado el videojuego desarrollado en este proyecto. A continuación se presentan las restricciones generales de los diferentes dispositivos en los cuales podrá ser ejecutado el videojuego. Además se presentarán algunas características específicas de los dispositivos que han sido relevantes a la hora de desarrollar el videojuego.

- **Capacidad de cómputo de los dispositivos:** Los dispositivos móviles no son muy potentes, por lo que en desarrollo hay que tener en cuenta que la inclusión de recursos no optimizados puede afectar muy negativamente al rendimiento del videojuego. Además teniendo en cuenta que hay dispositivos en el mercado con capacidades muy diferentes, es difícil saber si el videojuego va a poder funcionar en la mayoría de los dispositivos sin probarlos a priori.

- **Características de visualización:** Actualmente existe una amplia cantidad de dispositivos con diferentes resoluciones y tamaños de pantallas, que van desde pantallas de 3 pulgadas hasta tablets de 10.1 pulgadas. Además dependiendo del modelo de dispositivo, la resolución de éste puede ser diferente, lo cual puede influir en algunos detalles gráficos.

- **Interrupción del videojuego:** Los dispositivos que ejecutan el sistema operativo Android y/o iOS poseen un botón de inicio. Este botón cambia el estado de la aplicación que se esté ejecutando a estado de pausa, devolviendo al usuario a la pantalla de inicio. La aplicación no tiene ningún control de este evento y no se puede bloquear. Mediante este evento, la aplicación recibe una notificación de que va a segundo plano. Además en algunas situaciones la aplicación puede ser detenida por el sistema operativo sin recibir ninguna notificación. Esto produce que los datos que no han sido almacenados se perderán. En la Figura 25 se observa el ciclo de vida de una actividad en *Android*. Toda aplicación desarrollada sobre el sistema operativo *Android* debe estar formada por al menos una actividad.

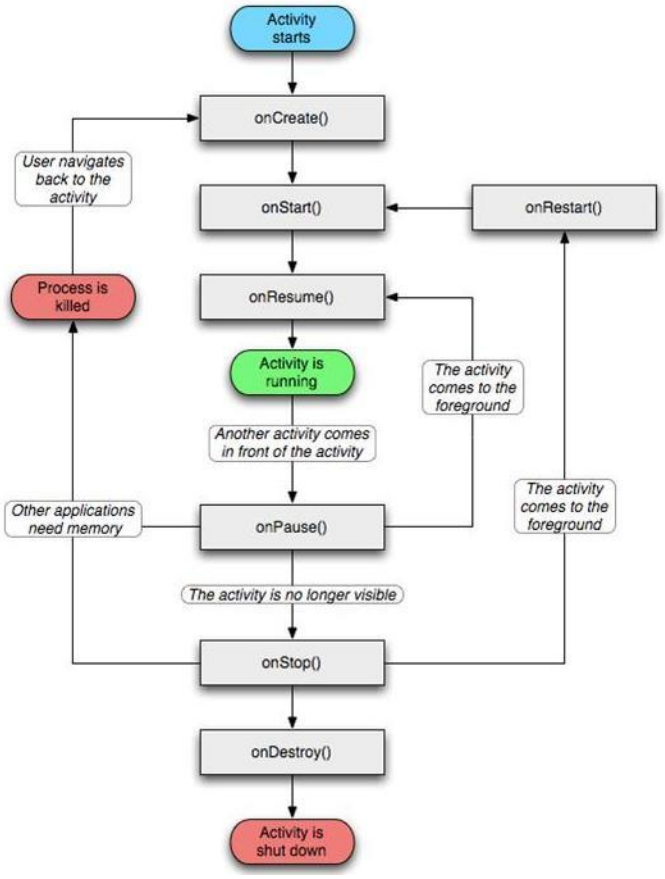


**Figura 25. Ciclo de vida de una actividad en *Android***

- **Limitación de recursos:** Los *smartphones* actuales pueden estar sin recargarse de media entre 1 y 3 días. Sin embargo, el consumo de la batería aumenta en gran medida cuando se ejecutan aplicaciones, especialmente las aplicaciones de uso intensivo de la pantalla.
- **Fiabilidad del *hardware*:** Existen un conjunto de dispositivos (acelerómetros, pantalla táctil, etcétera) que pueden ofrecer diferentes resultados dependiendo del *smartphone*, de forma que hay que adaptar el videojuego para que pueda ser utilizado dependiendo de estas variaciones.

#### *3.2.2.2 Restricciones software*

Este tipo de restricciones son derivadas del sistema operativo y de las tecnologías utilizadas para el desarrollo de la aplicación.

- **Entorno de desarrollo:** Se ha seleccionado *Unity* en su versión 3.5.
- **Lenguaje de programación:** C#.
- **Sistema operativo de desarrollo:** *Windows.*
- **Kit de desarrollo:** Android SDK (versión 19).
- **Dispositivos móviles de testeo:** Se han utilizado los siguientes dispositivos:
  - Dispositivo móvil *BQ Aquaris 5 HD*. *CPU*: *Quad Core Cortex A7* hasta 1,2 GHz. *GPU: PowerVR SGX544* hasta 286 MHz. Memoria *RAM* de 1GB. 5 pulgadas con resolución 720x1280 px.
  - *Tablet Connection CNC-TAB7*: Procesador ARMv7 1,2GHz. 1GB de RAM, memoria interna de 8GB. 7 pulgadas con resolución 800x400 px.

### 3.2.3 Entorno de desarrollo

A continuación se describen los distintos dispositivos necesarios para la realización de este trabajo.

#### *3.2.3.1 Entorno operacional Hardware*

- Ordenador *Acer Aspire 5742G*. *i5-480M*. Tarjeta gráfica *NVIDIA GeForce GT 540M*.

#### *3.2.3.2 Entorno operacional Software*

- Sistema operativo *Windows 7*.
- *Unity* con licencia Pro para desarrollo *Android* versión 3.5.7f6.
- Entorno de programación *Monodevelop* (integrado por defecto con *Unity*) 2.8.2.
- Sistema operativo del terminal móvil usado para pruebas *Android 4.2 Jelly Bean*.

### 3.2.4 Especificación de casos de uso

Los casos de uso ayudan a definir, con cierto nivel de abstracción, las relaciones que hay entre los actores (jugador en este caso), y el sistema.

#### *3.2.4.1 Descripción de los actores*

En el caso de la aplicación a desarrollar, el único actor por el que se describirán casos de uso del juego y requisitos (tanto funcionales como no funcionales) lógicamente será el **Jugador**.

#### *3.2.4.2 Descripción de los atributos de los casos de uso*

Para la realización de la descripción textual de los distintos casos de uso, se han seleccionado una serie de atributos que describen cada uno de los casos de uso. A continuación se realiza una descripción del significado de cada uno de los atributos utilizados para la descripción de los casos de uso.

- **Código:** Identificación unívoca abreviada del caso de uso, se construye mediante CU (Caso de Uso) seguido de un - y de tres dígitos, lo que representará de forma unívoca el caso. Ejemplo: CU – 001.
- **Nombre:** Identificación breve del caso de uso.
- **Actores:** Conjunto de entidades que interactúan con el caso de uso. El caso de uso representa una funcionalidad demandada por un actor.
- **Descripción:** Se realiza una descripción básica de la funcionalidad o funcionalidades del caso de uso.
- **Precondiciones y post-condiciones:** Se realiza una descripción de las condiciones que deben cumplirse para poder realizar una operación, y el estado en el que queda el sistema tras realizar una operación.
- **Escenario:** Se realiza una descripción básica de las acciones que se ejecutarán paso a paso en el caso de uso.

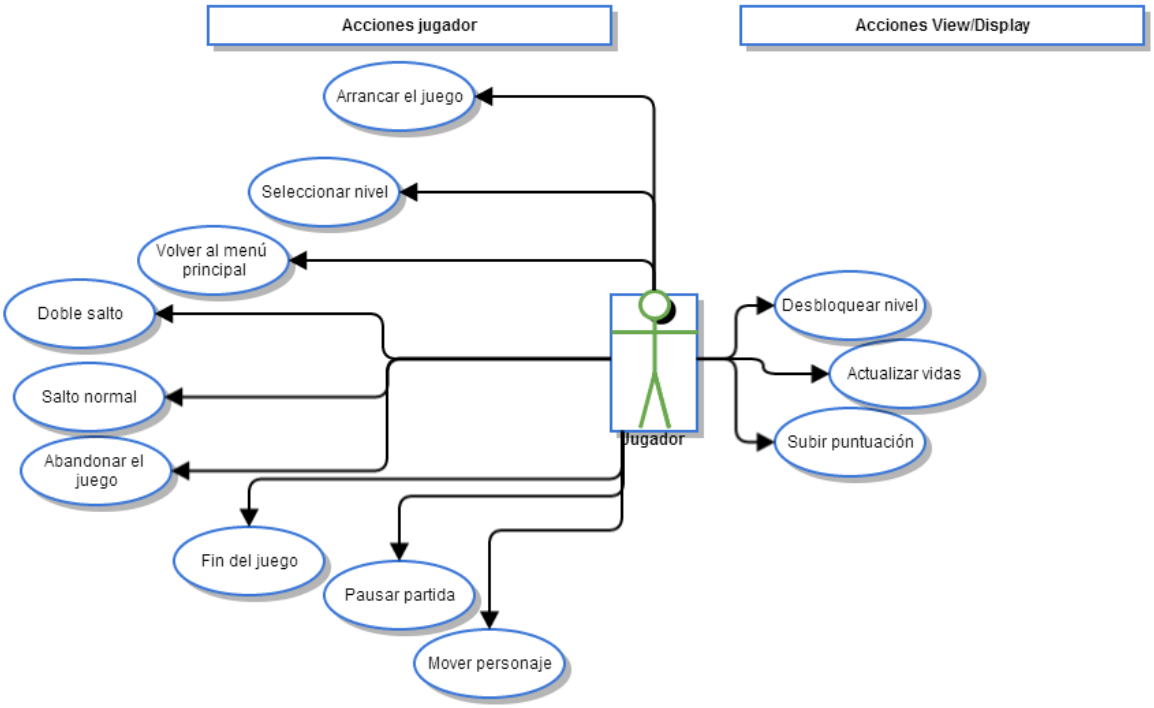


Figura 26. Diagrama de los casos de uso

### *3.2.4.3 Descripción textual de los casos de uso*

| Caso de uso | |
|---|---|
| **Código** | CU – 001 |
| **Nombre** | Arrancar el juego |
| **Actores** | Jugador |
| **Descripción** | Comienza el juego |
| **Precondiciones** | Tener instalada la aplicación en el terminal |
| **Poscondiciones** | Se carga el menú principal del juego. El jugador tendrá varias condiciones para elegir en el menú |

Tabla 2. CU – 001. Arrancar el juego

| Caso de uso | |
|---|---|
| **Código** | CU - 002 |
| **Nombre** | Selección de nivel |
| **Actores** | Jugador |
| **Descripción** | Seleccionar el nivel al que se desea jugar |
| **Precondiciones** | Haber arrancado el juego y pulsado “Play” |
| **Poscondiciones** | Se carga el nivel seleccionado |

Tabla 3. CU – 002. Selección de nivel

| Caso de uso | |
|---|---|
| **Código** | CU – 003 |
| **Nombre** | Volver a menú principal |
| **Actores** | Jugador |
| **Descripción** | Volver al menú principal |
| **Precondiciones** | Estar en la pantalla de selección de nivel o en modo pausa tras cargar un nivel |
| **Poscondiciones** | Se carga la pantalla de menú principal tras pulsar el botón Back |

Tabla 4. CU – 003. Volver al menú principal

| Caso de uso | |
|---|---|
| **Código** | CU – 004 |
| **Nombre** | Abandonar juego |
| **Actores** | Jugador |
| **Descripción** | El jugador podrá salir del juego al pulsar el botón exit |
| **Precondiciones** | Estar en el menú principal |
| **Poscondiciones** | Se cerrará la aplicación cuando el jugador pulse el botón exit |

Tabla 5. CU – 004. Abandonar juego

| Caso de uso | |
|---|---|
| **Código** | CU - 005 |
| **Nombre** | Pausar partida |
| **Actores** | Jugador |
| **Descripción** | Pausa el juego. Adicionalmente se mostrará un cuadro de diálogo. |
| **Precondiciones** | Haber pulsado el botón pausa con un nivel cargado |
| **Poscondiciones** | El juego permanecerá pausado hasta que no seleccionemos la opción Resume (continuará el juego) o la opción Back to Main Menu (Abandonar partida y salir al menú principal) |

Tabla 6. CU – 005. Pausar partida

| Caso de uso | |
|---|---|
| **Código** | CU – 006 |
| **Nombre** | Mover personaje |
| **Actores** | Jugador |
| **Descripción** | Mover el personaje por las zonas de pantalla habilitadas a tal efecto (plano XY, plano de pantalla) |
| **Precondiciones** | Haber cargado un nivel |
| **Poscondiciones** | El personaje se moverá un valor proporcional al acelerómetro |

Tabla 7. CU – 006. Mover personaje

| Caso de uso | |
|---|---|
| **Código** | CU - 007 |
| **Nombre** | Salto normal |
| **Actores** | Jugador |
| **Descripción** | El personaje realizará un salto |
| **Precondiciones** | Haber pulsado la pantalla táctil durante la partida mientras el personaje esté en contacto con el suelo. |
| **Poscondiciones** | El personaje sufrirá un impulso hacia arriba, realizando así un salto |

Tabla 8. CU – 007. Salto normal

| Caso de uso | |
|---|---|
| **Código** | CU – 008 |
| **Nombre** | Doble salto |
| **Actores** | Jugador |
| **Descripción** | El personaje realizará un salto extra |
| **Precondiciones** | No estar en contacto con el suelo y no haber realizado otro salto extra desde la última vez que se estuvo en contacto con el suelo |
| **Poscondiciones** | El personaje sufrirá un impulso extra, con fuerza y dirección dependientes de la posición de la pantalla en la que se pulse |

Tabla 9. CU – 008. Doble salto

| Caso de uso | |
|---|---|
| **Código** | CU – 009 |
| **Nombre** | Subir puntuación |
| **Actores** | Jugador |
| **Descripción** | Actualizar puntuación de la partida |
| **Precondiciones** | Haber iniciado partida y haber cogido una manzana |
| **Poscondiciones** | Se actualiza la puntuación de la partida |

Tabla 10. CU – 009. Subir puntuación

| Caso de uso | |
|---|---|
| **Código** | CU – 010 |
| **Nombre** | Actualizar vidas |
| **Actores** | Jugador |
| **Descripción** | Actualizar el número de vidas |
| **Precondiciones** | Haber iniciado partida y haber colisionado con un elemento que provoca la muerte |
| **Poscondiciones** | Se actualiza el número de vidas de la partida tras pulsar la opción Continue playing en el diálogo de texto de muerte. Vuele el personaje al punto inicial |

Tabla 11. CU – 010. Actualizar vidas

| Caso de uso | |
|---|---|
| **Código** | CU -011 |
| **Nombre** | Fin del juego |
| **Actores** | Jugador |
| **Descripción** | Termina el juego |
| **Precondiciones** | Haber iniciado partida y haber perdido todos los slots de vida |
| **Poscondiciones** | Se vuelve al menú principal tras pulsar en el cuadro de diálogo emergente la única opción disponible Back to Main Menu |

Tabla 12. CU – 011. Fin del juego

| Caso de uso | |
|---|---|
| **Código** | CU – 012 |
| **Nombre** | Desbloquear nivel |
| **Actores** | Jugador |
| **Descripción** | Desbloquea el siguiente nivel bloqueado |
| **Precondiciones** | Haber llegado al final de un nivel (zona de la bandera) |
| **Poscondiciones** | Se carga la escena de selección de nivel con el siguiente nivel desbloqueado si antes lo estaba |

Tabla 13. CU – 012. Desbloquear nivel

### 3.2.5 Especificación de requisitos

En este apartado se define el conjunto de requisitos necesarios para la realización del videojuego, así como los atributos que describen a cada requisito.

#### *3.2.5.1 Descripción de los atributos de los requisitos*

Para la realización de la descripción textual de los distintos requisitos que han sido identificados, se han seleccionado una serie de atributos que describen cada uno de los requisitos. A continuación se realiza una descripción del significado de cada uno de los atributos utilizados para su descripción:

- Código: Identificación unívoca abreviada del requisito, se construye mediante el código del requisito seguido de un - y de tres dígitos. Los requisitos serán divididos en funcionales y no funcionales y sus códigos son RF para los requisitos funcionales y RNF para los requisitos no funcionales. Por ejemplo RF – 001, RNF - 003.
- Nombre: Identificación breve del requisito.
- Descripción: Se realiza una descripción básica del requisito que ha sido identificado.

- Fuente: Indica a través de que fuente ha sido identificado el requisito. Normalmente este valor se corresponderá con uno o varios códigos de los casos de uso.
- Necesidad: Determina el grado de implementación del requisito. Los valores que puede tomar este atributo son los siguientes:
    - Esencial: El requisito tiene que ser implementado.
    - Deseable: Es preferible implementar el requisito, pero no es obligatorio.
    - Opcional: El requisito se podrá implementar, pero no es importante ni obligatorio.
- Prioridad: Define la importancia del requisito, de forma que permita definir el orden en el cual serán incluidos en el proceso de diseño e implementación. Los valores que puede tomar este atributo son los siguientes:
    - Alta: El requisito debe ser implementado en las fases iniciales del desarrollo.
    - Media: El requisito debe ser implementado una vez que hayan sido implementados los requisitos de prioridad alta.
    - Baja: El requisito debe ser implementado en las fases finales del desarrollo. Estos requisitos no influirán en el correcto funcionamiento del sistema.
- Estabilidad: Define la estabilidad del requisito durante la vida útil del software. Esto implica si el requisito podrá ser o no modificado durante el ciclo del vida. Los valores que puede tomar este atributo son los siguientes:
    - Estable: El requisito no puede variar durante el ciclo de vida del sistema.
    - Inestable: El requisito puede variar a lo largo de la ciclo de vida del sistema.
- Verificabilidad: Define el grado de verificabilidad de un requisito, es decir indica en qué grado es posible comprobar que el requisito se ha incorporado en el sistema desarrollado. Los valores que puede tomar este atributo son los siguientes:
    - Alta: Se puede verificar que el requisito ha sido implementado en el sistema. Este tipo de requisitos se corresponden con las funcionalidades básicas del sistema.

- Media: Se puede verificar que el requisito ha sido implementado en el sistema. Pero requiere de una comprobación compleja o del código fuente del sistema.
- Baja: Es difícil verificar si el requisito ha sido implementado en el sistema o en algunos casos no es posible.

#### *3.2.5.2 Especificación de requisitos*

En este apartado se presentan los requisitos funcionales del videojuego desarrollado.

**Requisitos Funcionales**

| **Requisito del sistema** | | | |
|---|---|---|---|
| **Código** | **RF - 001** | **Fuente** | CU – 001 |
| **Nombre** | Lanzar el juego | | |
| **Descripción** | El juego se lanzará mediante la ejecución de un fichero .apk que se genera al compilar la aplicación | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 14. RF – 001. Lanzar el juego

| **Requisito del sistema** | | | |
|---|---|---|---|
| **Código** | **RF - 002** | **Fuente** | |
| **Nombre** | Mostrar menú principal | | |
| **Descripción** | Cuando se lanza el juego, se debe mostrar una pantalla con las distintas opciones del menú: Play, Exit | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 15. RF – 002. Mostrar menú principal

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 003** | **Fuente** | CU – 002 |
| **Nombre** | Elección de nivel | | |
| **Descripción** | Al pulsar Play en la pantalla de menú, se accederá a otra pantalla donde podremos seleccionar entre los niveles disponibles | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Inestable | **Verificabilidad** | Alta |

Tabla 16. RF – 003. Elección de nivel

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF - 004** | **Fuente** | CU – 002 |
| **Nombre** | Pantalla de juego | | |
| **Descripción** | La pantalla de juego mostrará es escenario, los puntos acumulados, las vidas restantes, los enemigos, un botón de pausa, el personaje principal y las entidades con las que interacciona el jugador | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Inestable | **Verificabilidad** | Alta |

Tabla 17. RF – 004. Pantalla de juego

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF - 005** | **Fuente** | CU – 005 |
| **Nombre** | Mostrar el menú de pausa | | |
| **Descripción** | El jugador debe tener la posibilidad de pausar el juego en cualquier momento de la partida | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 18. RF – 005. Mostrar el menú de pausa

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 006** | **Fuente** | CU – 007 |
| **Nombre** | Salto normal | | |
| **Descripción** | Al pulsar en cualquier punto de la pantalla, si el personaje está en contacto con la superficie, éste saltará. | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Inestable | **Verificabilidad** | Alta |

Tabla 19. RF – 006. Salto normal

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 007** | **Fuente** | CU – 006 |
| **Nombre** | Movimiento | | |
| **Descripción** | El personaje se moverá en función de los acelerómetros del dispositivo móvil | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Inestable | **Verificabilidad** | Alta |

Tabla 20. RF – 007. Movimiento

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF - 008** | **Fuente** | CU - 012 |
| **Nombre** | Vidas | | |
| **Descripción** | El usuario contará con un número limitado de vidas durante cada partida | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 21. RF – 008. Vidas

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 009** | **Fuente** | CU – 004 |
| **Nombre** | Salir del juego | | |
| **Descripción** | El juego tiene que tener la capacidad de cerrarse a sí mismo cuando se quiera dejar de jugar | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 22. RF – 009. Salir del juego

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 010** | **Fuente** | CU – 008 |
| **Nombre** | Doble salto | | |
| **Descripción** | El personaje podrá realizar un salto adicional extra en el aire, que dependerá de la posición de la pantalla donde se pulse | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 23. RF – 010. Doble salto

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 011** | **Fuente** | CU - 012 |
| **Nombre** | Guardado de partida | | |
| **Descripción** | El jugador cada vez que supere un nuevo nivel, en el caso de que existan más niveles, el siguiente quedará desbloqueado para poder ser elegido | | |
| **Necesidad** | Deseable | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Media |

Tabla 24. RF – 011. Guardado de partida

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 012** | **Fuente** | CU - 011 |
| **Nombre** | Coger puntos | | |
| **Descripción** | El personaje al tocar manzanas éstas serán "recogidas", actualizando así el contador de puntos | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 25. RF – 012. Coger puntos

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 013** | **Fuente** | |
| **Nombre** | Gravedad | | |
| **Descripción** | La gravedad en el juego será poco convencional: Serán mundos circulares en forma de anillos, y el personaje se verá atraído por la superficie de éstos | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 26. RF – 013. Gravedad

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 014** | **Fuente** | CU - 010 |
| **Nombre** | Menú de muerte | | |
| **Descripción** | Cada vez que se pierda una vida, se debe mostrar un menú en el que se le da al jugador las opciones de seguir jugando o salir al menú principal | | |
| **Necesidad** | Deseable | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 27. RF – 014. Menú de muerte

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 015** | **Fuente** | CU – 010, CU -011 |
| **Nombre** | Menú fin de juego | | |
| **Descripción** | Al perder todas las vidas, se le avisará al jugador que ha perdido la partida y se mostrará un botón con el que volver al menú principal | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 28. RF – 015. Menú fin de juego

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 016** | **Fuente** | CU - 010 |
| **Nombre** | Enemigo Pájaro | | |
| **Descripción** | Este enemigo volará entre dos o más puntos del escenario de manera secuencial. Cuando el personaje esté dentro de su radio de acción, el pájaro soltará huevos que caerán según la gravedad del mundo. Si el personaje es tocado por uno de estos huevos, pierde una vida. | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 29. RF – 016. Enemigo Pájaro

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 018** | **Fuente** | CU - 010 |
| **Nombre** | Enemigo Serpiente | | |
| **Descripción** | Este enemigo se encontrará fijo en el escenario. Cuando el personaje esté dentro de su rango de acción, la serpiente escupirá veneno cada cierta frecuencia en la dirección en la que se encuentre el personaje. Si el personaje es tocado por veneno, pierde una vida. | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 30. RF – 018. Enemigo Serpiente

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 019** | **Fuente** | CU - 010 |
| **Nombre** | Pinchos | | |
| **Descripción** | Si el personaje toca los pinchos, pierde una vida | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 31. RF – 019. Pinchos

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 020** | **Fuente** | CU - 010 |
| **Nombre** | Enemigo Machacador | | |
| **Descripción** | Este enemigo realizará movimientos verticales entre dos puntos. Si el personaje toca a este enemigo, pierde una vida | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 32. RF – 020. Enemigo Machacador

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 021** | **Fuente** | CU - 012 |
| **Nombre** | Bandera | | |
| **Descripción** | Si el personaje alcanza este elemento, se terminará el nivel actual, dando la enhorabuena al usuario con un pop-up y cargando a continuación el menú de selección de niveles | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 33. RF – 021. Bandera

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 022** | **Fuente** | CU - 010 |
| **Nombre** | Respawn | | |
| **Descripción** | Cada vez que se pierda una vida, el personaje volverá al comienzo del nivel | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 34. RF – 022. Respawn

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 023** | **Fuente** | |
| **Nombre** | Tablones de madera | | |
| **Descripción** | Este elemento podrá moverse o no y el personaje podrá colisionar con él, así como usarlo de superficie para superar ciertos obstáculos | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 35. RF – 023. Tablones de madera

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 024** | **Fuente** | CU - 010 |
| **Nombre** | Bola de pinchos | | |
| **Descripción** | Este elemento giratorio recubierto de pinchos estará fijo en un punto del espacio. Si el personaje toca a este elemento, pierde una vida | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 36. RF – 024. Bola de pinchos

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 025** | **Fuente** | CU - 010 |
| **Nombre** | Vibración | | |
| **Descripción** | Cuando perdemos una vida, el dispositivo deberá vibrar como señal de muerte | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 37. RF – 025. Vibración

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 026** | **Fuente** | CU -001, CU -002 |
| **Nombre** | Sonido Menú | | |
| **Descripción** | Siempre que se esté en el menú principal o en la pantalla de selección de niveles, deberá sonar una melodía | | |
| **Necesidad** | Deseable | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 38. RF – 026. Sonido Menú

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 027** | **Fuente** | CU -002 |
| **Nombre** | Sonido de juego | | |
| **Descripción** | Siempre que se esté en un nivel, deberá sonar una melodía | | |
| **Necesidad** | Deseable | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 39. RF – 027. Sonido de juego

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RF – 028** | **Fuente** | CU - 009 |
| **Nombre** | Sonido Manzanas | | |
| **Descripción** | Cada vez que el personaje recoja una manzana, sonará un sonido indicativo | | |
| **Necesidad** | Deseable | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 40. RF – 028. Sonido Manzanas

**Requisitos no funcionales**

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RNF – 001** | **Fuente** | CU – 001, CU - 011 |
| **Nombre** | Aplicación en *Android* | | |
| **Descripción** | La aplicación debe ser desarrollada para *Android* | | |
| **Necesidad** | Esencial | **Prioridad** | Alta |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 41. RNF – 001. Aplicación en *Android*

| Requisito del sistema | | | |
|---|---|---|---|
| **Código** | **RNF – 002** | **Fuente** | |
| **Nombre** | Utilización de *Unity* | | |
| **Descripción** | Se deberá usar el motor de desarrollo *Unity* | | |
| **Necesidad** | Esencial | **Prioridad** | Media |
| **Estabilidad** | Estable | **Verificabilidad** | Alta |

Tabla 42. RNF – 002. Utilización de *Unity*

## 3.3 Diseño del sistema

El objetivo es realizar un diseño correcto para minimizar las posibles correcciones que se deban hacer en la fase de implementación y así cumplir con los plazos de entrega establecidos.

Se ha dividido este apartado en 4 sub-secciones:

- **Arquitectura del sistema:** Definición a alto nivel de la estructura del juego.
- **Descripción general del sistema:** Se muestran los diseños realizados para cada pantalla del juego, así como los diagramas de flujo de la aplicación.
- **Descripción de componentes:** Se especifican los componentes derivados de la arquitectura del sistema, contando su funcionalidad para la explicación del funcionamiento de los mismos.
- **Carpetas del proyecto:** Muestra el árbol de carpetas de la aplicación.

### 3.3.1 Arquitectura del sistema

La arquitectura de juego, mostrada en la Figura 27, se compone de dos módulos principales, El núcleo de la aplicación y el módulo de Entrada/Salida. El núcleo de la aplicación debe encargarse de toda la creación y gestión de las entidades de la aplicación, así como sus ciclos de vida. El módulo de Entrada/Salida se encarga de la comunicación direccional entre el usuario y el juego.

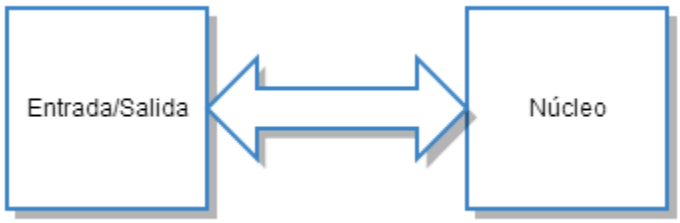


Figura 27. Diagrama de módulos de la arquitectura del juego

### 3.3.2 Descripción general del sistema

En este apartado se expondrá una descripción general del funcionamiento de la aplicación, identificando las distintas fases del ciclo de vida de la misma. Se tratará en dos fases: En la primera fase se estudiará el ciclo de vida de la aplicación desde un punto de vista general. En la segunda fase se dedicará especial atención al ciclo de vida del juego cuando se ejecuta un nivel en particular.

### *3.3.2.1 Diagrama de flujo de funcionamiento general de la aplicación*

En la Figura 28 se muestra un diagrama de flujo de la aplicación:

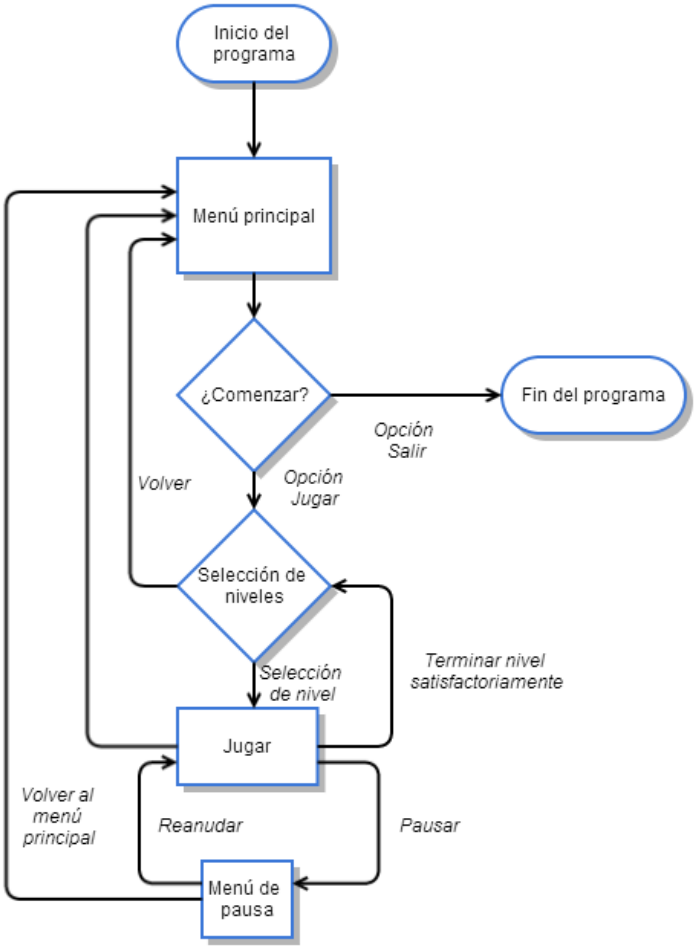


**Figura 28. Diagrama de flujo general de la aplicación**

Las distintas fases que componen este diagrama de flujo son las siguientes:

- **Inicio de programa:** El usuario ha decidido lanzar la aplicación.
- **Menú principal:** Como se muestra en la Figura 29, Se muestra el menú principal de la aplicación y se le ofrecen al usuario dos opciones: (1) finalizar la ejecución de la aplicación; y (2) avanzar al menú de selección de niveles.


Figura 29. Aspecto visual del Menú Principal

- **Selección de nivel:** Se muestra el menú de selección de niveles de forma que el usuario pueda seleccionar el nivel de juego y comenzar a jugar, como se muestra en la Figura 30. Además el usuario puede volver al menú principal.


Figura 30. Aspecto visual del Menú de Selección de Niveles

- **Jugar:** Comienza el juego en el nivel seleccionado por el usuario. En la Figura 31 se presenta el aspecto visual de uno de los niveles del juego. En este estado se pueden dar tres formas de finalización del nivel: La primera se produce cuando el usuario pierde todas las vidas. Esta nos lleva directamente al menú principal. La segunda se lanza cuando terminamos el nivel en cuestión satisfactoriamente. Esto produce una carga de la escena de selección de niveles permitiéndonos jugar

niveles nuevos en el caso de que quedara alguno por desbloquear. La tercera y última es pasar al menú de pausa.


Figura 31. Aspecto visual de la aplicación en la fase de juego

- **Menú de pausa:** En la Figura 32 se muestra el menú de pausa. Siempre que el usuario permanezca en este estado del flujo de ejecución, la aplicación el juego quedará "congelada" hasta que el jugador decida reanudar la partida (volviendo así al estado Jugar) o bien volver al menú principal (perdiendo todos los progresos realizados en la partida y mostrando a continuación dicho menú).


Figura 32. Aspecto visual del Menú de pausa

#### *3.3.2.2 Diagrama de flujo de funcionamiento en la fase de juego*

En la Figura 33 se muestra un diagrama de flujo de la fase de juego, correspondiente al nodo Jugar del diagrama de la Figura 28:

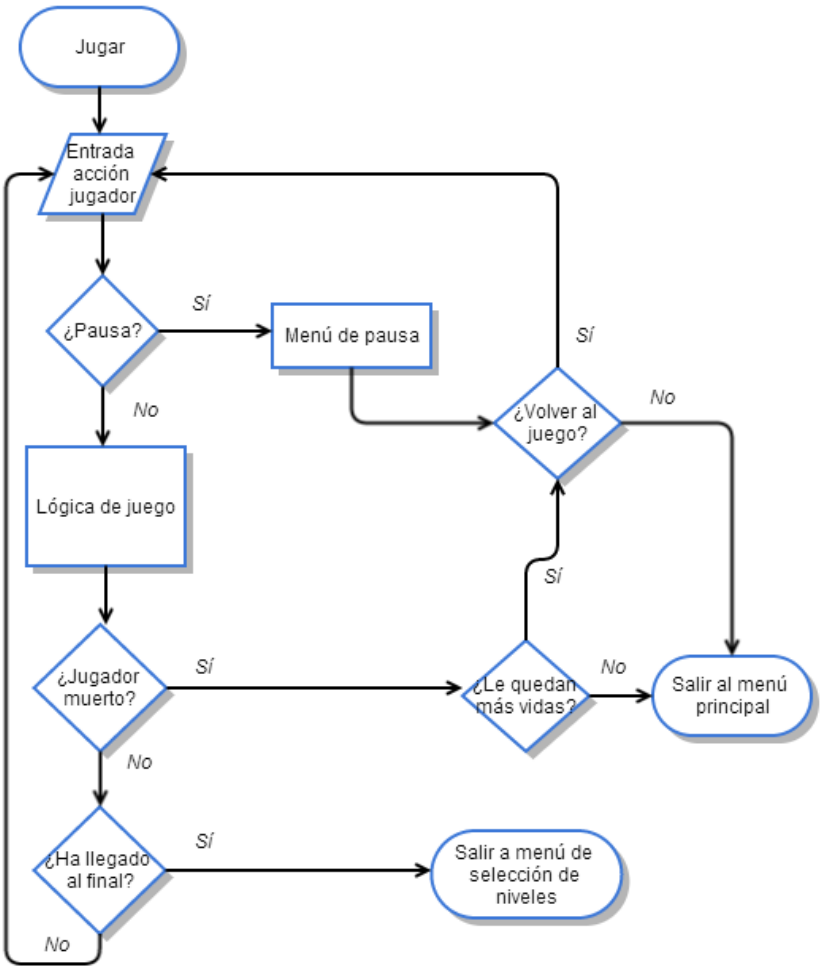


**Figura 33. Diagrama de flujo en la fase de juego**

Las distintas fases que componen este diagrama son las siguientes:

- **Jugar.** El jugador ha iniciado una partida tras cargar un nivel desde la pantalla de selección de niveles.
- **Entrada acción jugador.** Serie de entradas que el jugador puede usar para interactuar con el juego (acelerómetros, pantalla).

- **Menú de Pausa.** Si el jugador ha pulsado el botón de pausa, se volverá al menú de pausa explicado en el diagrama de flujo de funcionamiento general. Una de las opciones, aparte de salir al menú principal tal y como se dijo en el apartado anterior, es la de reanudar la partida, con lo que la aplicación dejará de estar en pausa y el jugador recuperará el control del personaje.
- **Lógica de juego.** Este estado está compuesto de todas aquellas cosas que interactúan con el personaje, estableciendo un compendio de condiciones y situaciones que harán que el jugador tenga que superarlas para avanzar y, en definitiva, poder acabar el juego. Una de las características que determinan el estado en el flujo de ejecución de la aplicación es el número de vidas restantes del personaje. Si el personaje muere y le quedan vidas disponibles, el jugador tiene la posibilidad de seguir jugando o salir al menú principal, como se muestra en la Figura 34. Si por el contrario al perder una vida no quedan vidas restantes, la única opción para el usuario será volver al menú principal, como se muestra en la Figura 35. Finalmente si el jugador consigue llegar al final del nivel sin perder todas las vidas, esto produce una carga de la escena de selección de niveles permitiéndonos jugar niveles nuevos en el caso de que quedara alguno por desbloquear.


Figura 34. Aspecto visual del menú de muerte


Figura 35. Aspecto visual del menú de fin de juego

### 3.3.3 Descripción de componentes

En la Figura 36 se muestra un diagrama de componentes que describe la estructura de la aplicación. A continuación se realiza una descripción de cada uno de los componentes, describiendo el conjunto de clases y métodos involucrados.

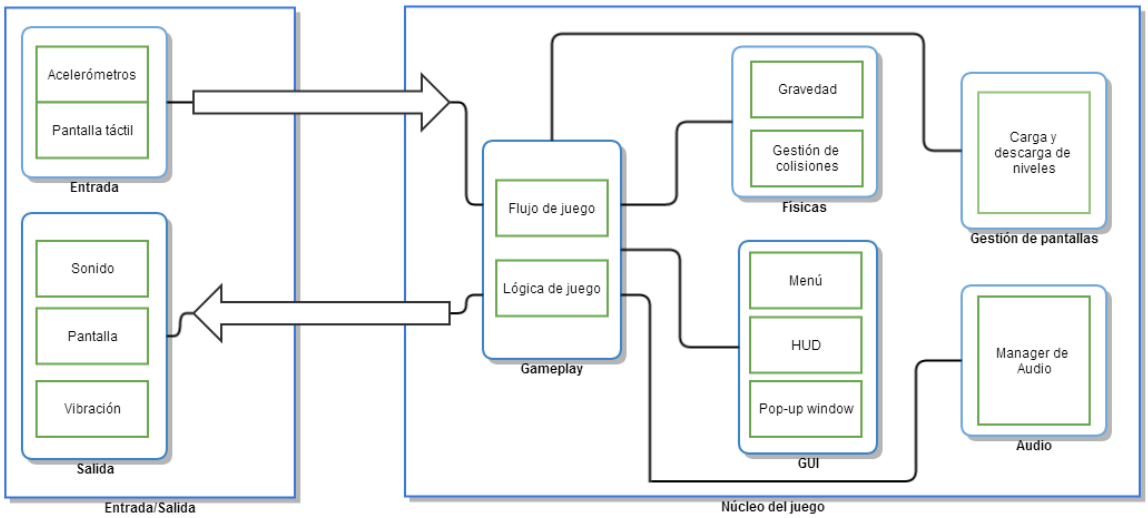


Figura 36. Diagrama de componentes

#### *3.3.3.1 Componente Entrada/Salida*

Este componente está formado por dos módulos que gestionan la interacción de la aplicación con los diferentes dispositivos de entrada y salida ofrecidos por la aplicación.

- **Módulo de Entrada.** Este componente se encarga de gestionar la información introducida por el usuario a la aplicación, es decir, se encarga de gestionar la interfaz de entrada del juego mediante el uso de dos dispositivos:
  - **Acelerómetros:** La rotación del terminal respecto de las coordenadas naturales hará que el personaje se mueva. Se le aplica al personaje una fuerza perpendicular a la dirección de la gravedad con módulo dependiente de la diferencia de ángulo con el estado de reposo, siendo dicha fuerza con módulo máximo de 1. En la Figura 37 se describe cómo se obtiene la información respecto de la posición del teléfono para realizar el movimiento del personaje.

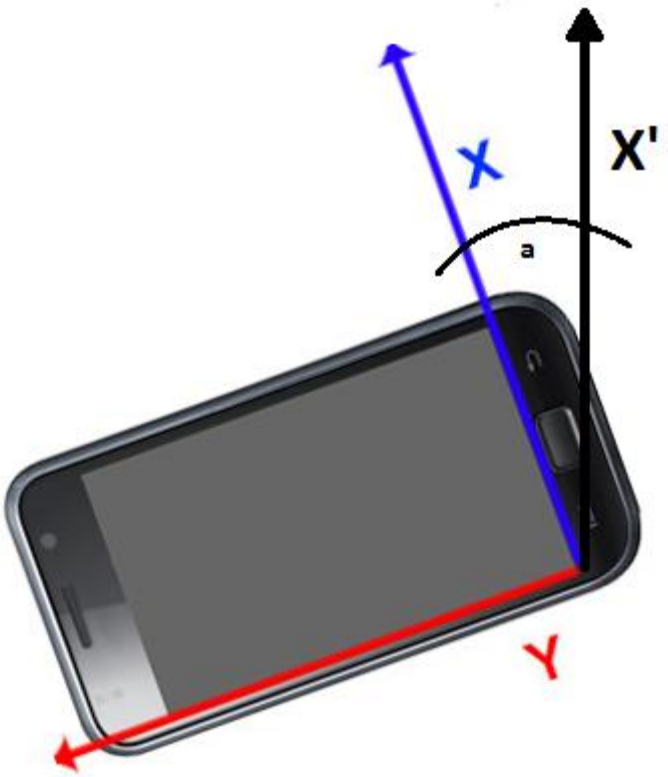


**Figura 37. Movimiento del personaje dependiente de acelerómetros**

Si suponemos X' como el vector dirección perpendicular a la superficie terrestre, y X como el vector dirección de la base de un terminal móvil (como se muestra en la Figura 37), la fuerza aplicada vendría dada por el ángulo *a*, dependiente de la diferencia de los vectores X y X'. Se tendrá acceso al valor normalizado de X, que dependiendo de su valor y su signo (inclinación en el sentido de las agujas del reloj tomado como positivo).Para poner un ejemplo, entendiendo como X' el valor normalizado 0, y X =-0.2 (que se calcula como se ha indicado en función de *a,* siendo *a=-18* °), se aplicará al personaje una fuerza con módulo 0.2 y dirección Y, con sentido negativo. Cabe decir que *Unity* nos proporciona el valor de los acelerómetros ya normalizados según los ejes X, Y, Z.

- **Pantalla táctil:** Todo el proceso de navegación por los diferentes menús de la aplicación, así como algunas funcionalidades del juego, como los saltos del personaje son realizados a través de la pantalla táctil del dispositivo. Se tendrá acceso a la posición en el plano XY de la pantalla donde se toque con el dedo y dependiendo de ciertas condiciones, se dispararán ciertos eventos, como por ejemplo la pulsación de un botón de menú o saltar con el personaje.

- **Módulo de Salida:** Este componente se encarga de gestionar la interacción de la aplicación con los diferentes dispositivos de salida. Hay tres tipos de dispositivos disponibles:
  - **Altavoz:** A modo de enriquecimiento, existirán dos melodías de fondo durante el juego (una para los menús y otra durante los niveles del juego). Adicionalmente existe un sonido para avisar al usuario que ha recogido una manzana satisfactoriamente.
  - **Pantalla:** Todo lo que suceda en el juego se representará gráficamente a través de la pantalla del dispositivo.

- **Vibración:** En el mismo instante en el que el personaje muera, una de las maneras que tiene el usuario de saber que el personaje ha muerto es mediante la vibración del móvil.

#### *3.3.3.2 Componente de núcleo del juego*

- **Gestión de pantallas:** Este módulo se encarga de la gestión de las distintas pantallas que se visualizarán durante la ejecución, así como la carga y descarga de las escenas que componen el juego.
  - **Carga y descarga de niveles.** Bajo ciertas condiciones, se harán llamadas a la aplicación correspondientes a la carga de escenas. Será el sistema de *GUI* encargado de hacer las llamadas correspondientes de carga de niveles a través de un "*Gameplay Manager"* programado entre otras cosas para tal fin.

- **Audio.** Este componente centraliza la gestión de los audios del juego.
  - **Manager de Audio.** Se ha desarrollado un *script* que centralice todas las llamadas de cambio de melodía en función de la escena que se cargue. Dichas llamadas, al igual que la carga de niveles, las hará el sistema de *GUI*.

- **Físicas.** Se ha utilizado el motor de físicas *PhysX* integrado en *Unity* para la gestión de colisiones y la aplicación de fuerzas. Así mismo, debido a la peculiar gravedad deseada que se explicó en la fase de diseño (3.2.1 Descripción de las características funcionales), dicha gravedad ha tenido que ser programada explícitamente en vez de poder utilizar la propia de *PhysX*.
  - **Gravedad.** La gravedad en el juego es poco convencional: Cada nivel está conformado por un mundo circular en forma de anillo, y el personaje se verá repelido por el centro de éste (Podemos decir tal y como se ve en la Figura 38

que el vector de dirección de la gravedad está compuesto por el centro del anillo como origen y el personaje como punto de destino).

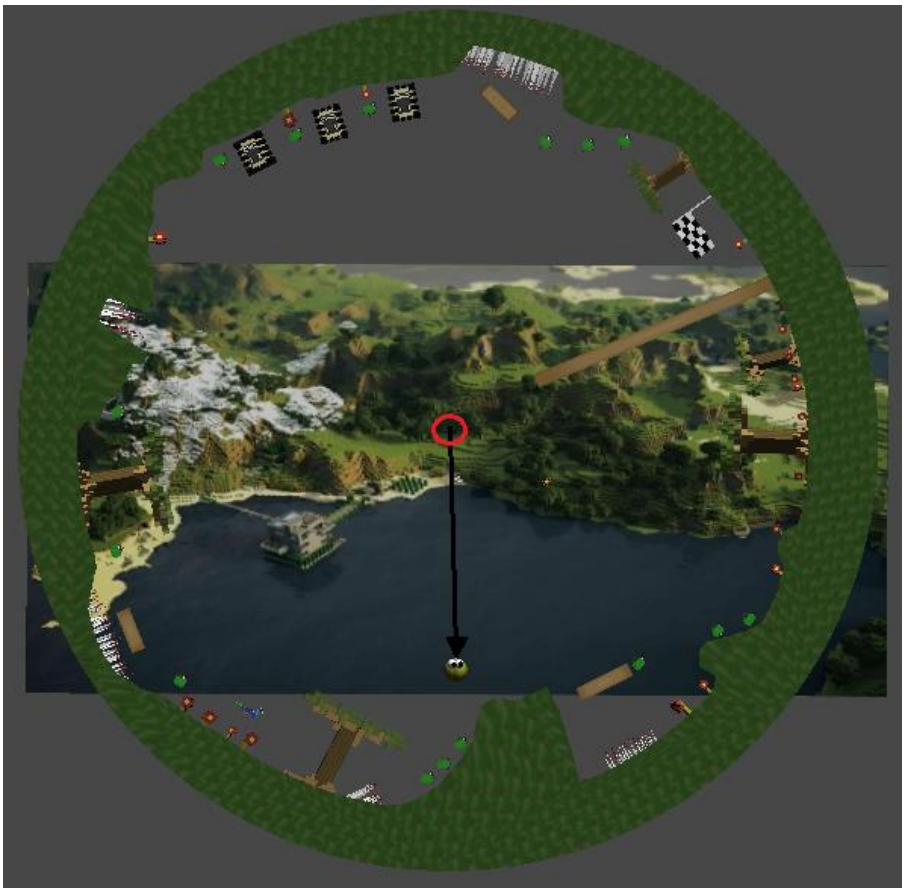

Figura 38. Gravedad

Como se ha dicho, el vector dirección de la gravedad está conformado por el origen del nivel (círculo rojo) y el personaje. Cabe decir que se ha utilizado la gravedad terrestre; esto es, una aceleración de 9.81 $m/s^2$.

- **Gestión de colisiones.** Se ha utilizado el sistema de gestión de colisiones del motor de físicas como manejador de eventos. A partir de estos eventos y mediante un sistema de *Tags,* se ha podido controlar eventos de físicas a partir de este sistema. Básicamente cuando un objeto que tiene un componente *Collider* detecta otro objeto con un componente *RigidBody,* podemos comprobar si el objeto que tiene *RigidBody* además contiene un *Tag* determinado. Por ejemplo supongamos que los pinchos tienen un *Collider.*

Cuando el personaje principal (Que tiene el componente *RigidBody*) colisiona con los pinchos, podemos comprobar si el objeto que ha entrado en contacto con ellos lleva el *Tag* "Player". Si es así, podemos enviar al *manager* que gestiona las vidas del personaje que el personaje ha muerto y lanzar la lógica de muerte.

- ***Gameplay*.** Este sistema se encarga de todo aquello relacionado con lo que está sucediendo durante el juego en un nivel. Es un componente de entrada y salida del resto de componentes pertenecientes al módulo de núcleo de juego, que procesa toda la información y actualiza tanto el flujo como la lógica del juego cuando procede. En definitiva cada funcionalidad gestiona y comunica los eventos que suceden en el juego al sistema de *Gameplay,* y éste a su vez toma las decisiones en cuanto a estados del flujo y lógica del juego.
  - **Flujo del juego.** Como ya hemos hablado en otras partes de este proyecto, el control de flujo del juego consiste en la gestión y ejecución del ciclo de vida de la aplicación. Este componente está centralizado en los controladores de *GUI* y el manager de *Gameplay*.
  - **Lógica del juego.** Engloba a todo lo relacionado con disparar eventos de físicas, audio, *GUI*… En definitiva, todo aquello que pueda ocurrir en el juego durante la ejecución de un nivel. La lógica del juego engloba a gran cantidad de entidades de juego, por lo que es necesario en esta fase del proyecto hablar de cada una de ellas.

#### *3.3.3.3 Entidades de la lógica de juego*

En este apartado veremos todas las entidades que por alguna razón participen en la lógica del juego, ya sea de modo indirecto (afectan a la estética y al acabado) o directo (afectan a la usabilidad del juego).

- **Entidades de la lógica de juego indirectas.** Tienen que ver con aspectos estéticos o de acabado del juego, que si bien no son críticas en funcionalidad, sí que representan el enriquecimiento y perfeccionamiento de la aplicación.
  - **Flores.** Las flores en este juego, mostradas en la Figura 39, son un elemento meramente decorativo, que tiene la peculiaridad: Sus pétalos giran en el sentido de las agujas del reloj o en el sentido contrario, dependiendo de lo que se decida en la ejecución del nivel de forma aleatoria.

**Figura 39. Personaje rodeado por flores girando**

  - **Árboles.** Los árboles mostrados en la Figura 40, son elementos estáticos que están esparcidos por la escena. Son elementos decorativos que le dan un toque natural al juego.

Figura 40. Árboles

- **Fondo.** Esta textura bidimensional le da sensación de profundidad al juego. Este elemento, aunque sea meramente estético, tiene una relativa alta importancia ya que ayuda al jugador a tener sensación de movimiento. Esta textura está mapeada en un plano que es hijo de la cámara y por lo tanto dará la sensación de que es el mundo el que gira.

- **Entidades de lógica de juego directas.** Estas entidades, aparte de poder tener también una finalidad estética como las entidades de lógica de juego indirectas, tienen una repercusión directa con la jugabilidad.
  - **Cámara.** Este elemento es primordial en el juego, ya que es el que representa gráficamente los elementos que están actuando y tienen malla texturizada en el juego. Se optó por hacer que la cámara fuera un objeto independiente y que tuviera acceso a la posición en el espacio del personaje y lo siguiera en el plano X-Y de forma suave, orientado según la gravedad aplicada al mismo:

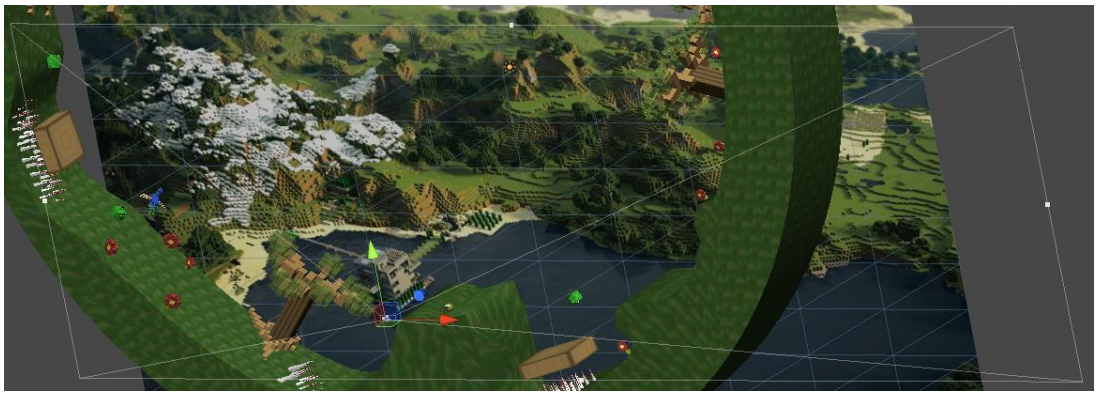

Figura 41. Posición de la cámara relativa al personaje

- **Pájaro.** Es uno de los enemigos del juego. Este enemigo vuela entre dos o más puntos del escenario de manera secuencial. Cuando el personaje está dentro de su radio de acción, como vemos en la Figura 42, el pájaro soltará huevos que caerán según la gravedad del mundo. Si el personaje es tocado por uno de estos huevos, pierde una vida.

Figura 42. Pájaro

- **Serpiente.** Este enemigo se encuentra fijo en el escenario. Cuando el personaje está dentro de su rango de acción, como se ve en la Figura 43, la serpiente escupe veneno cada cierta frecuencia (típicamente cada dos segundos) en la

dirección en la que se encuentre el personaje. Si el personaje es tocado por el veneno, pierde una vida.

Figura 43. Serpiente

- **Machacador.** Este último enemigo representado en la Figura 44 se mueve entre dos puntos de forma vertical obstaculizando el avance del personaje. Si el Machacador toca al personaje, éste pierde una vida.

Figura 44. Machacador

- **Bandera.** Este elemento, que se muestra en la Figura 45, es indicador de que el usuario ha llegado al final del nivel. Si se sobrepasa este elemento con el

personaje, el juego da por finalizado el nivel y carga el menú de selección de niveles.

Figura 45. Bandera

- **Manzana.** Este elemento sirve para que el jugador suba su puntuación durante el juego. Cuando el personaje entra en contacto con una, ésta desaparece de forma suave, y el contador de puntos sube (típicamente 10 puntos). Adicionalmente y como efecto estético, las manzanas darán vueltas sobre sí mismas en torno a su eje vertical.
- **Pinchos.** Estos elementos mostrados en la Figura 46 distribuidos de forma estática en el escenario tienen la simple misión de matar al personaje si éste entra en contacto con ellos.

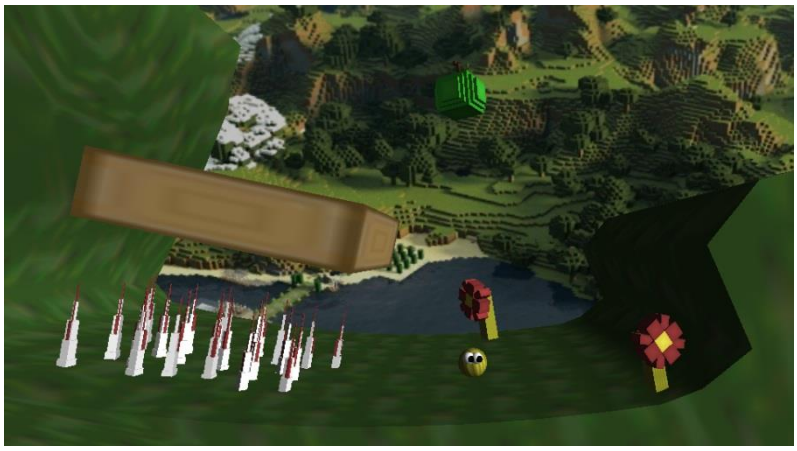

Figura 46. Pinchos y Plataforma

- **Plataformas.** Existen plataformas de dos tipos, estáticas y dinámicas. Estas segundas se moverán alternativamente entre dos puntos, como la de la Figura 46. El usuario podrá usarlas para evitar obstáculos presentes en el juego.
- **Bolas de pinchos.** Estos elementos estáticos que se representan en la Figura 47 que giran sobre sí mismos tienen como misión matar al personaje si éste entra en contacto con ellos.

Figura 47. Bolas de pinchos

- **Punto de *Spawn*.** Es el punto inicial de cada nivel y el punto donde el personaje reaparece cada vez que pierde una vida, como se ve en la Figura 48.

Figura 48. Punto de *Spawn*

- **Marcador de puntos.** Como se muestra en la Figura 49, es un elemento fijo en el *GUI* en la parte superior derecha (1) de la pantalla. Indica al jugador la puntuación acumulada en todo momento durante el nivel actual. Cada vez que se recoja una manzana, como hemos explicado en partes anteriores del proyecto, este marcador subirá.


Figura 49. Componentes de GUI *in-game*

- **Marcador de vidas.** Este marcador mostrado en la Figura 49, situado en la parte inferior derecha de la pantalla (2), nos indica con hojas de árbol el número de vidas restantes. En cada nivel el jugador empieza con 3 vidas restantes. Cada vez que el personaje pierda una vida, el elemento que lo mate enviará al Manager de *gameplay* un aviso de muerte. El personaje será enviado al punto de *spawn* y en el marcador de vidas, notificado por el Manager de *gameplay*, se restará una de las vidas, desapareciendo una de las hojas. Si en el momento de morir sólo quedaba una vida, al notificarse que ya no quedan más vidas se desencadena el proceso de fin de juego.
- **Botón de pausa.** Este botón, como se muestra en la Figura 49, situado en la esquina inferior izquierda de la pantalla (3), al ser pulsado por el usuario se parará el juego y se entrará en estado de pausa, abriendo una ventana en la que se podrá continuar el juego o salir al menú principal.

# Capítulo 4: Experimentación

En este capítulo se describe el conjunto de experimentos que han sido realizados sobre la aplicación desarrollada en este proyecto. Estos experimentos han sido definidos para analizar los requisitos de la aplicación establecidos inicialmente en este documento. Dichos experimentos consisten en un conjunto de pruebas que cada usuario tiene que realizar, para posteriormente responder a un cuestionario en el que se realizan varias preguntas relacionadas con el resultado de las pruebas. Los objetivos principales de la experimentación son:

- Comprobar que el flujo de la aplicación no contiene fallos.
- Observar el grado de fiabilidad de la aplicación respecto al diseño de la misma.
- Conocer las opiniones de los usuarios acerca de la jugabilidad y usabilidad.
- Recoger sugerencias de los usuarios para que puedan ser aplicadas en trabajos futuros.

## 4.1 Formulario de pruebas

En el siguiente formulario se han definido un conjunto de tareas con el fin de comprobar el correcto funcionamiento de la aplicación, el nivel de usabilidad, así como el grado de estabilidad de la misma. Para la realización de todas las pruebas se puso a disposición de los usuarios un terminal móvil BQ Aquaris 5 HD con la aplicación instalada.

Para evaluar los aspectos mencionados, se le plantean al usuario las siguientes tareas:

**Tarea 1**

El usuario debe lanzar la aplicación del juego *GyroWorld* pulsando el icono. ¿Ha comenzado el juego de manera satisfactoria?

☐ Sí

☐ No

Si la respuesta es no, indique qué ha pasado

**Tarea 2**

El usuario debe salir de la aplicación del juego *GyroWorld* desde el menú. ¿Ha salido correctamente?

☐ Sí

☐ No

Si la respuesta es no, indique qué ha pasado

**Tarea 3**

El usuario debe ser capaz de comenzar el primer nivel. ¿Ha sido capaz?

☐ Sí

☐ No

Si la respuesta es no, indique qué ha pasado

**Tarea 4**

Durante el juego, se debe poder hacer pausa. ¿Ha pausado el juego correctamente?

☐ Sí

☐ No

Si la respuesta es no, indique qué ha pasado

**Tarea 5**

Tras hacer una pausa, ¿ha podido continuar jugando?

☐ Sí

☐ No

Si la respuesta es no, indique qué ha pasado

**Tarea 6**

Tras hacer una pausa, ¿ha podido salir al menú principal?

☐ Sí

☐ No

Si la respuesta es no, indique qué ha pasado

**Tarea 7**

Tras probar el movimiento, el salto y el doble salto, ¿cree que funcionan correctamente?

☐ Sí

☐ No

Si la respuesta es no, indique por qué

**Tarea 8**

Al perder todas las vidas, ¿termina la partida y se le pide volver al menú principal?

- ☐ Sí
- ☐ No

Si la respuesta es no, indique por qué

**Tarea 9**

Cada manzana recogida suma 10 puntos al marcador de cada nivel, ¿sube el marcador según lo previsto?

- ☐ Sí
- ☐ No

Si la respuesta es no, indique por qué

**Tarea 10**

¿Ha conseguido terminar algún nivel?

- ☐ Sí
- ☐ No

Si la respuesta es no, indique por qué

**Tarea 11**

¿Le ha parecido el juego demasiado complicado?

- ☐ Fácil
- ☐ Complicado

Si la respuesta es Complicado, indique por qué

**Tarea 12**

¿Le ha gustado el juego?

- ☐ Sí
- ☐ No

**Tarea 13**

¿Qué aspectos mejoraría del juego?

## 4.2 Resultados del cuestionario

En este apartado se muestra el resultado obtenido de los cuestionarios entregados a los usuarios escogidos para la evaluación del videojuego. Los usuarios 1, 2 y 6 son jugadores habituales de videojuegos, los usuarios 3 y 4 son jugadores ocasionales de videojuegos y el usuario número 5 no es jugador de videojuegos pero sí utiliza un terminal móvil con frecuencia. Las sugerencias de la Tarea 13 se aglutinarán en el apartado de conclusiones a continuación de la tabla de resultados.

| Pregunta | Usuario 1 | Usuario 2 | Usuario 3 | Usuario 4 | Usuario 5 | Usuario 6 |
|---|---|---|---|---|---|---|
| **T1. ¿Ha comenzado el juego de manera satisfactoria?** | Sí | Sí | Sí | Sí | Sí | Sí |
| **T2. ¿Ha salido correctamente?** | Sí | Sí | Sí | Sí | Sí | Sí |
| **T3. ¿Ha sido capaz de comenzar el nivel 1?** | Sí | Sí | Sí | Sí | Sí | Sí |
| **T4. ¿Ha pausado el juego correctamente?** | Sí | Sí | Sí | Sí | Sí | Sí |
| **T5. ¿Ha podido continuar jugando?** | Sí | Sí | Sí | Sí | Sí | Sí |
| **T6. ¿Ha podido salir al menú principal?** | Sí | Sí | Sí | Sí | No | Sí |
| **T7. ¿Cree que los controles funcionan de forma correcta?** | No | Sí | Sí | Sí | No | Sí |
| **T8. ¿Termina la partida tras perder todas las vidas?** | Sí | Sí | Sí | Sí | Sí | Sí |
| **T9. ¿Sube el marcador según lo previsto?** | Sí | Sí | Sí | Sí | Sí | Sí |
| **T10. ¿Ha conseguido terminar algún nivel?** | Sí | Sí | Sí | Sí | No | Sí |
| **T11. ¿Le ha parecido el juego demasiado complicado?** | Sí | No | No | No | Sí | Sí |
| **T12. ¿Le ha gustado el juego?** | Sí | Sí | Sí | Sí | Sí | Sí |

Tabla 43. Resultados del cuestionario

## 4.3 Conclusiones de resultados y resumen de comentarios

Tras analizar los resultados del cuestionario, se han sacado las siguientes conclusiones:

- De forma general, el flujo de ejecución del juego funciona de manera robusta, pues no se han detectado fallos en cuanto a funcionamiento.
- Hay algunas discrepancias en cuanto a la jugabilidad; La mitad de los usuarios han encontrado el juego algo difícil, y en cuanto a los controles, en el caso particular del doble salto, dos de los seis usuarios han coincidido en que es complicado de dominar.
- A la pregunta de si ha gustado el juego, todos los usuarios lo han encontrado muy divertido, lo que cumple con uno de los pilares fundamentales de los videojuegos: Ha de ser entretenido.
- A excepción de un usuario, todos han sido capaces de superar uno o ambos niveles incluidos en la aplicación. De esto se concluye que la curva de aprendizaje está compensada con la longitud y complejidad de los niveles.
- Se ha recomendado en múltiples ocasiones asociar algunas funcionalidades del juego (sobre todo la del caso de volver) a los botones que nos ofrece la mayoría de terminales Android: alguno de los usuarios de forma intuitiva ha intentado usarlo en más de una vez.
- Otra de las sugerencias recogidas es la de poder matar a los enemigos que nos atacan, pues es algo que se considera también divertido y no rompe con la temática del juego.
- En general ha gustado mucho el diseño de niveles y la estética de los mismos.
- En cuanto a los controles, se tuvo que asistir al principio a un par de usuarios, especialmente con el doble salto. Tras unas breves indicaciones, los usuarios se adaptaron bien a las mecánicas de la aplicación.

# Capítulo 5: Gestión del proyecto

Cuando se estudia la viabilidad de un proyecto, la planificación previa para la realización del mismo es básica para que el desarrollo se efectúe de la forma más eficiente posible. La estructuración de las fases que componen el proyecto y la asignación de los recursos necesarios ayudan a cumplir con los plazos y funcionalidades previstos.

En este capítulo hablaremos del ciclo de vida que tiene el desarrollo de un juego comercial, y las distintas fases que lo componen. Tras describir cada una de las etapas del proyecto y el modelo elegido para producirlo, se procederá a describir la planificación final que se tuvo en cuenta para el desarrollo de *GyroWorld*. Finalmente se cerrará el capítulo con la presentación del presupuesto, calculado a partir de todos los elementos utilizados (*software*, *hardware*, personal, material fungible, etcétera).

## 5.1 Descripción de las fases del proyecto

En este apartado se describe el típico ciclo de vida de un videojuego comercial desde la concepción de la idea original hasta la última actualización añadida.

- **Diseño de concepto:** Es la fase inicial del ciclo de vida. En ella se define el juego a grandes rasgos, creando una propuesta con descripción de mecánicas de juego, ambientación, plataformas objetivo, viabilidad, etcétera.
- **Pre-producción:** En esta fase se definen las bases del resto del desarrollo. Se debe cerrar un documento de diseño con todas las especificaciones y funcionalidades necesarias acompañado de arte conceptual para generar una idea visual y funcional del juego. En paralelo en el apartado de programación se debe construir la arquitectura necesaria para que el desarrollo sea más fácil y esté bien estructurado. Es también común hacer varios prototipos para validar las mecánicas diseñadas y probar ideas.

- **Producción:** Es obviamente la fase más importante del proyecto, pues en ella se realiza la implementación de la aplicación.
- **Alfa:** En esta fase el flujo de ejecución de juego debe estar cerrado de principio a fin, con todas las funcionalidades desarrolladas, a falta de depurar detalles y errores por solucionar.
- **Beta:** Todos los elementos del juego están acabados a falta de testear en profundidad la aplicación para encontrar los últimos errores y pulirlos antes de dar la aplicación por cerrada.
- **Liberación:** El juego queda validado y se distribuye en el formato y la plataforma a los que vaya dirigido. Los usuarios finales por lo tanto podrán tener acceso a él.
- **Actualizaciones:** Es cada vez una práctica más común el uso de actualizaciones tras la liberación del producto tanto para arreglar errores surgidos con el juego ya en el mercado como para ofrecer nuevos contenidos descargables (éstos a su vez tendrán las distintas fases del ciclo de vida del desarrollo antes de liberarlos).

## 5.2 Planificación

La primera decisión para realizar la planificación consiste en decidir el modelo de ciclo de vida que se va a usar para el desarrollo de la aplicación. Dentro de los diferentes modelos tipo disponibles en este tipo de planificación (cascada, lineal, *Sashimi*, etcétera), se ha elegido el **ciclo de vida en espiral**.

Este modelo desde el principio se pensó que encajaba muy bien con la filosofía que se quería llevar para la elaboración de este proyecto, ya que ofrece gran flexibilidad para incluir nuevos requerimientos cuando no se tiene una experiencia muy amplia en diseño de videojuegos. Como se muestra en la Figura 50, este modelo se basa en la iteración del ciclo que contempla las fases de Planeación, Modelado, Construcción, Despliegue y Comunicación.

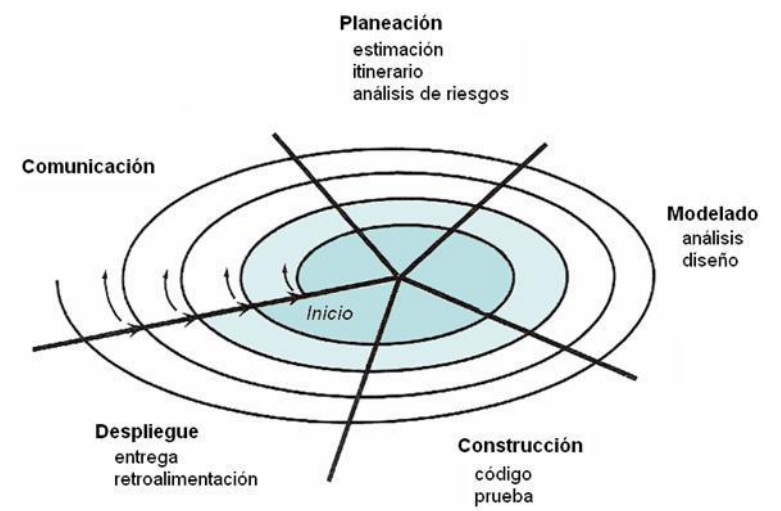


**Figura 50. Ciclo de modelo evolutivo**

En primer lugar se realiza una estimación de la tarea a realizar (**Planeación**) tanto en tiempo como en recursos, teniendo en cuenta los imprevistos que pueden surgir. La siguiente fase (**Modelado**) consta del análisis de aquello a desarrollar; se diseña a nivel conceptual todo aquel requerimiento que se quiera incluir y pensar cómo abordar los problemas con previsión. En la tercera fase (**Construcción**) se desarrolla y prueba la tarea en cuestión. Finalizada la fase de desarrollo, se tiene una versión del juego intermedia lista para ser evaluada (**Despliegue**), tras la cual se pueden añadir requerimientos nuevos o revisar los actuales (**Comunicación**) con el fin de pulir y mejorar el producto de forma continua. Otro aspecto de la planificación es que si se combina este ciclo de vida con un desarrollo bien planificado basado en objetivos intermedios, se consigue un método de desarrollo ágil (Como en el método *Scrum* [10]).

En una división basada en componentes o módulos del proyecto, se ha aplicado este modelo, para depurar los mismos por separado y finalmente obtener una versión definitiva.

Los módulos son:

- **Formación:** Consiste en la familiarización de los elementos necesarios para el desarrollo del proyecto, estudiando la plataforma objetivo y el motor de desarrollo con el que se va a trabajar.
- **Memoria y documentación:** A medida que se avanza en el desarrollo de la aplicación, es recomendable recopilar de forma paralela la documentación necesaria (diseño, pautas de programación, etcétera) para redactar el documento final.
- **Arquitectura del *software*:** En este componente se pretende iterar en las funcionalidades básicas del flujo de ejecución, tales como carga y descarga de escenas, manejo de eventos de pulsación, etcétera.
- **Interfaz gráfico:** Se diseñan las apariencias que se quieren tener de las distintas pantallas que están incluidas en el flujo del juego.
- **Núcleo del juego:** este módulo engloba gran parte del peso del proyecto. Gestión de físicas, enemigos y máquinas de estado en el juego son algunas de las tareas incluidas en este componente.
- **Evaluación y pruebas:** Una vez se tiene la versión final se pasa a una fase de validación para comprobar que funciona según lo deseado.

Los componentes mencionados se muestran en la tabla y en el diagrama de planificación (Tabla 44), donde se indican los recursos y tiempos que se han estimado para la realización de este proyecto.

**Diagrama de planificación**

| ETAPAS DEL PROYECTO | DURACIÓN ESTIMADA DE TAREAS | FECHA DE INICIO ESTIMADA DE TAREAS |
|---|---|---|
| **1. Formación** | **20 Días** | **23/12/2013** |
| 1.1 Estudio *Unity* | 15 días | 23/12/2013 |
| 1.2 Estudio *Android* para *Unity* | 5 días | 7/1/2014 |
| **2. Memoria y documentación** | **160 días** | **23/12/2013** |
| **3. Entorno de desarrollo** | **11 días** | **23/12/2013** |
| 3.1 Diseño | 1 día | 23/12/2013 |
| 3.2 Reconocimiento del sistema | 2 días | 24/12/2013 |
| 3.3 Modo de ejecución según sistema | 7 días | 26/12/2013 |
| 3.4 Evaluación | 1 día | 2/1/2014 |
| **4. Entrada** | **16 días** | **3/1/2014** |
| 4.1 Diseño | 2 días | 3/1/2014 |
| 4.2 Sistema de entrada para *PC* (pruebas) | 4 días | 5/1/2014 |
| 4.3 Sistema de entrada para *Android*: Acelerómetros | 3 días | 9/1/2014 |
| 4.4 Sistema de entrada para *Android*: Pantalla táctil | 6 días | 12/1/2014 |
| 4.5 Evaluación | 1 día | 18/1/2014 |
| **5. Físicas** | **16 días** | **19/1/2014** |
| 5.1 Diseño | 1 día | 19/1/2014 |
| 5.2 Gravedad del mundo | 9 días | 20/1/2014 |
| 5.3 Gestión de colisiones | 4 días | 29/1/2014 |
| 5.4 Evaluación | 2 días | 2/2/2014 |
| **6. Gestión de pantallas** | **7 días** | **4/2/2014** |
| 6.1 Diseño | 1 día | 4/2/2014 |
| 6.2 Carga de niveles | 5 días | 5/2/2014 |
| 6.3 Evaluación | 1 día | 10/2/2014 |
| **7. *GUI*** | **23 días** | **11/2/2014** |
| 7.1 Diseño | 2 días | 11/2/2014 |

| | | |
|---|---|---|
| 7.2 Menú principal | 4 días | 13/2/2014 |
| 7.3 Menú de selección de niveles | 3 días | 17/2/2014 |
| 7.4 HUD: Puntuación | 3 días | 20/2/2014 |
| 7.5 HUD: Vidas | 4 días | 23/2/2014 |
| 7.6 HUD: Pausa | 2 días | 27/2/2014 |
| 7.7 HUD: *"Pop-ups"* | 4 días | 1/3/2014 |
| 7.8 Evaluación | 1 día | 5/3/2014 |
| **8. Lógica de juego** | **37 días** | **6/3/2014** |
| 8.1 Diseño | 4 días | 6/3/2014 |
| 8.2 Entidades indirectas | 5 días | 10/3/2014 |
| 8.3 Entidades directas | 21 días | 15/3/2014 |
| 8.4 Niveles | 5 días | 5/4/2014 |
| 8.5 Evaluación | 2 días | 10/4/2014 |
| **9. Flujo de juego** | **9 días** | **12/4/2014** |
| 9.1 Diseño | 1 día | 12/4/2014 |
| 9.2 Arquitectura de flujo del juego | 7 días | 13/4/2014 |
| 9.3 Evaluación | 1 día | 20/4/2014 |
| **10. Audio** | **3 días** | **21/4/2014** |
| 10.1 Diseño | 1 día | 21/4/2014 |
| 10.2 Manager de audio | 1 día | 22/4/2014 |
| 10.3 Evaluación | 1 día | 23/4/2014 |
| **11. Otros componentes** | **15 días** | **24/4/2014** |
| 11.1 Diseño | 3 días | 24/4/2014 |
| 11.2 Vibración | 1 día | 27/4/2014 |
| 11.3 Estética | 6 días | 28/4/2014 |
| 11.4 Ajuste de componentes | 3 días | 4/5/2014 |
| 11.5 Evaluación | 2 días | 7/5/2014 |
| 12. Evaluación y pruebas | 6 días | 9/5/2014 |

Tabla 44. Planificación temporal del proyecto

A continuación en las Figuras 51 y 52 se muestran el diagrama de Gantt correspondiente a las estimaciones de las tareas y a su duración real; las estimaciones figuran con barras de colores y las

duraciones reales de las tareas se agrupan con barras en negro. Todas las tareas enumeradas en la tabla anterior requieren de recursos materiales y/o personales. En los módulos o componentes donde se encuentran disponibles más de un recurso, es posible la realización de tareas en paralelo, optimizando el tiempo de desarrollo del proyecto.

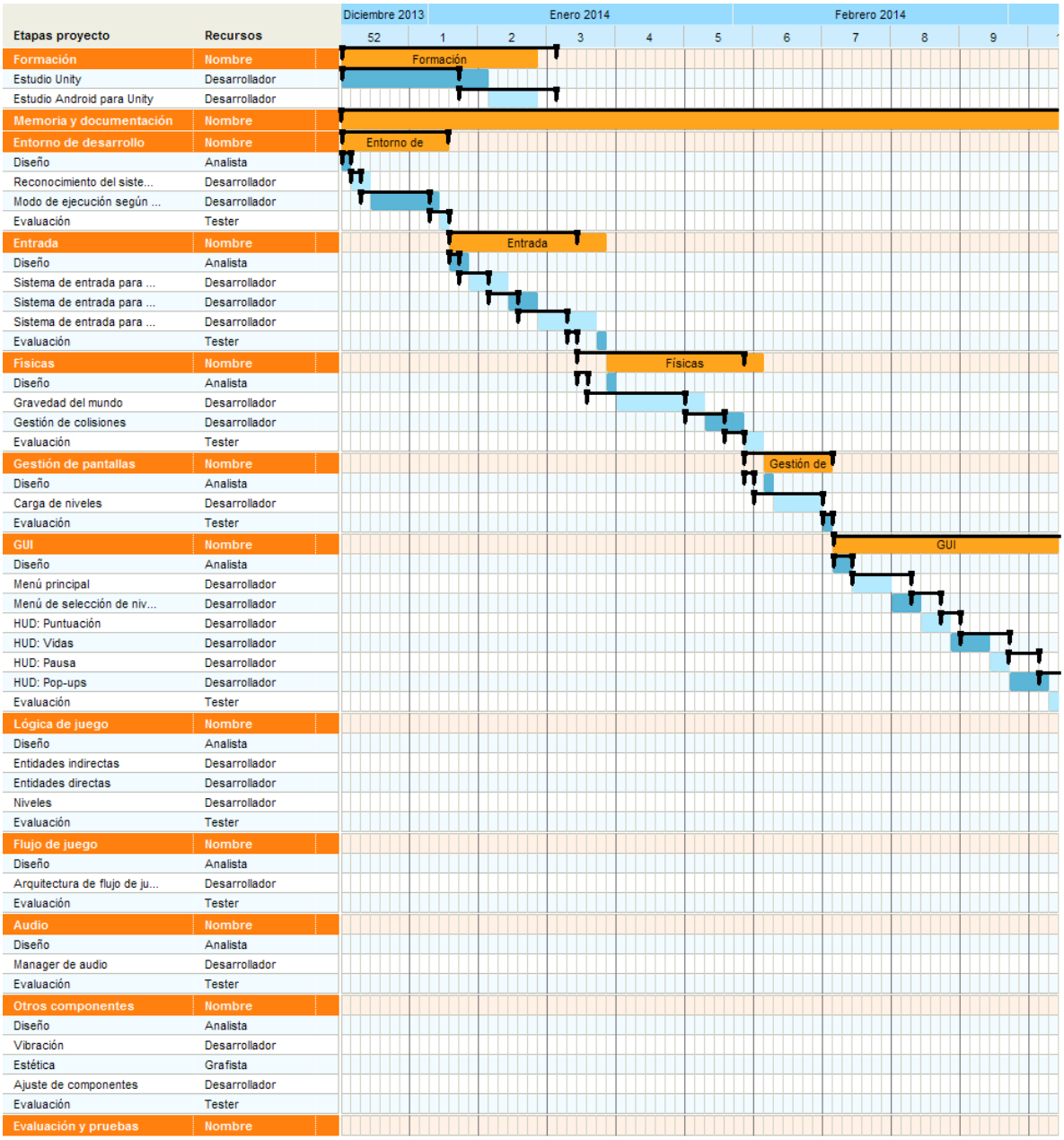


**Figura 51. Diagrama de Gantt (parte I)**

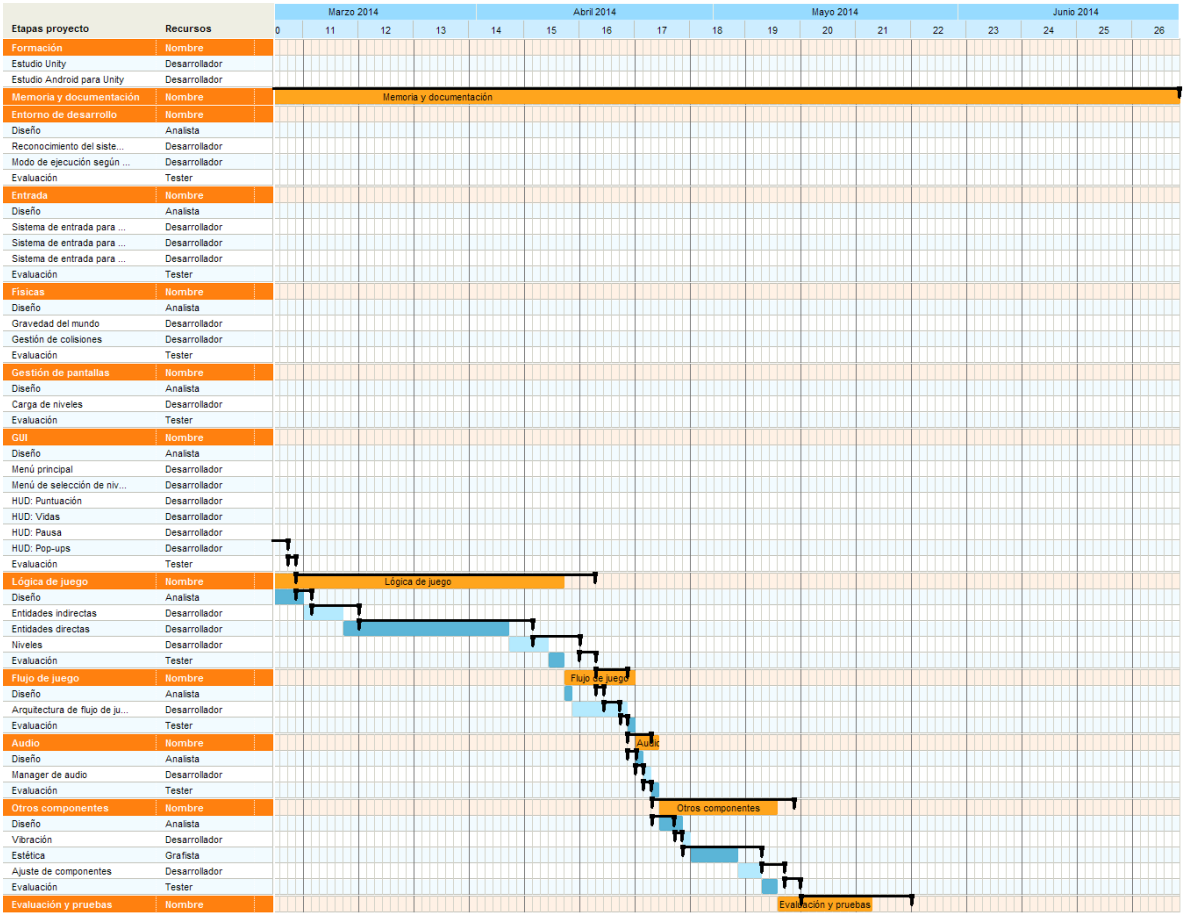


**Figura 52. Diagrama de Gantt (parte II)**

## 5.3 Presupuesto

En este apartado se llevará a cabo el desglose del presupuesto. Personal, Seguridad Social, medios materiales y amortización así como costo total del trabajo se verán reflejados en este apartado. Los salarios se han obtenido del Boletín Oficial del Estado según grupos profesionales y nivel de funciones.

## PRESUPUESTO DEL PROYECTO

**1.- Autor:**

Miguel Ángel Márquez Martínez

**2.- Departamento:**

Telecomunicaciones

**3.- Descripción del proyecto:**

- Título: **Diseño e implementación de un juego para smartphones con Android: GyroWorld**

- Duración (días): 210

- IVA: 15%

**4.- Presupuesto total del Proyecto (valores en Euros):**

**29.410,77 € (IVA incluido)**

**5.- Desglose presupuestario (costes directos):**

**PERSONAL**

| Perfil | Precio/Año (euros) | Número meses | Coste Final (euros) |
|---|---|---|---|
| **Analista programador G3/Nivel 2** | 21.800,00 | 7 | 12.716,67 |
| **Programador G2/Nivel 2** | 18.100,00 | 7 | 10.558,33 |
| **Tester G1/Nivel 2** | 13.000,00 | 2 | 2.166,67 |
| | | **Total (€)** | 25.441,67 |

Tabla 45. Costes de personal

## EQUIPOS

| Descripción | Coste (euros) | % Uso dedicado | Dedicación (meses) | Periodo de deprecación (días) | Coste imputable[a] (euros) |
|---|---|---|---|---|---|
| **Ordenador** | 714,00 | 100 | 7 | 60 | 83,30 |
| **BQ Aquaris 5 HD** | 167,56 | 100 | 7 | 60 | 19,55 |
| | | | | **Total (€)** | 102,85 |

Tabla 46. Costes de equipos

[a]Fórmula de cálculo de la amortización

$$\frac{A}{B} \times C \times D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = Periodo de deprecación (60 meses)

C = Coste del equipo (sin IVA)

D = % de uso dedicado a proyecto (100% normalmente)

## COSTES INDIRECTOS DEL PROYECTO

| Descripción | Empresa | Coste computable[b] (euros) |
|---|---|---|
| **Viajes** | | 50,00 |
| | **Total (€)** | 50,00 |

Tabla 47. Costes indirectos del proyecto

[b]Incluidos todos los gastos no indicados anteriormente: Consumibles, viajes, dietas, etc.

## 6.- Resumen presupuestario:

| Concepto costes totales | Presupuesto costes totales (euros) |
|---|---|
| **Personal** | 25.441,67 |
| **Amortización** | 102,85 |
| **Costes de funcionamiento** | 50,00 |
| **IVA** | 3.816,25 |
| **Total (€)** | 29.410,77 |

Tabla 48. Resumen presupuestario

# Capítulo 6: Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones obtenidas tras la realización de este proyecto. Además se incluyen aquellos posibles trabajos futuros que se podrían desarrollar sobre la aplicación.

## 6.1 Conclusiones generales

El objetivo principal marcado al principio del proyecto consistía en conocer y aprender todas las fases de desarrollo de un videojuego de plataformas para dispositivos móviles. Este objetivo se ha conseguido satisfactoriamente, ya que al tener tras finalizar este proyecto un juego de plataformas para móviles cerrado, con varios niveles, y con un flujo de juego y usabilidad coherentes, implica que se ha entendido cada una de las fases de desarrollo de este tipo de *software*. No obstante desarrollar videojuegos de cualquier tipo que puedan ser utilizados en diferentes plataformas es muy complicado. A lo largo del desarrollo de este proyecto se ha visto que el desarrollo de un videojuego implica una serie de dificultades y restricciones que son complicadas de definir a priori. Es vital realizar un estudio previo de las tecnologías que se quieren usar, tanto las tecnologías para el desarrollo como las plataformas objetivo, pues las conclusiones de este estudio definirán las “reglas del juego” para poder desarrollar la aplicación satisfactoriamente.

El diseño de la arquitectura del *software* y la previsión en el desarrollo a la larga ahorra una cantidad de tiempo muy valiosa. Además el desarrollo de herramientas para hacer entornos de pruebas ágiles aporta mucha eficiencia en los procesos de desarrollo del videojuego.

Hacer un videojuego con una curva de aprendizaje ajustada al usuario medio es complicado. En el periodo de evaluación en el que usuarios hicieron una serie de pruebas, se constató que el juego a priori era algo difícil, hecho que es complicado de ver por el desarrollador por estar demasiado acostumbrado a las mecánicas del juego. Con este

hecho queda demostrado que la figura de los *testers* es importante para definir la dificultad y así hacer el juego más divertido.

## 6.2 Conclusiones referentes a los objetivos

En base al objetivo principal, en el capítulo 1 se marcaron un conjunto de objetivos parciales. A continuación se presentan las conclusiones obtenidas para cada uno de ellos.

- **Estudio de *Unity***

  El estudio de este motor de videojuegos fue muy útil para poder desarrollar el videojuego posteriormente de manera fluida. No obstante gran parte del aprendizaje total obtenido vino por la implementación propia del videojuego. Debido a esto se comprendieron mucho mejor las capacidades y el funcionamiento del motor.

- **Lenguaje de programación *C#***

  Si bien es un lenguaje bastante amigable, el motor tiene sus propias clases que hacen que no sólo haga falta aprender *C#*, sino también la arquitectura de programación asociada a *Unity*. En cualquier caso se ha aprendido muchísimo de este lenguaje de cara a futuros desarrollos.

- **Diseño y documentación**

  Tanto a nivel de *software* como de usabilidad, el hacer documentación y diseño de forma previa al desarrollo ha sido un acierto mayúsculo. Como ya se ha explicado en otras ocasiones en este documento, pensar bien lo que se quiere y cómo se quiere en general ayuda a entender mejor cómo se deben hacer las cosas a todos los niveles.

- **Juego funcional**

Este objetivo ha sido la parte del proyecto más compleja, pues implica que el *software* funcione de forma correcta según los requisitos marcados al comienzo del desarrollo. No obstante, la dificultad de saber cuál es la mejor solución para cada problema planteado a lo largo del proyecto es lo que más me ha enriquecido. Se puede decir como conclusión que se ha conseguido desarrollar un sistema robusto y capaz de llevar a cabo la funcionalidad esperada.

- **Juego divertido**

La evaluación del proyecto ha permitido constatar que esto no es tan sencillo como parece. Es importante que haya gente involucrada en el proyecto que pruebe constantemente la aplicación para equilibrar de forma continua la usabilidad y hacer que el juego esté nivelado y sea divertido. Tras mucho empeño en este punto y ver a usuarios probar la aplicación, podemos decir que el juego desarrollado es divertido tras muchas iteraciones en su balanceo de dificultad y modificación de los niveles.

- **Código accesible**

Se ha intentado implementar código estructurado y de fácil comprensión para que en futuros usos del mismo, sea fácil interpretar el funcionamiento y que la inclusión de nuevas funcionalidades no sea una tarea traumática. Comentarios, regiones y pautas de escritura de código son algunas de las medidas que se han tomado para tal fin.

## 6.3 Trabajos futuros

Cuando se plantea un proyecto sobre el desarrollo de un videojuego, siempre se ha de establecer los límites para los cuales se dan por alcanzados los objetivos marcados. Hay que tener en cuenta que siempre existirán mejoras posibles en un videojuego, no obstante también hay que saber dar por cerrado un proyecto y ajustarse al tiempo estimado para la realización del mismo. No obstante, es importante resaltar aquellas funcionalidades que se han quedado en el tintero;

aquellas funcionalidades que podrían ser incluidas en futuros desarrollos y son muy interesantes tanto desde el punto de vista del desarrollo como desde el punto de vista del jugador.

### 6.3.1 Incremento de niveles y mundos temáticos

Actualmente existen dos niveles disponibles en el videojuego. Una opción interesante que se desarrolla en este tipo de juegos es tener mundos temáticos. Podemos decir que si bien los dos niveles desarrollados tienen una temática de "bosque", y tienen características y enemigos asociados a esta temática (El Pájaro, la Serpiente y el Machacador de piedra), sería muy interesante hacer otras temáticas para darle variedad al juego y enriquecer mucho su contenido. Unos ejemplos de temáticas pueden ser "hielo" (en el que se podrían hacer superficies deslizantes, Pingüinos como enemigos) o "lava" (cráteres de fuego que hacen daño al personaje, vapores de gas que le impulsan), y en cada uno de ellos un conjunto representativo de niveles. Además sería muy interesante hacer al final de cada mundo un nivel especial en el que el personaje se enfrente a un "Jefe" (enemigo más complicado de lo normal).

### 6.3.2 Puntuación en línea

En la actualidad es muy común publicar en redes sociales los hitos o resultados alcanzados en un videojuego. El poder publicar récords del juego tales como los puntos máximos que se han conseguido en un nivel o el tiempo mínimo que se ha utilizado para superarlo le daría un componente muy fuerte de re-jugabilidad y añadiría el componente social que muchos jugadores buscan cuando adquieren un videojuego.

### 6.3.3 Mejoras y personalización

Otro de los aspectos claramente favorecedores en este tipo de aplicaciones es la sensación de mejora del personaje. Si se almacenara de manera global todos los puntos que obtiene el usuario a lo largo de varias partidas, se podría hacer una pantalla de

"Tienda", accesible desde el menú principal donde pudiéramos comprar mejoras en la jugabilidad (salto más potente, aumento en el número de vidas máximas) o de personalización meramente estéticos (distintos colores y aspectos para el personaje). Estos elementos no son muy difíciles de implementar y enriquecen mucho la experiencia del jugador.

### 6.3.4 Otras mejoras

Además de las mejoras importantes mencionadas, también existe una serie de pequeñas mejoras que se podrían realizar:

- **Poder acabar con los enemigos**: Estaría bien que, si el personaje "pisa" a los enemigos, éstos desaparezcan (y podrían devolver alguna bonificación por ello, como una manzana o una vida).
- **Inteligencia de los enemigos**: Se podría implementar enemigos inteligentes que supieran reaccionar a los movimientos del usuario y lo atacaran acorde a los mismos.
- **Animación de muerte**: Otra posible pequeña mejora sería implementar algún tipo de animación cada vez que el personaje muere para que el usuario tenga mayor constancia de que ha perdido una vida.

# Capítulo 7: Anexos

## 7.1 Manual de instalación

Para instalar el *.apk GyroWorld* en un terminal con sistema operativo *Android*, hay que realizar los siguientes pasos:

1. **Transferir el *.apk* al terminal.** Es común guardarlo en una ruta fácil de recordar para posteriormente encontrarlo de manera sencilla en la ubicación donde se haya almacenado.
2. **Dar permisos de ejecución a aplicaciones de orígenes desconocidos.** Para indicar al sistema operativo *Android* del terminal que permita la instalación de programas sin utilizar la plataforma de *Google Play*, se debe recorrer la ruta "Ajustes -> Seguridad -> Orígenes desconocidos" y activar dicha opción como la que aparece en la Figura 53.

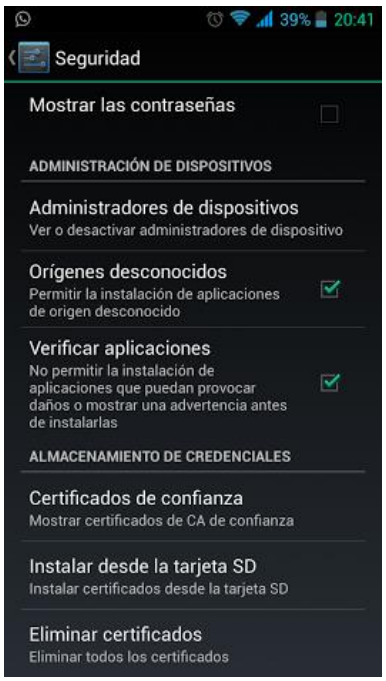


Figura 53. Permisos de ejecución Android

3. **Instalar la aplicación.** Cuando se realice el proceso anterior, ya se podrá proceder a la instalación del juego. Buscamos el archivo en la carpeta donde se haya almacenado y se procede finalmente a la ejecución del mismo.

## 7.2 Manual de usuario

A continuación se presentan los diagramas de navegación en la interfaz de la aplicación (Figura 54) y diagrama de uso de juego (Figura 55).

### 7.2.1 Diagrama de navegación de la aplicación

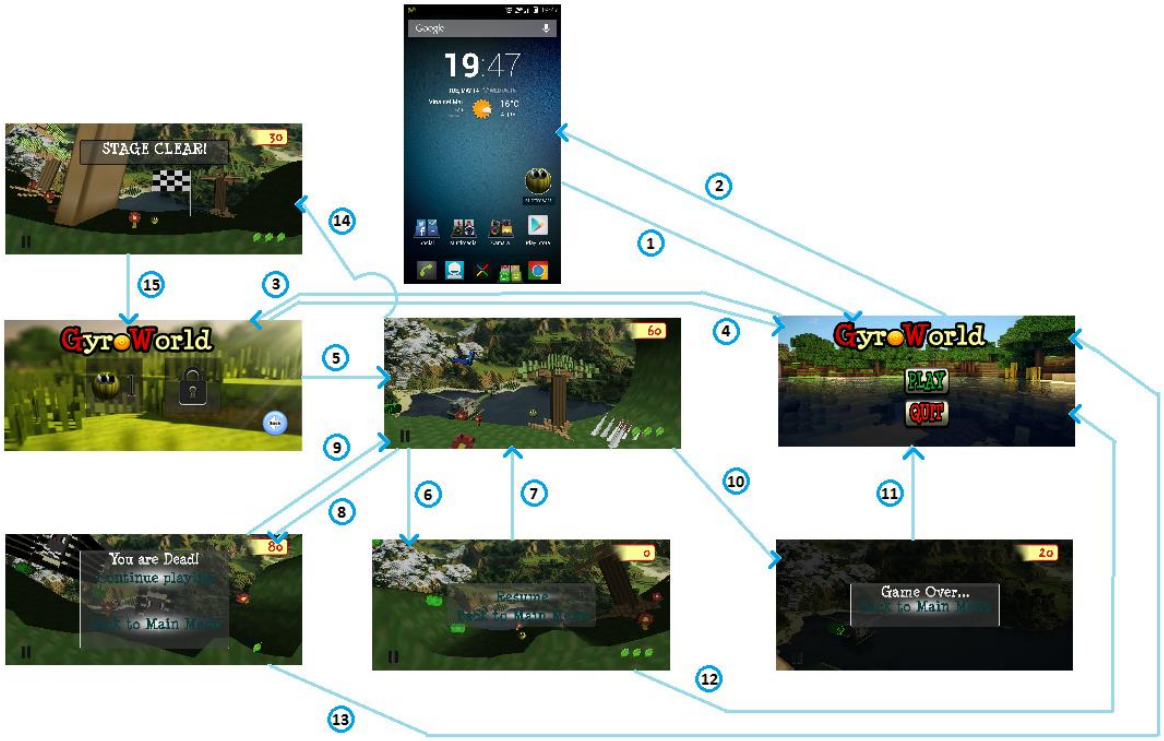


Figura 54. Diagrama de navegación de la interfaz

**Leyenda de navegación de la interfaz**

1. **Lanzamiento de la aplicación**. Se ejecuta pulsando en el icono de la aplicación.
2. **Salir de la aplicación**. Esta acción se realiza pulsando el botón *"QUIT"* en el menú principal.
3. **Pasar a menú selección de nivel**. Pulsando el botón *"PLAY"* en el menú principal se pasa al menú de selección de nivel.
4. **Volver a menú principal**. Pulsando el botón *"Back"* en el menú selección de nivel se vuelve al menú principal.
5. **Cargar nivel**. Pulsando en uno de los niveles disponibles, se carga dicho nivel.
6. **Pausar el juego**. Pulsando el botón de pausa, se pausa el juego.

7. **Terminar pausa**. Pulsando la opción *"Resume"* en el pop-up de pausa, reanudamos el juego.
8. **Muerte**. Al morir, se pierde una vida y se lanza el *"pop-up"* de muerte.
9. **Volver al juego después de morir**. Pulsando la opción *"Continue Playing"* en el *"pop-up"* de muerte.
10. **Fin de partida**. Al perder todas las vidas se lanza el *"pop-up"* de fin de partida.
11. **Vuelta al menú principal tras fin de partida**. Al pulsar la opción *"Back to Main Menu"* en el *"pop-up"* de fin de partida se vuelve al menú principal.
12. **Vuelta al menú principal desde menú de pausa**. Al pulsar la opción *"Back to Main Menu"* en el *"pop-up"* de pausa se vuelve al menú principal.
13. **Vuelta al menú principal tras muerte**. Al pulsar la opción "Back to Main Menu" en el *"pop-up"* de muerte se vuelve al menú principal.
14. **Nivel completado**. Al alcanzar la bandera en el nivel de juego, aparece el *"pop-up"* de nivel completado.
15. **Vuelta a selección de nivel tras completar nivel**. Al completar un nivel, al cabo de unos segundos tras salir el *"pop-up"* de nivel completado se cargará el menú de selección de nivel.

### 7.2.2 Diagrama de uso de juego


Figura 55. Diagrama de uso de juego

**Leyenda de interfaz de juego**

1. **Botón de pausa.** Se accede al pop-up de menú de pausa.
2. **Pantalla.** Al pulsar el personaje podrá realizar un salto normal (cuando está en contacto con el suelo) o un doble salto (salto extra cuando el personaje está en el aire).
3. **Puntuación.** Son los puntos acumulados del usuario en el nivel actual. Este marcador se incrementa al coger manzanas durante el juego.
4. **Vidas.** Son las vidas restantes del usuario. Cuando no quedan vidas restantes, se acaba la partida.
5. **Movimiento hacia la derecha.** Al inclinar el terminal móvil como se indica en la figura, el personaje se moverá a la derecha, con una velocidad proporcional a la inclinación.
6. **Movimiento hacia la izquierda.** Al inclinar el terminal móvil como se indica en la figura, el personaje se moverá a la izquierda, con una velocidad proporcional a la inclinación.

## 7.3 Diagrama de clases

En las figuras mostradas a continuación (Figuras 56 y 57) se presenta el diagrama de clases implementado en la aplicación.

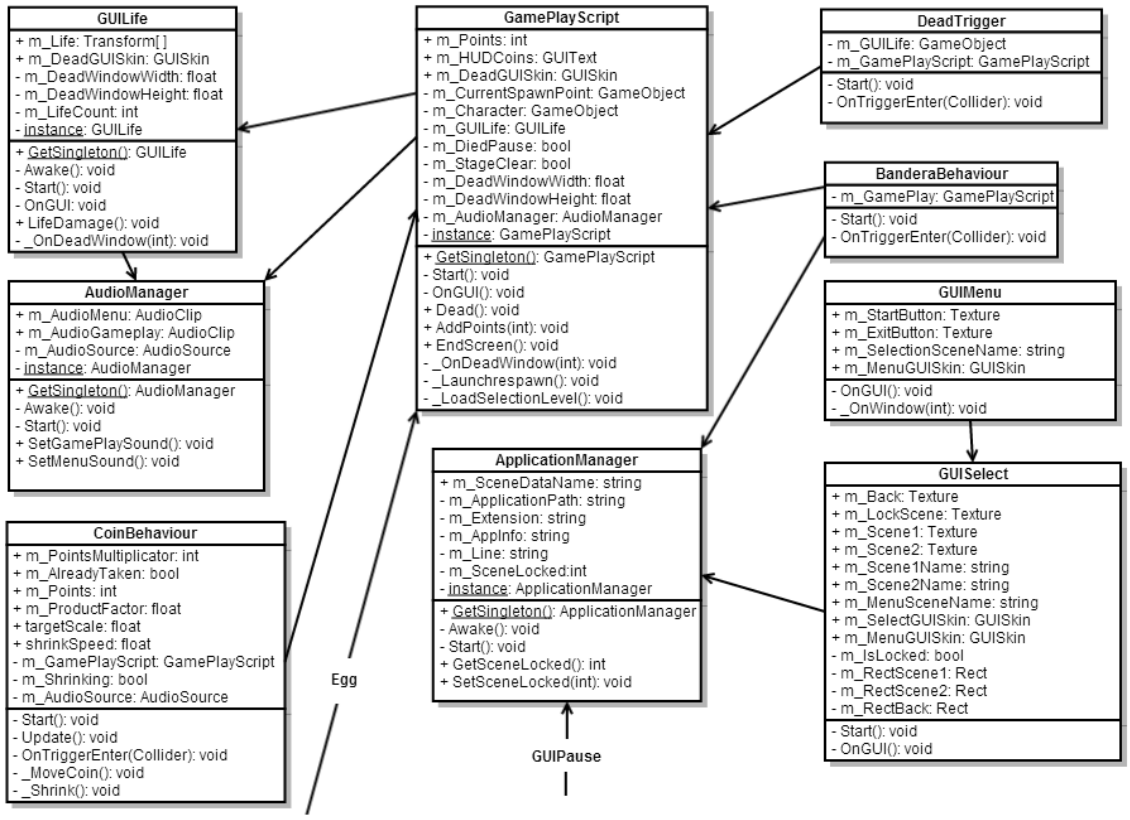


**Figura 56. Diagrama de clases (parte I)**

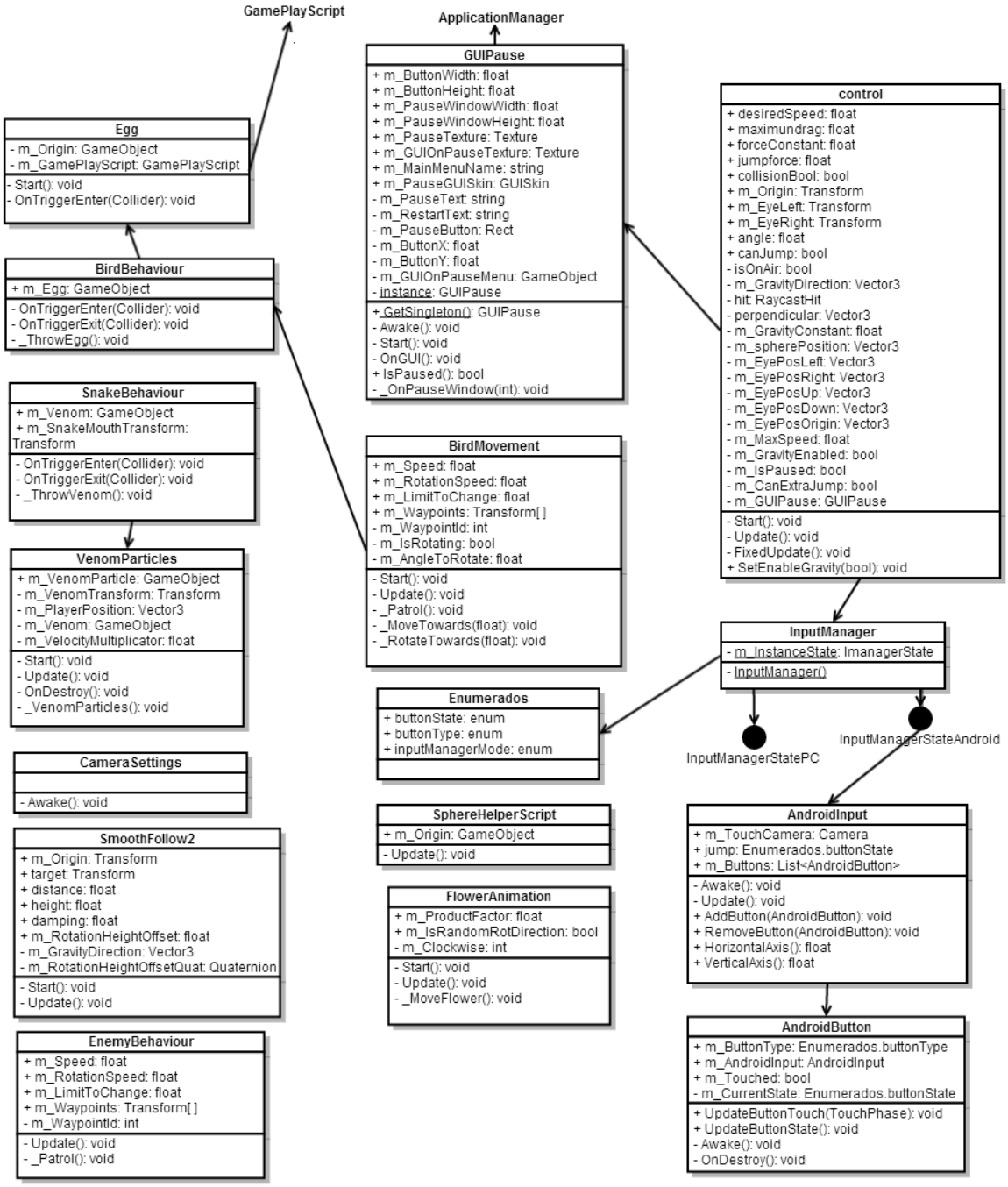


**Figura 57. Diagrama de clases (parte II)**

# Bibliografía

[1] Información general del motor Unity
http://unity3d.com/

[2] Juul J. A visual history of genres and platforms.
http://www.jesperjuul.net/ludologist/a-visual-history-of-genres-and-platforms

[3] OpenGL. Especificación del estándar de desarrollo
http://es.wikipedia.org/wiki/OpenGL

[4] Umbra, motor de oclusión y optimización de rendimiento usado en Unity
http://www.umbrasoftware.com/en/

[5] Beast, herramienta integrada en Unity para iluminación global y realización de mapas de iluminación precalculados
http://en.wikipedia.org/wiki/Beast_(lighting_software)

[6] Physx. Motor de física usado por Unity
http://www.nvidia.es/object/nvidia-physx-es.html

[7] Página oficial del motor Unreal
https://www.unrealengine.com/

[8] Enlace a página oficial del motor de desarrollo ShiVa3D
http://www.stonetrip.com/

[9] SandBox 3, herramienta integrada en CryEngine para gestión de creaciones multiplataforma en tiempo real
http://cryengine.com/features/tools

[10] Scrum, ejemplo de metodología ágil.
http://es.wikipedia.org/wiki/Scrum